Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues

Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
e-mail: aschaerf@dis.uniroma1.it

June 17, 1994
Author’s Address:
Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
E-mail: aschaerf@dis.uniroma1.it
# Contents

## Acknowledgements

**Abstract**  
11

1 **Introduction**  
1.1 History .................................................. 11  
1.2 Representing Knowledge with Concept Languages  .... 14  
1.3 Goal of the Thesis and Main Results .................... 19  
1.4 Structure of the Thesis ................................. 21

2 **General Framework**  
2.1 Syntax and Semantics of Concept Languages ............ 23

2.2 Concept-Based Knowledge Bases ........................ 27

2.2.1 Intensional Knowledge ............................... 27

2.2.2 Extensional Knowledge ............................... 29

2.2.3 Knowledge Bases ..................................... 30

2.3 Reasoning Services ...................................... 30

2.3.1 Special Cases ....................................... 31

2.4 Computational Complexity .............................. 32

2.4.1 Complexity Measures for Concept Languages ........ 32

3 **Reasoning Techniques and Previous Results** ....... 35

3.1 Tableaux Calculus ........................................ 35

3.1.1 Constraint Systems and Completion Rules ............ 36

3.1.2 Specialization of the Calculus ....................... 41

3.2 Previous Results ........................................ 43

3.2.1 Relations between the Reasoning Services ............ 43

3.2.2 Complexity of Concept Satisfiability and Subsumption . 44

3.2.3 Complexity of Consistency and Instance Checking ... 48
7.3.1 Decidability of Free TBoxes in $\mathcal{ALNN}$ .......................... 137
7.3.2 Free TBoxes versus General TBoxes ................................. 145
7.4 Weak Inclusions as Rules .............................................. 148

8 Design of a Powerful Concept-Based KR System 151
8.1 Definition of the System AI-K ........................................ 151
8.2 Example ................................................................. 152
8.2.1 Epistemic Operator in the Query Language .................... 152
8.2.2 Weakening the Inclusions ........................................ 158

9 Discussion and Conclusions ............................................. 161
9.1 Comments to the Results .............................................. 161
9.1.1 Complexity Analysis .............................................. 161
9.1.2 Expressive Representation Mechanisms ......................... 162
9.1.3 Complete (Intractable) Reasoning Procedures ................ 163
9.2 Open Problems ......................................................... 164
9.2.1 Missing Complexity Results ...................................... 165
9.2.2 Implementation ..................................................... 165
9.2.3 Restrictions ......................................................... 165
9.2.4 Extensions .......................................................... 166

References ............................................................... 167

List of Publications .................................................... 181
Acknowledgements

I wish to thank my tutor Maurizio Lenzerini, for the inspiration and the support necessary to the fulfillment of this work. I owe Maurizio a lot, he taught me how to do research and how to write a research paper.

I also would like to thank my coauthors: Martin Buchheit, Francesco Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. A special thank goes to Francesco, for many fruitful discussions that directly contributed to this thesis.

Giuseppe Attardi and Bernhard Nebel, as external referees of the Ph.D. Committee of the Dipartimento di Informatica e Sistemistica dell'Università di Roma “La Sapienza”, gave me many useful comments on the previous version of the thesis, helping me to improve it.

Paolo Atzeni and Pierluigi Crescenzi, as internal referees of the Ph.D. Committee, and Luigia Carlucci Aiello, as president of it, helped me in the fulfillment of the work. Paolo, in particular, gave me valuable advises on how to organize the work.

I am also in debt with Marco Schaefer, Enrico Franconi, and Diego Calvanese for useful comments on earlier drafts of some of the papers that compose the thesis.

I thank the Dipartimento di Informatica e Sistemistica of Università di Roma “La Sapienza”, for the support offered to me to accomplish this work. I also acknowledge Yoav Shoham for his hospitality at the Computer Science Department of Stanford University, where part of this research has been done.

I also thank the CNR project “Tasso”, and Alfonso Miola in particular, for providing me the SparcStation2 disco1, which turned out to be an essential tool for the accomplishment of the work.

This work has been supported by the ESPRIT Basic Research Action N.6810 (COMPULOG 2) and by the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council).

June 17, 1994
Andrea Schaefer
Abstract

The idea of developing knowledge representation systems based on a structured representation of knowledge was first pursued with Semantic Networks and Frames. Later, concept languages (also called terminological languages, or description logics) have been introduced with the aim of providing a simple and well-established first-order semantics to capture the meaning of the most popular features of the structured representations of knowledge.

The basic inference tasks to be carried out on concept-based knowledge bases are Subsumption ("is one concept more general than the other one?") and Instance Checking ("is an individual an instance of a concept?"). More complex query answering services, such as Retrieval ("find all the individuals that are instances of a concept"), can be reduced to a set of parallel Instance Checking tests, although the implementations may exploit suitable optimization strategies. In this thesis, we concentrate on Instance Checking, being understood that all the results on Instance Checking directly apply to more complex services.

First, we analyze Instance Checking, providing algorithms for its solution and results for its intrinsic complexity in the various concept languages. The central question we address is whether Instance Checking can be easily reduced to Subsumption.

To this aim, we present sound and complete methods for Instance Checking, and provide a complexity analysis of this problem, singling out languages for which Instance Checking is strictly harder than Subsumption, as well as languages for which Instance Checking can be solved by relying on Subsumption algorithms. This analysis highlights a new source of complexity in concept languages, which does not show up when checking Subsumption between concepts, and is due to the presence of individuals in the knowledge base. A practical implication of this fact is that any actual deduction procedure for reasoning in concept-based systems cannot be based solely on Subsumption, but has to embed some reasoning mechanisms that are not easily reducible to Subsumption.

Such situation leads us to reconsider the semantics of knowledge bases and to address the various problems associated with it. Based on this, we consider
concept constructors that allow to refer to individuals in the concept language. Constructors of that kind are generally used in concept-based systems (e.g., ONE-OF in CLASSIC) although the semantics employed for their treatment is a non-standard one. We perform a theoretical investigation on the use of those constructors with respect to the standard first-order semantics, which shows various semantic issues and new sources of complexity.

Further, we study the possibility of modifying the Instance Checking problem to the case in which the knowledge base and the concept representing the query are expressed in different languages. The aim of this is to gain expressivity of the query language without giving up the tractability of reasoning. Further, we introduce an epistemic operator that makes the query facilities to go beyond the first-order setting of concept languages.

Such an operator allows the user to express queries that address both aspects of the external world and aspects of the knowledge base state. We show that epistemic queries may have both a natural interpretation and good computational properties.

We also show how the epistemic operator gives us the ability to formalize some procedural mechanism usually employed in actual systems, namely integrity constraints and trigger rules. We also discuss the complexity of reasoning with the most general form of statements about concepts (called free TBox), which allow us to express inclusions between general concepts, and terminological cycles as a particular case.

Finally, we exploit all the results obtained in order to come up with an example of a concept-based system whose capabilities go beyond actual systems. Such system is meant to capture and precisely formalize the various features of early knowledge representation tools, embedding them in a concept-based framework that is still based on first-order logic, but goes beyond it and overcomes some of the limitations of first-order logic as a representational tool.
Chapter 1

Introduction

1.1 History

The representation of knowledge is one of the central issues in the field of Artificial Intelligence (AI). For this reason, Knowledge Representation (KR) has become a research field itself [Lev86], and a large amount of work has been done on it since its beginnings.

The birth of KR cannot be precisely characterized. One of the earliest KR tool has been the Semantic Networks. The first Semantic Network used in a computer program was Nude, created by Richens in 1956 [Ric56]. However, a very influential work has been done by Quillian in 1967 [Qui67], with his Semantic Memory Model, which turned out to be an important step toward the current setting of Semantic Networks.

Semantic Networks represent knowledge under the form of a labeled directed graph. Specifically, each node is associated with a concept, and the arcs represent the various relations between concepts.

As pointed out by Woods in [Woo75] and Brachman in [Bra79], despite their name, early Semantic Networks suffered from the drawback that they did not have a clear semantics.

The ambiguity in the semantics arises from the fact that in Semantic Networks arcs can represent different kind of relations between the nodes. In particular, Woods identified two main types of arcs: Those representing intensional knowledge and those representing extensional one.

An arc that contributes to the definition of a concept carries intensional knowledge. For example, an arc between the node Person and the node Integer labeled with the name Age, represents the age of a person and it is part of the definition of the concept person. On the other hand, an arc from the node Mary to the node John, labeled with the name Hit, asserts the fact that Mary hit John, which is part of the extensional knowledge.
Such ambiguity is not acceptable for an AI application. In fact, as argued by Hayes [Hay79], what distinguishes a knowledge base in an AI application from any other data in an application in Computer Science is that the knowledge base is given a certain a priori semantics, or, in other words, “it carries its meaning”. In particular, one can characterize a representational language as one that has a semantic theory, by which Hayes means an account (more or less formal) of how expressions of the language relate to the objects of the world about which the language claims to express knowledge. Conversely, the lack of a precise semantics makes reasoning difficult to automatize in a way that satisfies the user in the general case.

Another seminal contribution in KR has been given by Minsky in 1975 (published in [Min81]) with his proposal to use “frames” as a representation tool. A frame usually represents a concept (or a class) and it is defined by an identifier, and a number of data elements called slots, each of which corresponds to an attribute that members of its class can have. Each slot contains information about the corresponding attribute, such as default values, restrictions on possible fillers, attached procedures or methods for computing values when needed, and procedures for propagating side effects when the slot is filled. The values of the attributes are either elements of a concrete domain (e.g. integers, strings) or identifiers of other frames. A frame can also represent a single individual, in this case it is related with the attribute instance-of to the frame representing the class of which the individual is an instance.

However, early frame-based systems suffer from the very same problem highlighted above for Semantic Networks. In fact, the semantics of frames was not completely defined, and in particular the distinction between the various forms of link was not clear in frame systems, too.

The above discussion explains the rise of a line of research that tries to give Semantic Networks and frames a formal semantics. The first contribution in this field has been given by Brachman in 1977 with his Ph.D. thesis, which led to the development of the system KL-ONE [BS85].

Subsequently, starting in the Eighties, the research has gone further in this direction by providing the systems with an explicit model-theoretic semantics (see for example the system OMEGA [AS81]). More recently, concept-based KR systems (or terminological systems, or description logics) have been proposed as successors of KL-ONE in which the formal semantics is the Tarskian semantics for first order logic (FOL).

The role of FOL in KR has been always discussed in the KR community. In fact, although there is now a reasonable agreement on the need of a formal logic at least for the definition of formal semantics of a KR system, it is not yet clear whether such logic should be FOL.

Early KR formalisms generally exhibited a behavior that could hardly be captured using FOL. In fact, they usually included some procedural mech-
anism that cannot be expressed in a declarative language such as FOL. In addition, they require some form of nonmonotonic reasoning, whereas FOL is strictly monotonic. However, the nonmonotonic aspects of KR systems were not completely clear, and only recently nonmonotonic logics, that could formalize those aspects, started to be understood.

The attempt done with concept-based systems is to reconstruct logically, using a fragment of FOL, Semantic Networks and frames. The procedural and nonmonotonic aspects have been left out, hoping to reintroduce them later on, once they have been better understood, and logical tools to formalize them have been developed. Following this line, starting with KL-ONE [BS85], a family of concept-based systems have been proposed and studied, whose semantics is derived from the corresponding one for FOL. Such family includes systems like KRYPTON [BPL85], NIKL [KBR86], BACK [QK90], LOOM [MB87], CLASSIC [BBM89a], KRIS [BH91b], and others (see [Ric91, WS92]).

Only very recently, the nonmonotonic aspect of KR have been considered in a principled way, and we have seen some attempts to formalize them, based on some forms of nonmonotonic logic (see, for example, [BH92, QR92, PN93, BH93, Str93, PZ93]).

An attempt to provide a semantic characterization of some procedural mechanisms considered in several frame-based systems, such as trigger rules, has been proposed in [DLN+92, DLN+93] and it is reported in Chapter 7 of this thesis.

In concept-based systems the use of FOL goes beyond the simple import of the formal semantics. In fact, such systems also employ the reasoning tools designed for FOL.

The reason that makes FOL not particularly attractive as a proof system is its computational complexity. In fact, it is well known that entailment in FOL is an undecidable problem. However, in concept-based systems, the proof-theory of FOL can be specialized to the specific fragment they describe, and the reasoning algorithms can be made specially suitable for it.

Summarizing, the idea of concept-based systems can be explained as an attempt to combine the following features:

1. the formal semantics of FOL,
2. the representational capabilities of Semantic Networks and frames,
3. specialized (and efficient) reasoning algorithms.

The challenge of such an attempt is to identify the exact fragment of FOL that can capture the features needed for representation purposes, and can still allow for the design of efficient reasoning algorithms. To this aim, several systems have been considered and their expressiveness and complexity have been studied.
As we have already mentioned, such reconstruction cannot be completely faithful, since it has those limits related to nonmonotonic aspects of reasoning. However, a good formalization of the monotonic part can help in the understanding of the overall system, and later such limits can be overcome.

One main advantage of concept-based systems over semantic networks and frame-based systems is given by the “Automatic Classification”. In fact, in every Semantic Network and frame-based system there is a hierarchy based on the subclass relation. In concept-based systems, such hierarchy can be automatically computed, instead of being given by the designer of the knowledge base. Such hierarchical structure of concepts is determined by the subsumption relation, which is interpreted as set containment.

The possibility of recognizing implicit hierarchical relations, through the computation of the subsumption relation, is obviously a byproduct of having introduced a formal semantics for the interpretation of concepts.

The importance of hierarchical reasoning is testified by recent work done with the aim of introducing it also in Semantic and Object-Oriented Data Models. In fact, various attempts have been done to import techniques from KR into the Database field to develop suitable subsumption algorithms for classes (or objects) in those Data Models (see e.g., [BGN89, NP91, BS92, BN93, BJNS94, CLN94, CL94]). In that framework, the detection of subsumption can be profitable exploited for various purposes, including, for example, query optimization.

Another advantage of having a formal semantics is that it makes possible the identification of related reasoning tasks in others research areas. In fact, several correspondences have been established between reasoning problems in concept-based systems and problems in other areas of Computer Science, e.g. automata theory, dynamic logics, unification grammars (see [NS89, Sch91, NS91]). Such correspondences give the possibility of a cross-fertilization and the import of known results in both directions.

1.2 Representing Knowledge with Concept Languages

A general characteristic of concept-based systems is that the knowledge base is made up of two different components. Informally speaking, the first is a general schema concerning the classes of individuals to be represented, their general properties and mutual relationships, while the second is a (partial) instantiation of this schema, containing assertions relating either individuals to classes, or individuals to each other.

This characteristic, typical of many actual systems, is inherited from the seminal one KL-ONE [BS85], and is shared also by several proposals of Database
models such as Abrial's [Abt74], CANDIDE [BGN89], and TAXIS [MBW80].

Retrieving information in actual concept-based knowledge bases is a deductive process involving both the schema (TBox) and its instantiation (ABox). In fact, the TBox is not just a set of constraints on possible ABoxes, but contains intensional information about classes. This information is taken into account when answering queries to the knowledge base.

The core of a concept-based KR system is the concept language. In concept languages, concepts are used to represent classes as sets of individuals, and roles are binary relations used to specify their properties or attributes. Typically, concepts are denoted by expressions formed by means of special constructors, and are structured into hierarchies determined by the properties associated with them.

For example, the concept expression Parent \( \cap \) Male \( \forall \) CHILD.Male denotes the class of fathers (male parents) all of whose children are male. The symbol \( \cap \) denotes concept conjunction and is interpreted as set intersection. The expression \( \forall \) CHILD.Male denotes the set of individuals whose children are all male, thus specifying a property which relates to other individuals through the role CHILD. Expressions of the form \( \forall R.C \) are called universal role quantifications. Instead, \( \exists \) CHILD.Male is an example of existential role quantification, denoting the set of individuals with a male child. Existential role quantification is sometimes unqualified and written \( \exists \) CHILD, in which case it denotes the set of individuals with a child. The basic language that we consider (called \( \mathcal{FO}^- \), in [BL84]) includes concept conjunction, universal role quantification and unqualified existential role quantification. More powerful languages are then defined by adding other constructors to this basic language.

In a concept-based knowledge base, the TBox is a set of concept definitions and specifications, whereas the ABox is a set of assertions stating membership relations between individuals and concept, or pair of individuals and roles. The semantics employed is an open world one, which means that no restrictions are imposed on the possible interpretations of the knowledge base.

The basic inference tasks to be carried out on concept-based knowledge bases are the following ones:

1. **Concept Satisfiability:** given a knowledge base and a concept, does there exist at least one model of the knowledge base assigning a non-empty extension to the concept? This is important not only to rule out meaningless concepts in the knowledge base design phase, but also in processing the user's queries, to eliminate parts of a query which cannot contribute to the answer.

2. **Subsumption:** given a knowledge base and two concepts, is one concept more general than the other one in any model of the knowledge base?
Subsumption detects implicit dependencies among the concepts in the knowledge base.

3. **Consistency**: are an ABox and a TBox consistent with each other? That is, does the knowledge base admit a model? A positive answer is useful in the validation phase, while the negative answer can be used to make inferences in refutation-style. The latter is precisely the approach taken in this thesis.

4. **Instance Checking**: given a knowledge base, an individual and a concept, is the individual an instance of the concept in any model of the knowledge base?

The above services can be precisely characterized once the concept-based system is given a semantics (see Chapter 2), which defines models of the knowledge base and gives a meaning to expressions in the knowledge base. Once the problems are formalized, one can start both a theoretical analysis of them, and—maybe independently—a search for reasoning procedures accomplishing the tasks. Completeness and correctness of procedures can be judged with respect to the formal statements of the problems.

Other (more complex) reasoning services have been considered in the literature on concept languages. A list of them can be found in [BBH+91]. We mention the following two:

5. **Retrieval** (or **Query Answering**): Given a concept, find all the objects occurring in the knowledge base that are instances of the concept.

6. **Realization**: Given an individual occurring in the knowledge base, find the most specific concepts, w.r.t. the subsumption relation, of which the individual is an instance.

It is easy to see that such tasks can in principle be reduced to a set of parallel Instance Checking tests, although the implementations may exploit suitable optimization strategies. For this reason, we mostly concentrate on Instance Checking, being understood that all the results on Instance Checking directly apply to the more complex reasoning tasks. In addition, we show in Chapter 2 that the services 1-3 can be reduced to Instance Checking, too. Therefore, Instance Checking can be considered as the central reasoning task for drawing conclusions in a knowledge base.

Up to now, all the proposed systems give incomplete procedures for solving Instance Checking, except for KRS and CLASSIC (the latter however is complete only w.r.t. a deviant semantics for the treatment of individuals, see Chapter 5). That is, some inferences are missed, in some cases without a
precise semantical characterization of which ones are. If the designer or the user needs (more) complete reasoning, she/he must either write programs in a suitable programming language (as in the Database proposal of Abrial, and in TAXIS), or define appropriate inference rules completing the inference capabilities of the system (as in BACK, LOOM, and CLASSIC). From the theoretical point of view, for several systems (e.g. LOOM) it is not even known if complete procedures can ever exist—i.e., the decidability of the corresponding problems is not known.

Recent research on the computational complexity of Subsumption had an influence in many concept-based systems on the choice for incomplete procedures. The research started with [BL84], which analyzed complexity of Subsumption between pure concept expressions, abstracting from knowledge bases (later in the thesis we call this problem Pure Subsumption). The motivation for focusing on such a small problem was that Pure Subsumption is a fundamental inference in any concept-based system.

It turned out that Pure Subsumption is tractable (i.e. polynomial-time solvable) for simple languages, and intractable (i.e. NP-hard or coNP-hard) for slight extensions of such languages, as subsequent research definitely confirmed [Neb88, DLNN91a, DLNN91b, SS91, DHL+92]. Also, beyond computational complexity, Pure Subsumption was proved undecidable in KL-ONE [Sch89b], NIKL [Pat89] and UT [Sch88].

Note that extending the concept language results in enhancing the system’s expressiveness, therefore the result of that research could be summarized as: The more a system is expressive, the higher is the computational complexity of reasoning—as Levesque first noted [Lev84]. This result has been interpreted in two different ways, leading to two different concept-based system design philosophies:

1. ‘General-purpose languages for concept-based systems are intractable, or even undecidable, and tractable languages are not expressive enough to be of practical interest’. Following this interpretation, in several systems (such as NIKL, LOOM and BACK) incomplete procedures for Pure Subsumption are considered satisfactory (e.g. see [MB92] for LOOM). Once completeness is abandoned for this basic subproblem, completeness of overall reasoning procedures is not an issue anymore; but other issues arise, such as how to compare incomplete procedures [HKNP92], and how to judge a procedure “complete enough” [Mac91]. As a practical tool, inference rules can be used in such systems to achieve the expected behavior of the knowledge base w.r.t. the information contained in it.

2. ‘A concept-based system is (by definition) general-purpose, hence it must provide tractable and complete reasoning to a user’. Following this line, other systems (such as KRYPTON and CLASSIC) provide limited tractable
languages for expressing concepts, following the “small-can-be-beautiful” approach (see [Pat84]). The gap between what is expressible in the system and what is needed to be expressed for the application is then filled by the user, by a (sort of) programming with inference rules. Of course, the usual problems present in program development and debugging arise [McG92].

What is common to both approaches is that a user must cope with incomplete reasoning. The difference is that in the former approach, the burden of regaining useful yet missed inferences is mostly left to the developers of the concept-based system (and the user is supposed to specify what is “complete enough”), while in the latter this is mainly left to the user. These are perfectly reasonable approaches in a practical context, where incomplete procedures and specialized programs are often used to deal with intractable problems. In our opinion incomplete procedures are just a provisional answer to the problem—the best possible up to now. In order to improve on such an answer, a theoretical analysis of Instance Checking is to be done.

In addition, we think that the expressiveness of actual systems should be enhanced making terminological cycles (see [Neb90a, Chapter 5]) available in concept-based systems. Such a feature is of undoubtable practical interest [Mac92], yet most present systems can only approximate cycles, by using forward inference rules. An example of concept-based system that implements cycles is K-rep [MDW91]. However, K-rep includes only TBox statements, without considering the ABox. In our opinion, in order to make terminological cycles fully available in complete systems, a theoretical investigation is still needed.

Previous theoretical work on cycles was done in [Baa90b, Baa90a, BBH+90, Neb90a, Neb91, Sch91], but considering knowledge bases formed by the TBox alone. Moreover, these approaches do not deal with number restrictions (except for [Neb90a, Section 5.3.5]), which are a basic feature already provided by concept-based systems, and the techniques used do not seem easily extensible to reasoning with ABoxes.

Previous results in the database field, are in general not directly importable in the framework of concept languages. In fact, as pointed out by Reiter in [Rei84] for the relational model, a database represents a single model, whereas a concept-based knowledge base is a FO theory, admitting an arbitrary (possibly infinite) number of models.

The possibility of having multiple models in databases is taken into account by the use of null value. Reiter and Vardi in [Rei84, Var86] worked on the complexity of querying a relational database with null values. Their approach is to view it as a particular FO theory, which Vardi called logical database. The results in [Rei84, Var86] show that querying a logical database is generally
harder than querying a pure relational database (physical databases). However, the setting used in those papers is different from ours (e.g. use of Closed World Assumption) therefore their results are not directly applicable in our framework. However, those results help in understanding why reasoning with concept-based knowledge bases is generally intractable whereas using a full FO language (the relational calculus) for querying a relational (physical) database is polynomial (w.r.t. the data complexity).

1.3 Goal of the Thesis and Main Results

The thesis is centered on the Instance Checking problem. To this regard, we fix several interrelated goals.

First, we intend to analyze Instance Checking, providing algorithms for its solution and results for its intrinsic complexity in the various concept languages. One question we address is whether Instance Checking can be easily reduced to Subsumption. The problem of establishing the relationship between Subsumption and Instance Checking is not only of theoretical importance, but has practical implications in the implementation of the deduction services for concept-based systems. In fact, a careful analysis of the relationship between Subsumption and Instance Checking is lacking, although most of the existing systems (e.g. LOOM, CLASSIC, BACK) provide implementations of reasoning services based on Subsumption algorithms.

We address the above mentioned problem by presenting sound and complete methods for Instance Checking, and providing a complexity analysis of this problem, thus singling out cases where Instance Checking is strictly harder than Subsumption and cases where Instance Checking can be solved by relying on Subsumption algorithms. We also present a general technique for checking the consistency of a knowledge base, which can be seen as a subproblem of Instance Checking. In our analysis, we consider several languages where Subsumption belongs to different complexity classes, obtained from the simple language $\mathcal{FL}^-$, mentioned above, by adding constructors for concept disjunction, qualified existential quantification on roles, negation, number restrictions, and role conjunction.

Our study is carried on using the reasoning technique based on constraint systems introduced in [DLNN91a, DLNN91b, SS91], which can be regarded as a refinement of the deduction method of tableaux for FOL (see [BM77]). In particular, we extend the technique to the treatment of individuals.

This analysis singles out a new source of complexity in concept languages, which does not show up when checking Subsumption between concepts, and is due to the presence of individuals in the knowledge base. A practical implication of this fact is that any actual deduction procedure for reasoning in
concept-based systems cannot be based solely on Subsumption, but has to embed some reasoning mechanisms that are not easily reducible to Subsumption, at least when non-trivial languages are used.

Another outcome of our analysis is that there are cases in which the reasoning process underlying this source of complexity is not in the intuition of the user, as pointed out in some examples. Conversely, there are cases in which this kind of reasoning seems to agree with the intuition and therefore it must be taken into account.

Such situation leads us to reconsider the semantics of knowledge bases and to address the various problems associated with it. Based on this, our second goal is the formalization of concept constructors that allow to refer to individuals in the concept language. Constructors of that kind are generally used in concept-based systems (e.g., ONE-OF in CLASSIC) although the semantics employed for their treatment is a non-standard one. A theoretical investigation on the use of those constructors with respect to the standard FO semantics is still missing. In this thesis, we perform such analysis highlighting the various semantic issues related to the use of individuals in the concept language and we study the computational complexity of the various reasoning services in this case. The outcome is that reasoning with individuals in the concept language is generally hard. The intuitive reason is that they break the strict separation between the intensional knowledge and the extensional one. Such separation of the two components limits the expressive power of the overall system. In order to recover some of the expressive power, a limited mixing of the two components and/or a more complex interaction between them is needed.

Our third goal is to study the possibility of modifying the Instance Checking problem to the case in which the knowledge base and the concept representing the query are expressed in different languages. The aim of this is to gain expressivity of the query language without giving up the tractability of reasoning. Further, we introduce the epistemic operator that makes the query facilities to go beyond the FO setting of concept languages.

Such an operator allows the user to express queries that address both aspects of the external world and aspects of the knowledge base state. We will see that epistemic queries may have both a natural interpretation and good computational properties.

The fourth goal of the thesis is the introduction of various features that are of practical interest in concept-based systems, to formalize them, and study their complexity. In particular, we show how, the epistemic operator gives the ability to formalize some procedural mechanism usually employed in actual systems, namely integrity constraints and trigger rules. We also discuss the complexity of reasoning with the most general form of TBox mechanism (called free TBox), which allow to express inclusion statement between general
concepts, and terminological cycles as a particular case.

Finally, we exploit all the results obtained in order to come up with an example for a concept-based system whose capabilities go beyond actual systems. Such system should capture and precisely formalize the various features of early KR tools, embedding them in a concept-based framework that is still based on FOL, but goes beyond it and overcomes some of the limitations of FOL as a representational tool.

1.4 Structure of the Thesis

The thesis is organized as follows:

- Chapter 2 provides a formal introduction to concept languages, their use in the construction of knowledge bases, and the reasoning services provided by knowledge bases.

- Chapter 3 presents the reasoning techniques employed in the thesis and reviews previous results on the computational complexity of deduction problems for concept languages. Such results are also discussed and interpreted identifying the various sources of complexity.

- Chapter 4 deals with the problems of Consistency and Instance Checking. It shows that the former can be solved by exploiting the algorithms for Concept Satisfiability. Regarding the latter, it contains the complexity analysis of Instance Checking, providing both algorithms for computing Instance Checking via Subsumption, and hardness proofs, in those cases where Instance Checking belongs to a higher complexity class with respect to Subsumption.

- In Chapter 5, we investigate on the consequences of introducing individuals in the concept language. In particular, we give a survey of the various issues associated with their use, we briefly describe some of the strategies chosen by the implementors of the actual systems in order to deal with them, and we present several complexity results.

- Chapter 6 generalize the Instance Checking problem allowing for two different languages for asserting facts and querying the knowledge base. Moreover, we introduce the epistemic operator that provides the concept languages with powerful and sophisticated querying capabilities. We also show how the use of the epistemic operator allows the overcome some semantic issues that arise in concept languages.

- In Chapter 7, we provide a formal account of several mechanisms generally used in implemented systems, such as trigger rules, that are not
directly expressible in first order logic and we also discuss the free TBox representational mechanisms.

- Chapter 8 provides an example of a concept-based system, which arises from the results of previous chapters. We also show an extensive example of knowledge base and several queries directed to it.

- Conclusions are drawn in Chapter 9.
Chapter 2

General Framework

In this chapter we present the basic notions regarding concept languages, knowledge bases built up using concept languages, and reasoning services that must be provided for inferring information from such knowledge bases.

2.1 Syntax and Semantics of Concept Languages

A concept language is composed by the symbols taken from the following alphabets:

- CN: the alphabet of Concept Names.
- RN: the alphabet of Role Names.
- IN: the alphabet of Individual Names (or Individuals).

Besides concept, role, and individual names, the alphabet of concept languages includes a number of constructors that permit the formation of concept expressions and role expressions. The set of constructors for concepts expressions and role expressions considered in this thesis are listed in Tables 2.1 and 2.2, respectively. We do not claim the list of the constructors in Tables 2.1 and 2.2 to be exhaustive. The description of other constructors can be found in [BBH+91, PS93].

Concept names are denoted with the letters $A, B$, role names with $P$, and individuals names with $a, b$, possibly with subscripts. Concept expressions and role expressions (or simply concepts and roles) are denoted respectively with the letters $C, D$ and $Q, R$.

Given a concept $C$, we call subconcept of $C$ any substring of $C$ (including $C$ itself) that is a concept, according to the syntax rules. Notice that, denoting with $|C|$ the size of $C$, the number of subconcepts of $C$ is bounded by $|C|$.
<table>
<thead>
<tr>
<th>Constructor Name</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept name</td>
<td>(A)</td>
<td>(A^2 \subseteq \Delta^2)</td>
</tr>
<tr>
<td>top</td>
<td>(\top)</td>
<td>(\Delta^2)</td>
</tr>
<tr>
<td>bottom</td>
<td>(\bot)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>conjunction</td>
<td>(C \sqcap D)</td>
<td>(C^2 \cap D^2)</td>
</tr>
<tr>
<td>disjunction ((\cup))</td>
<td>(C \sqcup D)</td>
<td>(C^2 \cup D^2)</td>
</tr>
<tr>
<td>negation ((\neg))</td>
<td>(\neg C)</td>
<td>(\Delta^2 \setminus C^2)</td>
</tr>
<tr>
<td>universal quantification</td>
<td>(\forall R.C)</td>
<td>({d_1 \mid \forall d_2 : (d_1, d_2) \in R^2 \rightarrow d_2 \in C^2})</td>
</tr>
<tr>
<td>existential quantification ((\exists))</td>
<td>(\exists R.C)</td>
<td>({d_1 \mid \exists d_2 : (d_1, d_2) \in R^2 \land d_2 \in C^2})</td>
</tr>
<tr>
<td>number restrictions ((N))</td>
<td>((\geq n R))</td>
<td>({d_1 \mid \exists d_2 : (d_1, d_2) \in R^2 \geq n})</td>
</tr>
<tr>
<td>collection of individuals ((O))</td>
<td>({a_1, \ldots, a_n})</td>
<td>({a_1^<em>, \ldots, a_n^</em>})</td>
</tr>
<tr>
<td>role filler ((\tau))</td>
<td>(R : a)</td>
<td>({d \mid (d, a^*) \in R^2})</td>
</tr>
</tbody>
</table>

Figure 2.1: Syntax and semantics of the concept forming constructors.

<table>
<thead>
<tr>
<th>Constructor Name</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>role name</td>
<td>(P)</td>
<td>(P^2 \subseteq \Delta^2 \times \Delta^2)</td>
</tr>
<tr>
<td>role conjunction ((\sqcap))</td>
<td>(Q \sqcap R)</td>
<td>(Q^2 \cap R^2)</td>
</tr>
<tr>
<td>role chain ((\circ))</td>
<td>(Q \circ R)</td>
<td>({(d_1, d_2) \mid \exists d_3 : (d_1, d_3) \in Q^2 \land (d_3, d_2) \in R^2})</td>
</tr>
<tr>
<td>inverse role ((\check{\tau}))</td>
<td>(R^{-1})</td>
<td>({(d_1, d_2) \mid (d_2, d_1) \in R^2})</td>
</tr>
</tbody>
</table>

Figure 2.2: Syntax and semantics of the role forming constructors.

The various languages differ from each other based on the set of constructors they allow. A language \(\mathcal{L}\) is uniquely identified by the set of its constructors. A concept (resp. role) obtained using the constructors of \(\mathcal{L}\) is called an \(\mathcal{L}\)-concept (resp. \(\mathcal{L}\)-role).

We say that a language \(\mathcal{L}_1\) is a superlanguages (resp. sublanguage) of a language \(\mathcal{L}_2\) if the set of constructors of \(\mathcal{L}_1\) includes (resp. is included in) the one of \(\mathcal{L}_2\). An \(\mathcal{L}\)-concept is obviously also a concept of any superlanguage of \(\mathcal{L}\).

In order to give a name to each language, a letter should be associated with each constructors and a language could be identified by the string composed of its constructors. However, we do not consider all the possible languages, which are exponentially many; instead we take into account almost only languages that have been already employed in the literature and are considered important from both practical and theoretical purposes. For naming those languages, we follow a terminology that was already introduced in the literature [SS91, DLN91a] and is quite broadly accepted. Following such terminology (explained below), an identifying letter is associated to a subset of the constructors, as shown in Tables 2.1 and 2.2.
In agreement with [BL84], we call $\mathcal{F}^-$ the language including universal quantification, conjunction and unqualified existential quantification\(^1\) ($\exists R. T$); and $\mathcal{A}C$ (as named in [SS91]) the language obtained from $\mathcal{F}^-$ adding $T, \perp$ and negation of concept names, i.e. negation only of the form $\neg A$.

Following the convention, the superlanguages of $\mathcal{A}C$ will be identified with a string of the form

$$\mathcal{A}C[\mathcal{E}][U][C][\mathcal{R}][\mathcal{N}][\mathcal{H}][\mathcal{I}][\mathcal{O}][\mathcal{B}]$$

indicating which constructors are allowed in the language besides those of $\mathcal{A}C$.

In a similar way, languages that are superlanguages of $\mathcal{F}^-$ (but not of $\mathcal{A}C$) will be identified by a string of the form $\mathcal{F}[\mathcal{E}][U][C][\mathcal{R}][\mathcal{N}][\mathcal{H}][\mathcal{I}][\mathcal{O}][\mathcal{B}]$.

Almost all the languages we consider in this thesis are superlanguages of $\mathcal{F}^-$ (and most of them are superlanguages of $\mathcal{A}C$, too), therefore all of them can be expressed with the above convention.

An interpretation $I = (\Delta^I, \tau)$ consists of a nonempty set $\Delta^I$ (the domain of $I$) and a function $\tau$ (the interpretation function of $I$) that maps every concept to a subset of $\Delta^I$, every role to a subset of $\Delta^I \times \Delta^I$, and every individual to an element of $\Delta^I$. Complex concepts and roles are interpreted according to the semantics given in Tables 2.1 and 2.2, respectively. Individuals are assumed to denote different elements in every interpretation, i.e. the interpretation of individuals is done in such a way that $a^I \in \Delta^I$ for each individual $a \in IN$ and $a^I \neq b^I$ if $a \neq b$. This property is called Unique Name Assumption and is usually assumed in the database world.

An interpretation $I$ is a model for a concept $C$ if $C^I$ is nonempty. A concept is satisfiable if it has a model and unsatisfiable otherwise. We say that $C$ is subsumed by $D$ if $C^I \subseteq D^I$ for every interpretation $I$, and $C$ is equivalent to $D$, written $C \equiv D$, if $C^I = D^I$ for every interpretation $I$.

It is important to observe that, according to the given semantics, the above constructors are not all independent of each other. More precisely, given a languages $L_1$ and a constructor $\Phi$, we say that $L_1$ simulates $\Phi$ if for every concept expressible in the language obtained adding $\Phi$ to $L_1$ there exists an equivalent one expressible in $L_1$.

For example, the language $\mathcal{A}E\mathcal{O}$ simulates $B$. In fact, due to the following equivalence

$$R: a \equiv \exists R.\{a\}$$

it is possible to eliminate, preserving the equivalence, all the occurrences of $B$ from an $\mathcal{A}E\mathcal{O}B$-concept, obtaining an equivalent $\mathcal{A}E\mathcal{O}$-concept.

\(^1\)the construct $\exists R. T$, also written $\exists R$, denotes the set of objects $d_1$ such that there exists an object $d_2$ related to $d_1$ by means of the role $R$. The existential quantification is unqualified in the sense that no condition is stated to $d_2$ other than its existence.
Moreover, as we show in Chapter 3, any concept containing general negation can be rewritten in an equivalent one containing negation only of concept names. In fact, the \( \neg \) sign can be "pushed" inside until we obtain only negation of the form \( \neg A \). For this reason, the constructor \( \mathcal{C} \) can be simulated by the language \( \mathcal{ALEU} \). In a similar way, we can also show that \( \mathcal{ALE} \) simulates \( \mathcal{E} \) and \( \mathcal{U} \).

As another example of simulation, the following two equivalences show that \( \mathcal{ALE} \) simulates \( \mathcal{H} \).

\[
\forall (Q \circ R).C \equiv \forall Q.\forall R.C \\
\exists (Q \circ R).C \equiv \exists Q.\exists R.C
\]

Notice that the same is not necessarily true for languages more expressive than \( \mathcal{ALE} \), in particular those languages including other role constructors. For example, we do not obtain that \( \mathcal{ALER} \) simulates \( \mathcal{H} \). This is because the above equivalences do not give us a way to simulate complex roles involving nested conjunction and chains, like for instance \( P_1 \sqcap (P_2 \circ P_3) \).

From this point on, we assume (for the sake of simplicity) that in a language \( \mathcal{L} \), all the constructors simulated by \( \mathcal{L} \) are available in \( \mathcal{L} \). This implies that different strings may identify the same language. For example, \( \mathcal{ALEU} \) is the same language as \( \mathcal{ALE} \) (and \( \mathcal{ALEUC} \)), \( \mathcal{ALEO} \) is the same as \( \mathcal{ALEOB} \), and so on.

**Example 2.1.1** Consider the following two \( \mathcal{ALE} \)-concepts

\[
\text{Person}\sqcap \exists \text{CHILD}.\text{Graduate}, \quad \text{Person}\sqcap \exists \text{CHILD}\sqcap \forall \text{CHILD}.\text{Graduate}
\]

The first one denotes the persons having at least one graduate child. The second one denotes the individuals having at least one child and having only graduate children. It is easy to see that they are both satisfiable and that the first one subsumes the second. Conversely, the following concept

\[
\exists \text{CHILD}\sqcap (\forall \text{CHILD}.\text{Female}) \sqcap (\forall \text{CHILD}.\neg \text{Female})
\]

is not satisfiable and it is therefore trivially subsumed by both the others. □

The constructors \( \mathcal{O} \) and \( \mathcal{B} \) have the peculiarity of involving the individual names. Such peculiarity gives the languages including one of them some special properties. We discuss such properties in Chapter 5, which is specifically devoted to the constructors \( \mathcal{O} \) and \( \mathcal{B} \). For reasons that will be clear in Chapter 5, we call *pure languages* those languages including neither \( \mathcal{O} \) nor \( \mathcal{B} \), and *mixed languages* those including at least one of them.
2.2 Concept-Based Knowledge Bases

A knowledge base built by means of concept languages is formed by two components: The \textit{intensional} one, called TBox, and the \textit{extensional} one, called ABox.

2.2.1 Intensional Knowledge

We first focus our attention on the intensional component of a knowledge base, i.e., the TBox. Given a language $\mathcal{L}$, a TBox-statement in $\mathcal{L}$ has one of the forms:

\[ A \leq C \quad \text{Primitive Concept Specification}, \]
\[ A \equiv C \quad \text{Concept Definition} \]

where $A$ is a concept name and $C$ is an $\mathcal{L}$-concept. We call introduction of $A$, a TBox-statement in which the concept name $A$ appears in the left-hand side. An \textit{acyclic} TBox is a finite set of TBox-statements respecting the following conditions:

I. Each concept name has at most one introduction;

II. the TBox contains no \textit{cycles}.

Formally, a cycle is defined as follows [Neb91]: A concept name $A$ \textit{directly uses} another concept name $B$ if and only if $B$ appears in the introduction of $A$. A concept name $A_0$ uses $A_n$ if there is a chain $A_0, A_1, \ldots, A_n$ such that $A_i$ directly uses $A_{i+1}, 0 \leq i \leq n - 1$. Finally, we say that a TBox contains a cycle if some concept name uses itself.

A concept name appearing on the left-hand side of a concept definition is called a \textit{defined concept}, a concept name appearing on the left-hand side of a concept specification is called a \textit{primitive concept}, and a concept name not appearing on the left-hand side of any TBox-statement is called an \textit{atomic concept}.

Actual systems generally provide the user also with a mechanism for definitions and specifications of roles. Although they have practical significance, we neglect them here, since they are not necessary for the work carried in this thesis and they do not substantially change the results of the thesis.

An interpretation $\mathcal{I}$ \textit{satisfies} the statement $A \leq C$ if $A^\mathcal{I} \subseteq C^\mathcal{I}$ and the statement $A \equiv C$ if $A^\mathcal{I} = C^\mathcal{I}$. An interpretation $\mathcal{I}$ is a \textit{model} for a TBox $T$ if $\mathcal{I}$ satisfies all the statements in $T$.

Acyclic TBoxes are the ones employed in most existing concept-based KR systems. However, there exist other possible mechanisms for expressing the
intensional knowledge, which have been considered in the literature on concept
languages.

General TBoxes are obtained removing Condition (II) (i.e. the acyclicity
condition) from the definition of TBox. Thus, general TBoxes are composed
also of cyclic definitions, which have the same syntactic form of a definition
with the difference that the concept \( C \) in the right-hand side can be an arbi-
trary concept.

Cyclic definitions enhance substantially the expressive power of a system,
giving the opportunity to define concepts that are naturally expressed in a
recursive way. For example the concept Human can be defined in the following
way

\[
\text{Human} \equiv \text{Mammal} \cap (\leq 2 \text{PARENT}) \cap (\geq 2 \text{PARENT}) \cap \forall \text{PARENT.Human}.
\]

The researchers working with general TBoxes have considered at least three
different semantics for cyclic definitions. One of them is obtained by the same
semantic specification given above for acyclic TBoxes, which in the case of
cyclic definitions is called \textit{descriptive semantics}. Beside that, they proposed
other two, namely the least fixed point and the greatest fixed point semantics.

Fixed point semantics restrict the set of models of a TBox to the least or
greatest fixed point of the equation corresponding to a concept introduction.
Conversely, the descriptive semantics accepts any fixed point as a model of the
statement. However, the discussion about the semantics of cyclic definitions is
out of the scope of this thesis (see [Neb91] for a survey of the topic). We assume
to use the descriptive semantics, which seems to be the most appropriate one,
although it is not completely satisfactory in all practical cases [Baa90b, Neb91,
DL94b]. It is worth to remark that when the definitions are acyclic, the three
semantics are equivalent and therefore the question does not arise.

An even more general kind of TBox, called \textit{free TBox}, is obtained admitting
general concepts in the left-hand side of a TBox-statement. More precisely,
free TBox-statements are of the form

\[
C \preceq D \quad \text{Concept Inclusion}, \\
C \equiv D \quad \text{Concept Equation}
\]

where both \( C \) and \( D \) are arbitrary concepts. In free TBoxes, no restriction is
imposed on the form of the statement, therefore they include cyclic definitions
as a particular case. When using free TBoxes, the distinction between defined,
primitive and atomic concepts does not make sense anymore.

The semantics used for inclusions and equations is still the descriptive
semantics: An interpretation \( I \) satisfies the TBox-statement \( C \preceq D \) if \( C^I \subseteq
D^I \) and the TBox-statement \( C \equiv D \) if \( C^I = D^I \). Notice that fixed point
semantics naturally apply only to fixed point statements like \( A \equiv D \) (where
$D$ is a “function” of $A$, i.e. $A$ appears in $D$), and there is no obvious way to extend them to inclusions and equations.

In this thesis we consider acyclic TBoxes, general TBoxes, and free TBoxes. However, it is worth noticing that even other kinds of TBox can be considered. For example, a further one is obtained allowing only primitive concept specifications, i.e. statements of the form $A \subseteq C$, without Conditions (I) and (II). This mechanism, that we call primitive-TBox, offers nice computational properties (see e.g. [BJNS94, BDNS94]). For this reason, a mechanism of this kind is employed in other field such as Object-Oriented Databases and Semantic Data Models. However, we do not consider it in this thesis.

2.2.2 Extensional Knowledge

The construction of the extensional component of a knowledge bases, the ABox, is realized by permitting concept and role to be used in assertions on individuals. Given a concept language $\mathcal{L}$, an ABox-statement in $\mathcal{L}$ has one of the forms:

$$
C(a) \quad \text{Concept Membership Assertion}
$$

$$
R(a, b) \quad \text{Role Membership Assertion}
$$

where $C$ is an $\mathcal{L}$-concept, $R$ is an $\mathcal{L}$-role, and $a, b$ are individuals in $IN$.

The semantics of the above assertions is as follows: If $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is an interpretation, $C(a)$ is satisfied by $\mathcal{I}$ if $a^\mathcal{I} \in C^\mathcal{I}$, and $R(a, b)$ is satisfied by $\mathcal{I}$ if $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$.

A set $\mathcal{A}$ of assertions is called an ABox. An interpretation $\mathcal{I}$ is said to be a model of $\mathcal{A}$ if every assertion of $\mathcal{A}$ is satisfied by $\mathcal{I}$. A is said to be satisfiable if it admits a model.

As for the TBox, the ABox formalism introduced above is not the only one considered in the literature. For example, some systems (e.g. KRYPTON [BPL85]), allow the use of logical connectives in the assertions, i.e. they allow assertions of the form $R(a, b) \lor C(b)$. We call complex ABox-statement an assertion obtained combining atomic assertions with the connectives and, or, and not. Complex ABox-statements are provided with the above semantics for atomic assertions and the standard one for connectives. A complex ABox is then defined as a set of complex ABox-statements.

Complex ABoxes are not considered in this thesis, they are just briefly discussed in Section 5.1.5, where we discuss some cases in which logical connectives (and, or, and not) can be simulated with the use of the corresponding concept constructors (conjunction, disjunction, and negation)


2.2.3 Knowledge Bases

Given a language $\mathcal{L}$, a Concept-Based Knowledge Base (or simply knowledge base) $\Sigma$ in $\mathcal{L}$ is a pair $\Sigma = \langle T, A \rangle$, where $T$ is a TBox in $\mathcal{L}$ and $A$ is an ABox in $\mathcal{L}$. An interpretation $I$ is a model for $\Sigma$ if it is both a model for $T$ and a model for $A$. A knowledge base $\Sigma$ logically implies $\alpha$, where $\alpha$ is either a TBox- or an ABox-statement, written $\Sigma \models \alpha$, if $\alpha$ is true in every model of $\Sigma$.

We have considered different mechanisms for defining a TBox, therefore, we have different types of knowledge base. Therefore, throughout this thesis, we specify in each case (when needed) which TBox mechanism is in use.

2.3 Reasoning Services

There are several reasoning services to be provided by a concept-based knowledge base. As we already said in Chapter 1, we concentrate on the following basic ones, that we can now formally define.

Definition 2.3.1 Given a knowledge base $\Sigma = \langle T, A \rangle$, two concepts $C$ and $D$, and an individual $a$, we call:

- Concept Satisfiability, written as $\Sigma \not\models C \equiv 1$, the problem of checking whether $C$ is satisfiable w.r.t. $\Sigma$, i.e. whether there exists a model $I$ of $\Sigma$ such that $C^I \neq \emptyset$;

- Subsumption, written as $\Sigma \models C \subseteq D$, the problem of checking whether $C$ is subsumed by $D$ w.r.t. $\Sigma$, i.e. whether $C^I \subseteq D^I$ in every model $I$ of $\Sigma$;

- Consistency, written as $\Sigma \not\models$, the problem of checking whether $\Sigma$ is satisfiable, i.e. whether it has a model;

- Instance Checking, written as $\Sigma \models C(a)$, the problem of checking whether the assertion $C(a)$ is satisfied in every model of $\Sigma$;

The importance of Concept Satisfiability and Subsumption has been pointed out by several authors (see for example [Neb90a]). Consistency is used for verifying whether the information contained in a knowledge base is coherent. Finally, Instance Checking is used to check whether the knowledge base entails that an individual is an instance of a concept; it can be considered the central reasoning task for retrieving information on individuals in the knowledge-base. In fact, as we already said in Chapter 1, Instance Checking is a basic tool for more complex reasoning problems. For example, the Retrieval problem can be formulated in the following way: "given a knowledge base $\Sigma$ and a concept
2.3 Reasoning Services

$D$, find the set $\{a \in IN \mid \Sigma \models D(a)\}$, and it can be performed simply by iterating Instance Checking for all the individuals in $\Sigma$.

The Relation Checking problem $\Sigma \models R(a, b)$, i.e. the problem of checking whether an assertion of the form $R(a, b)$ is satisfied in every model of a knowledge base $\Sigma$, is also important in practical cases. We do not consider it in this thesis because it requires essentially the same technique as Instance Checking.

**Example 2.3.2** Let $\Sigma = \langle A, T \rangle$ be the following knowledge base in $\mathcal{ALC}$:

$T = \{\text{HappyParent} \leq \forall \text{CHILD.Graduate} \}$

$A = \{\text{CHILD(john,mary)},
\text{HappyParent(john)},
\text{Female} \lor \neg \text{FRIEND.Graduate(mary)}\}$

The knowledge base $\Sigma$ states that happy parents have only graduate children, Mary is a child of John, John is an happy parent, and Mary is a female and she has a graduate friend. It is easy to verify that $\Sigma$ is consistent. Notice also that the addition of the assertion $\neg \text{Graduate(mary)}$ to $\Sigma$ would make the knowledge base inconsistent. In fact, due to the first two assertions, $\Sigma$ logically implies $\text{Graduate(mary)}$. $
$

2.3.1 Special Cases

Researchers on concept languages have concentrated on some special case of the reasoning services presented above. In particular, they have considered the cases in which either one component or the whole knowledge base is empty.

In general, for all the reasoning services identified above, we have three different special cases, based on which component is empty and which is not. As a notation, the various different cases of the same service are identified using a prefix Pure, TBox, and ABox, respectively in the cases of both components empty (i.e. empty knowledge base), empty ABox, and empty TBox.

For example, Instance Checking w.r.t. an empty TBox and a (possibly) nonempty ABox is called ABox Instance Checking, and Subsumption w.r.t. an empty knowledge base is called Pure Subsumption (that is subsumption between two concepts considering all possible interpretations).

Moreover, abstracting from the specific service considered, we speak about pure-, TBox-, and ABox-reasoning, in the three above cases and about KB-reasoning in the general case.

For example, most of the research on Concept Satisfiability and Subsumption concentrates on the special case of pure-reasoning. In order to simplify the notation, in this case, we write those problems as $C \not\equiv \bot$ and $C \subseteq D$ instead of $\langle \emptyset, \emptyset \rangle \not\models C \equiv \bot$ and $\langle \emptyset, \emptyset \rangle \models C \subseteq D$. 

According to our classification, we have a two-dimensional grid of reasoning tasks. In fact, on one side we have the four reasoning services and on the other the four forms of reasoning. We have therefore 16 different reasoning tasks. In addition, we have different cases based on which kind of TBox we consider. However, not all of them are really meaningful and some of them are reducible to others. We see in Chapter 3 the relation between them and which of them are the most important.

2.4 Computational Complexity

We use standard notions from complexity theory as presented in [GJ79]. In particular, we will speak about the following complexity classes \( P, NP, \Pi_2^p, \PSPACE, \EXPTIME \). We briefly recall the definitions of these classes here.

The classes \( P, \PSPACE, \EXPTIME \) are defined in terms of the resources needed by a deterministic Turing Machine (TM) to solve a specific decision problem: The class \( P \) (resp. \( \PSPACE, \EXPTIME \)) contains the problems that can be solved by a deterministic TM in polynomial time (resp. polynomial space, exponential time). \( NP \) contains the problems that can be solved by a nondeterministic TM in polynomial time.

Given a class \( C \), the class \( \co C \) is the set of problems that are the complement of a problem in \( C \).

Given a class \( C \), a problem \( P_1 \) is said to be \( C \)-hard (w.r.t. the polynomial reduction) if for every problem \( P_2 \) in \( C \), there is a polynomial reduction from \( P_2 \) to \( P_1 \). If a \( C \)-hard problem \( P_1 \) is in \( C \) then \( P_1 \) is said to be \( C \)-complete.

Given two classes \( C_1 \) and \( C_2 \), we will call \( C_1 \subseteq C_2 \) the class of problems solved by a Turing Machine for \( C_1 \) that uses an oracle that solves (in constant time) a problem in \( C_2 \). The class \( \Pi_2^p \) is defined as \( \co(NP) \).

2.4.1 Complexity Measures for Concept Languages

The complexity of a problem is generally measured with respect to the size of its whole input. Thus, the complexity of Pure Subsumption, for instance, is measured w.r.t. the sum of the size of the candidate subsumer and that of the candidate subsume.

However, in our reasoning services various different pieces of input are given. For example, for Instance Checking, three pieces of input are given, namely the TBox, the ABox and the concept representing the query. For this reason, different kind of complexity measures can be considered for Instance Checking, similarly to what has been suggested in [Var82] for querying relational databases.

The complexity analysis performed in this thesis regards the case of empty TBox. We therefore consider Instance Checking problems of the following kind
$\langle \emptyset, \mathcal{A} \rangle \models C(a)$. In this specific case we account for the following measures:

1. **Knowledge Base Complexity**, the complexity with respect to $|\mathcal{A}|$

2. **Query complexity**, the complexity with respect to $|\mathcal{C}|$

3. **Combined Complexity**, the complexity with respect to both $|\mathcal{C}|$ and $|\mathcal{A}|$

where $|\mathcal{A}|$ and $|\mathcal{C}|$ denote the size of $\mathcal{A}$ and $\mathcal{C}$ respectively.

The combined complexity, taking into account the whole input, is the default one. Therefore, throughout this thesis, if not differently stated, we speak about combined complexity. The other two, instead, consider only a part of the input, neglecting the other one. For this reason, they are meaningful only in those cases in which it is reasonable to suppose the size that part negligible with respect to the size of the other. Since the case in which the size of the knowledge base is negligible w.r.t. the size of the query seems not reasonable in the actual applications, we consider only combined complexity and knowledge base complexity.

It is worth noticing that for every language the combined complexity, taking into account the whole input, is at least as high as the other two. In fact, the complexity with respect to $|\mathcal{A}|$ and $|\mathcal{D}|$ obviously includes as particular cases the possibilities that $|\mathcal{D}|$ or $|\mathcal{A}|$ is small with respect to the other.
Chapter 3

Reasoning Techniques and Previous Results

This chapter describes the state of the art of the research on the complexity of reasoning with concept languages. In particular, we discuss reasoning techniques and complexity results for the various languages we consider in this thesis.

An important remark is that we focus only on sound and complete reasoning techniques. That is, we do not consider the numerous incomplete algorithms proposed in the literature and employed in the implementation of the actual systems, such as BACK and LOOM.

In Section 3.1, we provide a tableaux-like calculus for reasoning in concept-based systems. Such calculus is the basic tool employed in the thesis to design reasoning algorithms and to prove complexity results. In Section 3.2, we present the picture of the complexity results for the various problems in the languages of interest for the thesis.

3.1 Tableaux Calculus

The calculus based on constraint systems was first introduced by Schmidtschauß and Smolka in 1988 (see [SS91]) to compute Pure Concept Satisfiability in $\mathcal{ALC}$ and its sublanguages. Later, Donini and others, in the period 1990-92, refined the calculus and extended it to compute Pure Concept Satisfiability and Pure Subsumption in a large class of languages (see [DHL+92, DLNN91a, DLNN91b]). In the mean time, Hollunder and Baader [Hol90, BH91b], proposed a way to extend the calculus to solve KB-reasoning problems, too.

The calculus has been proposed in various forms. The one we present here is an extension of the one that will appear in [DLNN94]. In the cited
paper, the calculus is conceived to deal with Pure Concept Satisfiability for the language $\mathcal{ALC}N^{\prime}RHIL$ and its sublanguages. We extend it here to include also the constructors $O$ and $B$ and to deal with KB-reasoning problems. In order to have the maximal generality, we consider free TBox-statements. However, for the sake of simplicity, we consider only statement of the form $C \leq D$, taking a statement of the form $C \equiv D$ as a shorthand for $C \leq D$ and $D \leq C$ (as done in [BDS93a]). Concept definitions (both cyclic and acyclic), being a special case of free TBox statements, are considered as well.

We assume that roles and concepts are given in a normal form: We say that a role is in inverse normal form if the constructor for the inverse role is applied only to role names. The following rewriting rules can be used to transform any role into an equivalent role in inverse normal form:

\[
\begin{align*}
(R_1 \cap R_2)^{-1} & \rightarrow R_1^{-1} \cap R_2^{-1} \\
(R_1 \circ R_2)^{-1} & \rightarrow R_2^{-1} \circ R_1^{-1} \\
(R^{-1})^{-1} & \rightarrow R.
\end{align*}
\]

We say a concept is in negation normal form if it contains only negations either of the form $\neg A$ or $\neg\{a_1, \ldots, a_n\}$, where $A$ is a concept name. Arbitrary concepts can be rewritten to equivalent concepts which are in negation normal form by the following rules:

\[
\begin{align*}

\neg \top & \rightarrow \bot \\
\neg \bot & \rightarrow \top \\
\neg \neg C & \rightarrow C \\
\neg (\leq n R) & \rightarrow (\geq (n + 1) R) \\
\neg (\geq n R) & \rightarrow \begin{cases} 
\bot & \text{if } n = 0 \\
\leq (n - 1) R & \text{if } n > 0
\end{cases} \\
\neg (C \cap D) & \rightarrow \neg C \cup \neg D \\
\neg (C \cup D) & \rightarrow \neg C \cap \neg D \\
\neg (\forall R. C) & \rightarrow \exists R. \neg C \\
\neg (\exists R. C) & \rightarrow \forall R. \neg C \\
\neg R : a & \rightarrow \forall R. \neg \{a\}.
\end{align*}
\]

From this point on, we assume that concepts are always in negation normal form and role in inverse normal form.

3.1.1 Constraint Systems and Completion Rules
We assume that there exists an alphabet $\mathcal{V}$ of variable symbols, which are denoted by the letters $x, y, z,$ and $w$, possibly with subscript. The calculus
operates on constraints consisting of individuals, variables, concepts, and roles. We use the term *object* for the elements of \( \mathcal{O} \cup \mathcal{V} \) (i.e. as an abstraction for variables and individuals), and use \( s, t \) to denote objects. A *constraint* has one of the forms

\[
s : C \quad s R t \quad s \neq t \quad \forall x \cdot x : C
\]

where \( C \) is a concept and \( R \) is a role, \( s, t \) are objects. Intuitively, \( s : C \) represents the constraint that \( s \) is in the interpretation of \( C \), \( s R t \) the constraint that the pair \((s, t)\) is in the interpretation of \( R \), \( s \neq t \) the constraint that \( s \) and \( t \) are interpreted differently, and \( \forall x \cdot x : C \) the constraint that every object is in the interpretation of \( C \).

Let \( \mathcal{I} \) be an interpretation. An *\( \mathcal{I} \)-assignment* is a function \( \alpha \) that maps every variable to an element of \( \Delta^\mathcal{I} \) and every individual \( a \) to \( a^\mathcal{I} \) (i.e. \( \alpha(a) = a^\mathcal{I} \) for all \( a \in IN \)). We say that the pair \( \mathcal{I}, \alpha \) satisfies the constraint \( s : C \) if \( \alpha(s) \in C^\mathcal{I} \), \( \mathcal{I}, \alpha \) satisfies \( s R t \) if \((\alpha(s), \alpha(t)) \in R^\mathcal{I} \), \( \mathcal{I}, \alpha \) satisfies \( s \neq t \) if \( \alpha(s) \neq \alpha(t) \), \( \mathcal{I}, \alpha \) satisfies \( \forall x \cdot x : C \) if \( C^\mathcal{I} = \Delta^\mathcal{I} \) (notice that \( \alpha \) does not play any role in this case).

A constraint is *satisfiable* if there is an interpretation \( \mathcal{I} \) and an \( \mathcal{I} \)-assignment \( \alpha \) such that \( \mathcal{I}, \alpha \) satisfies it. A *constraint system* \( S \) is a finite, nonempty set of constraints. A pair \( \mathcal{I}, \alpha \) satisfies \( S \) if \( \mathcal{I}, \alpha \) satisfies every constraint in \( S \). We say that \( S \) is *satisfiable* if there is an interpretation \( \mathcal{I} \) and an \( \mathcal{I} \)-assignment \( \alpha \) such that \( \mathcal{I}, \alpha \) satisfies \( S \).

A knowledge base \( \Sigma = \langle \mathcal{I}, A \rangle \) can be translated into a constraint system \( S_\Sigma \) by replacing every inclusion \( C \equiv D \in \mathcal{T} \) with the constraint \( \forall x \cdot x : \neg C \cup D \), every membership assertion \( C(a) \) with the constraint \( a : C \), and every assertion \( R(a, b) \) with \( a R b \), and including the constraint \( a \neq b \) for every pair \((a, b)\) of individuals appearing in \( \Sigma \).

The following proposition is an immediate consequence of the above definitions.

**Proposition 3.1.1** Given a knowledge base \( \Sigma \), two concepts \( C, D \), and an individual \( a \):

1. \( \Sigma \not\models C \equiv \bot \) if and only if the constraint system \( S_\Sigma \cup \{x : C\} \) is satisfiable.
2. \( \Sigma \models C \subseteq D \) if and only if the constraint system \( S_\Sigma \cup \{x : C \cap \neg D\} \) is unsatisfiable.
3. \( \Sigma \not\models \) if and only if \( S_\Sigma \) is satisfiable.
4. \( \Sigma \models C(a) \) if and only if \( S_\Sigma \cup \{a : \neg C\} \) is unsatisfiable.

Notice that if the knowledge base \( \Sigma \) is empty, the corresponding constraint system \( S_\Sigma \) is empty as well. Therefore checking if \( C \equiv \bot \) and \( C \subseteq D \) can be
done checking the constraint systems \( \{ x : C \} \) and \( \{ x : C \cap \neg D \} \) for satisfiability and for unsatisfiability, respectively.

Before presenting the completion rules, we need some definitions. Let \( S \) be a constraint system, \( x \) a variable, \( s \) an object both occurring in \( S \), and \( R \) a role. With \( S[x/s] \) we denote the constraint system obtained from \( S \) by replacing each occurrence of \( x \) by \( s \). We say that \( t \) is an \( R \)-successor of \( s \) in \( S \) if the constraint \( sRt \) is in \( S \), and we say that \( t \) is a direct successor of \( s \) in \( S \) if \( t \) is an \( R \)-successor of \( s \) in \( S \) for some \( R \). Moreover, we denote by \( \text{successor} \) the transitive closure of the relation direct successor, and we denote by (direct) predecessor the inverse of the relation (direct) successor.

We say that \( s, t \) are separated in \( S \) if the constraint \( s \neq t \) is in \( S \). We say that \( t \) is a filler of \( R \) for \( s \) in \( S \) if either \( t \) is an \( R \)-successor of \( s \) in \( S \), or one of the following conditions is satisfied:

- \( R \) is a role name \( P \) and \( tP^{-1}s \in S \);
- \( R \) is \( P^{-1} \) and \( tPs \in S \);
- \( R \) is \( R_1 \cap R_2 \), and \( t \) is a filler of \( R_1 \) and \( R_2 \) for \( s \) in \( S \);
- \( R \) is \( R_1 \circ R_2 \) and there is a \( t_1 \) such that \( t_1 \) is a filler of \( R_1 \) for \( s \) in \( S \) and \( t \) is a filler of \( R_2 \) for \( t_1 \) in \( S \).

We say that \( t \) is a filler for \( s \) in \( S \) if there is an \( R \) such that \( t \) is a filler of \( R \) for \( s \) in \( S \), and we say that \( t \) is a filler in \( S \) if there is an \( s \) such that \( t \) is a filler for \( s \) in \( S \). We do not mention the constraint system \( S \) if it is clear from the context.

The completion rules are the following:

- \( S \to_1 \{ s : C_1, s : C_2 \} \cup S \)
  if \( s : C_1 \cap C_2 \) is in \( S \), \( s : C_1 \) and \( s : C_2 \) are not both in \( S \);
- \( S \to_\omega \{ s : D \} \cup S \)
  if \( s : C_1 \cup C_2 \) is in \( S \), neither \( s : C_1 \) nor \( s : C_2 \) is in \( S \), and \( D = C_1 \) or \( D = C_2 \);
- \( S \to_3 \{ sRy, y : C \} \cup S \)
  if \( s : \exists R.C \) is in \( S \), there is no \( t \) such that \( t \) is an \( R \)-successor of \( s \) in \( S \) and \( t : C \) is in \( S \), and \( y \) is a new variable;
- \( S \to_\forall \{ t : C \} \cup S \)
  if \( s : \forall R.C \) is in \( S \), \( t \) is a filler of \( R \) for \( s \), and \( t : C \) is not in \( S \);
3.1 Tableaux Calculus

- \( S \rightarrow \{ sR_{y_1}, \ldots, sR_{y_n} \} \cup \{ y_i \neq y_j \mid i, j \in 1..n, i \neq j \} \cup S \)
  
  if \( s: (\geq n R) \) is in \( S \), there do not exist \( n \) pairwise separated fillers of \( R \) for \( s \) in \( S \), and \( y_1, \ldots, y_n \) are new variables;

- \( S \rightarrow \leq S[t/z] \)
  
  if \( s: (\leq n R) \) is in \( S \), \( s \) has more than \( n \) fillers of \( R \) for \( s \), and \( t, z \) are two fillers of \( R \) for \( s \) that are not separated;

- \( S \rightarrow \neg \{ tPs \} \cup S \)
  
  if \( sP^{-1}t \) is in \( S \) and \( tPs \) is not in \( S \);

- \( S \rightarrow \exists \{ sR_1t, sR_2t \} \cup S \)
  
  if \( s(R_1 \cap R_2)t \) is in \( S \), and either \( sR_1t \) or \( sR_2t \) is not in \( S \);

- \( S \rightarrow \circ \{ sR_1z, zR_2t \} \cup S \)
  
  if \( s(R_1 \circ R_2)t \) is in \( S \), \( z \) is a new variable and there is no \( t_1 \) such that both \( sR_1t_1 \) and \( t_1R_2t \) are in \( S \);

- \( S \rightarrow \sigma [x/a_i] \)
  
  if \( x: \{ a_1, \ldots, a_n \} \) is in \( S \) and \( i \in \{ 1, \ldots, n \} \);

- \( S \rightarrow B \{ sRa \} \cup S \)
  
  if \( s: (R : a) \) is in \( S \) and \( sRa \) is not in \( S \);

- \( S \rightarrow \forall x \{ s: C \} \cup S \)
  
  if \( \forall x \cdot x: C \) is in \( S \), \( s \) appears in \( S \), and \( s: C \) is not in \( S \).

We distinguish between two kinds of rules: deterministic ones (\( \rightarrow_{\neg} \), \( \rightarrow_{\exists} \), \( \rightarrow_{\circ} \)), \( \rightarrow_{\forall} \), \( \rightarrow_{B} \), \( \rightarrow_{\sigma} \)) and nondeterministic ones (\( \rightarrow_{1} \), \( \rightarrow_{\leq} \), \( \rightarrow_{\sigma} \)). A deterministic rule can be applied only in one way to a specific constraint whereas a nondeterministic rule can be applied in more than one way.

A further distinction is between generating rules (\( \rightarrow_{\geq} \) and \( \rightarrow_{\exists} \) and non-generating ones (all the others). The former being those which introduce new variables in the constraint system.

**Proposition 3.1.2 (Invariance)** Let \( S \) and \( S' \) be constraint systems. Then:

1. If \( S' \) is obtained from \( S \) by the application of a deterministic rule, then \( S \) is satisfiable if and only if \( S' \) is satisfiable.

2. If \( S' \) is obtained from \( S \) by the application of a nondeterministic rule, then \( S \) is satisfiable if \( S' \) is satisfiable. Furthermore, if a nondeterministic rule applies to \( S \), then it can be applied in such a way that it yields a constraint system \( S' \) such that \( S' \) is satisfiable if and only if \( S \) is satisfiable.
One can verify that rules are always applied to a constraint system \( S \) either because of the presence in \( S \) of a given constraint \( s : C \), or, in the case of the \( \rightarrow \varphi \)-rule, because of the presence of an object \( s \) in \( S \). When no confusion arises, we say that a rule is applied to the object \( s \) or to the constraint \( s : C \) (instead of saying that it is applied to the constraint system \( S \)).

We say that a constraint system \( S \) contains a clash if one of the following conditions holds:

1. there exists an object \( s \) such that \( s : \bot \in S \);
2. there exists an object \( s \) and a concept name \( A \) such that \( s : A \in S \) and \( s : \neg A \in S \);
3. there exists an object \( s \) and \( n + 1 \) distinct role fillers \( t_1, \ldots, t_{n+1} \) of the role \( R \) for \( s \) in \( S \) such that \( S \) contains the constraints \( s : (\leq n R) \) and \( t_i \neq t_j \), where \( 1 \leq i, j \leq n + 1 \) and \( i \neq j \);
4. there exists an object \( s \) such that \( s : (\geq n R) \) and \( s : (\leq m R) \) are in \( S \), with \( n > m \).
5. there exists an individual \( a \) such that \( a : \{a_1, \ldots, a_n\} \) is in \( S \) with \( a \neq a_i \) for all \( i = 1, \ldots, n \);
6. there exists an individual \( a \) such that \( a : \neg\{a_1, \ldots, a_n\} \) is in \( S \), with \( a = a_i \) for some \( i = 1, \ldots, n \).

If \( S, S' \) are two constraint systems, then \( S' \) is said to be directly derived from \( S \) if it is obtained from \( S \) by the application of one completion rule, and \( S' \) is said to be derived from \( S \) if it is obtained from \( S \) by a sequence of applications of the completion rules. A constraint system is complete if no completion rule applies to it. Any complete constraint system that is derived from \( S \) is called a completion of \( S \).

It is possible to show that if a constraint system contains a clash, then it is not satisfiable, and that a complete constraint system is satisfiable if it contains no clash.

In [BH91b, DLNN91a], several computational properties of the calculus are discussed. Here, we report, without proof, some basic properties that are used in the subsequent sections.

**Proposition 3.1.3** Let \( S \) be a constraint system. The following problems can be solved in polynomial time with respect to \(|S|\):

1. Deciding which rules are applicable to \( S \);
2. Computing the constraint system resulting from applying a rule to \( S \);
3. Checking whether \( S \) is clash-free.
3.1.2 Specialization of the Calculus

The calculus proposed in the previous section works for the most expressive language $\mathcal{ALCN^RHIOTB}$ with the most expressive TBox mechanism. Therefore, it also works for any sublanguage of it, with any form of TBox. However, when we deal with smaller languages, the calculus can be specialized. The reason to specialize it, is to put it in a form more suitable to make some optimizations.

The specialized calculus can be turned into an effective decision procedure in order to prove certain property of the language (e.g. decidability, complexity). In the general case, instead, we cannot turn the calculus into a procedure since all reasoning services in the general language have been proved undecidable in [Len93].

Notice that each constructor is associated with a completion rule, in the sense that the rule is applied only to constraints containing such constructor. Therefore, an obvious specialization for any languages $\mathcal{L}$ is obtained eliminating all the rules associated with constructors not included in $\mathcal{L}$.

A second possible specialization regards the TBox: If the TBox is empty, the $\rightarrow_{\psi}$-rule can be eliminated. In the case of different TBox mechanism it is possible to refine the $\rightarrow_{\psi}$-rule in order to make its application more efficient. An example of it is given in [BJNS94] for primitive TBox, i.e. for a TBox composed only by primitive concept specification.

Another possible optimization is related to the working space required by the procedure. In fact, a procedure derived directly from the calculus, would require to keep in memory a whole completion of a constraint system $S$. In the general case, the completions may be very large w.r.t. the size of $\Sigma$. Conversely, when the calculus is used to prove some upper bounds for reasoning services in specific languages, it is crucial to keep in memory only the least portion of information needed, and reason only on it. Efficient algorithms therefore should not keep the entire complete constraint system in the memory but store only small portions of it at a time.

With the aim of saving working space, we now introduce a different set of completion rules, called trace rules, that are meant to build up only a portion of complete constraint systems.

The trace rules consist of the nongenerating rules together with the following two generating rules that replace the $\rightarrow_{\exists}$ and the $\rightarrow_{\geq}$-rule and are obtained from them by adding a further condition:

- $S \rightarrow_{T\exists} \{sRy, y: C\} \cup S$
  
  if $s : \exists R.C$ is in $S$, there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t : C$ is in $S$, $y$ is a new variable, and for all constraint $tRx$ in $S$, $t$ is a predecessor of $s$ or $s = t$
$S \rightarrow_{T \geq} \{sRy_1, \ldots, sRy_n\} \cup \{y_i \neq y_j \mid i, j \in 1..n, i \neq j\} \cup S$

if $s : (\geq n R)$ is in $S$ there do not exist $n$ pairwise separated fillers of $R$ for $s$ in $S$, $y_1, \ldots, y_n$ are new variables, and for all constraint $tRx$ in $S$, $t$ is a predecessor of $s$ or $s = t$

Let $T$ be a constraint system obtained from $S$ by application of the trace rules. We call $T$ a trace of $S$ if no trace rule applies to $T$.

Trace rules exhibit the following behavior: Given an object $s$, if at least one generating rule is applicable, all its successors $y_1, \ldots, y_n$ are introduced. Then, after some nongenerating rules are applied, one variable $y_i$ is (nondeterministically) chosen, and all successors of $y_i$ are introduced. Unlike normal completion rules, no successor is introduced for any object different from $y_i$. Then, one variable is chosen among the successors of $y_i$, only its successors are added to the constraint system, and so on. An example of trace is shown in Figure 3.1.

The reason why we introduce all the successors of the “chosen” object is the following: For every chosen object $s$ all successors of $s$ must be present simultaneously at some stage of the computation, since only the interplay of role conjunctions and number restrictions forces us to identify certain succes-
ors. This is important because, when identifying variables, the constraints imposed on them are combined, which may lead to clashes that otherwise would not have occurred. In the languages that do not support number restrictions, the definition of trace can be simplified, creating only one successor for each variable.

Trace rules have been used in [DLNN91a] for deciding Pure Concept Satisfiability and Pure Subsumption of \( ALCNR \)-concepts. Algorithms based on trace rules have been given also in [SS91, HNS90] for various sublanguages of \( ALCNR \).

Later in the thesis, we make use of the trace rules to establish various results for different languages.

### 3.2 Previous Results

In this section we present the previous complexity results for reasoning with concept languages. All those results regard exclusively pure languages; therefore from this point on we suppose to deal only with pure languages, referring to Chapter 5 for the discussion about mixed languages.

We start presenting a set of reductions between the four reasoning tasks defined in Chapter 2. In the two following subsections, we describe results for Concept Satisfiability and Subsumption (Subsection 3.2.2), and Consistency and Instance Checking (Subsection 3.2.3). We also discuss such results, highlighting the sources of complexity and the reasoning patterns associated with them.

#### 3.2.1 Relations between the Reasoning Services

The four reasoning problems mentioned above are not independent of each other. In particular, consider a knowledge base \( \Sigma = \langle T, A \rangle \), two concepts \( C, D \), and an individual \( a \) not appearing in \( \Sigma \), the following relations hold:

\[
\begin{align*}
\langle T, A \rangle & \not\models C \equiv \bot \quad \iff \quad \langle T, A \cup \{ C(a) \} \rangle \not\models \tag{3.1} \\
\langle T, A \rangle & \not\models C \equiv \bot \quad \iff \quad \langle T, A \rangle \not\models C \subseteq \bot \tag{3.2} \\
\langle T, A \rangle & \models C \sqsubseteq D \quad \iff \quad \langle T, A \cup \{ C(a) \} \rangle \models D(a) \tag{3.3} \\
\langle T, A \rangle & \not\models \quad \iff \quad \langle T, A \rangle \not\models \bot(a) \tag{3.4}
\end{align*}
\]

It follows that Concept Satisfiability can be reduced to both Consistency (3.1) and the complement of Subsumption (3.2). Subsumption can be reduced to Instance Checking (3.3). Finally, Consistency can be reduced to the complement of Instance Checking (3.4).
For languages with the constructor $C$ for expressing the negation of concepts, the following other relations hold too:

\[
C \sqsubseteq D \iff (C \cap \neg D) \equiv \bot \tag{3.5}
\]

\[
\langle T, \mathcal{A} \rangle \models C(b) \iff \langle T, \mathcal{A} \cup \{ \neg C(b) \} \rangle \models \tag{3.6}
\]

Therefore, Subsumption can be reduced to the complement of Concept Satisfiability (3.5). Instance Checking can be reduced to the complement of Consistency (3.6).

### 3.2.2 Complexity of Concept Satisfiability and Subsumption

As we said in Chapter 2, the research on Concept Satisfiability and Subsumption has been concentrated on the pure-reasoning case. There are mainly two reasons to restrict the problems to those special cases. The first is a pragmatic reason, researchers started dealing with the smaller problem to move to the bigger one once the former is well understood. The second reason is a technical one: It has been proved that the general problems can be easily solved by means of the special cases as explained later in this section.

#### Pure Concept Satisfiability and Pure Subsumption

In [SS91, DLNN91a, DLNN91b, DHL+92] several complexity results for Pure Concept Satisfiability and Pure Subsumption have been presented. In particular, in [DLNN91a] a comprehensive analysis of the complexity of both problems for all the languages obtained combining the constructors $C, \mathcal{E}, \mathcal{N}, \mathcal{R}$ and $\mathcal{U}$ is accomplished. The results of that analysis that are interesting for this thesis are summarized in Table 3.1. Each entry of Table 3.1 reports the class to which the problem belongs and the citation to the paper in which the result is achieved. The diction NP, coNP, and PSPACE mean that the problem is complete for the given class, while P means that the problem is in the class P.

The results concerning $\mathcal{ALE}$, $\mathcal{ALR}$, and $\mathcal{ALU}$ show that there are two different sources of complexity that have an impact on the tractability of Pure Concept Satisfiability and Pure Subsumption. More precisely, augmenting $\mathcal{A}\mathcal{E}$ with qualified existential quantification ($\mathcal{ALE}$) or role conjunction ($\mathcal{ALR}$) makes Concept Satisfiability coNP-complete, while augmenting $\mathcal{A}\mathcal{E}$ with disjunction of concepts ($\mathcal{ALU}$) makes the problem NP-complete. In addition, in the language including both constructs ($\mathcal{ALC}$) concept satisfiability is PSPACE-complete.

The upper bounds for $\mathcal{ALE}$ and $\mathcal{ALC}$ are given in [SS91] and they are obtained exploiting the trace rules. The key observation for deriving the upper bound of the complement of Pure Concept Satisfiability in $\mathcal{ALE}$ is that we can
3.2 Previous Results

<table>
<thead>
<tr>
<th></th>
<th>$C \not\equiv \bot$</th>
<th>$C \equiv D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{AL}$</td>
<td>P [SS91]</td>
<td>P [DLNN91b]</td>
</tr>
<tr>
<td>$\mathcal{ALN}$</td>
<td>P [DLNN91b]</td>
<td>P [DLNN91b]</td>
</tr>
<tr>
<td>$\mathcal{ALE}$</td>
<td>coNP [DHL+92]</td>
<td>NP [DHL+92]</td>
</tr>
<tr>
<td>$\mathcal{ALR}$</td>
<td>coNP [DLNN91b]</td>
<td>NP [DLNN91b]</td>
</tr>
<tr>
<td>$\mathcal{ALU}$</td>
<td>NP [DLNN91b]</td>
<td>coNP [DLNN91b]</td>
</tr>
<tr>
<td>$\mathcal{ALC}$</td>
<td>PSSPACE [SS91]</td>
<td>PSSPACE [SS91]</td>
</tr>
</tbody>
</table>

Table 3.1: Complexity of Pure Concept Satisfiability and Pure Subsumption

encode the applications of the completion rules into a nondeterministic Turing machine $T$ that, using the $\rightarrow_{T\exists}$-rule, computes a trace of $\{s: C\}$ and checks whether such a trace contains a clash. It is easy to see that the number of variables in a trace of $\{s: C\}$ is bounded by the number of subconcepts of $C$ of the form $\exists R.D$, and therefore the size of a trace of $\{s: C\}$ is polynomially bounded by $|C|$. It follows that both computing a trace and checking whether it contains a clash can be done in polynomial time with respect to $|C|$. We can conclude that $T$ is a nondeterministic polynomial time Turing machine, which implies that the complement of Pure Concept Satisfiability in $\mathcal{ALC}$ is in NP, and therefore Pure Concept Satisfiability is in coNP. The same considerations hold for $\mathcal{ALR}$ as shown in [DLNN91a].

The PSSPACE algorithm for $\mathcal{ALC}$ is based on the same idea. In fact, keeping track of only a trace at a time, it is possible to solve Pure Concept Satisfiability (and Subsumption) in $\mathcal{ALC}$ with a nondeterministic algorithm that uses linear space.

Let us now discuss the complexity result about $\mathcal{ALU}$. We refer, in particular, to checking whether an $\mathcal{ALU}$-concept $C$ is satisfiable. Again, we can encode the applications of the completion rules for checking the satisfiability of $C$ into a Turing machine that nondeterministically computes a completion of $\{s: C\}$, and then check whether such a completion is clash-free. The nondeterminism is clearly due to the $\rightarrow_{U}$-rule. It is shown in [DLNN91a] that
both computing one completion of \( \{ s : C \} \) and checking whether it is clash-free can be done in polynomial time. It is also shown that the complement of Subsumption can be computed by a nondeterministic Turing machine in polynomial time. It follows that Pure Concept Satisfiability in \( \mathcal{ALH} \) is in NP, and Subsumption is in coNP.

**TBox Concept Satisfiability and TBox Subsumption**

We start considering the case of acyclic TBox. In [Neb90b], it is shown that TBox-reasoning problems w.r.t. an acyclic TBox can be reduced to pure-reasoning. In details, in the cited paper it is proved that for every TBox the following results hold:

\[
\langle T, A \rangle \not\models (C \equiv \bot) \text{ iff } \langle \emptyset, A \rangle \not\models (U(C) \equiv \bot) \tag{3.7}
\]

\[
\langle T, A \rangle \models (C \sqsubseteq D) \text{ iff } \langle \emptyset, A \rangle \models (U(C) \sqsubseteq U(D)) \tag{3.8}
\]

where \( U(\cdot) \) is a function that gets a concept of a language and returns a concept of the same language. We do not supply the details of the definition of \( U(\cdot) \), they can be found in [Neb90b]. Intuitively, it is obtained iteratively substituting in \( C \) every concept name with its definition. Notice that the termination of the procedure is ensured by the acyclicity of the TBox; in case of cyclic TBox the above result does not hold.

Relations 3.7, 3.8 (specialized to the case \( A = \emptyset \)) explain why the research on TBox-reasoning is replaced by research on pure-reasoning, without taking into account the TBox.

It is important to notice that, in [Neb01], it is proved that the transformation \( U(\cdot) \), which makes the above simplification possible, is exponential in size, in the general case. However, in the cited paper it is also shown that under reasonable restrictions it works in polynomial space. For this reason, we suppose that such transformation does not affect the complexity of the above reasoning tasks.

We now discuss previous work about TBox-reasoning with general TBoxes and free TBoxes.

In [Neb90b] the language \( FLN^- \) (called \( TF \) in the paper) is considered, with TBoxes containing (possibly cyclic) concept definitions, role definitions and disjointness axioms (stating that two concept names are disjoint). Nebel shows that Subsumption between \( FLN^- \)-concepts w.r.t. a TBox composed by the above axioms is decidable. However, the argument he uses is not constructive: He shows that it is sufficient to consider finite interpretations of a size bounded by the size of the TBox in order to decide Subsumption.

In [Baa90b] the effect of the three types of semantics—descriptive, greatest fixed-point and least-fixed-point semantics—for the language \( FL_0 \), containing only concept conjunction and universal quantification, is described with the
help of *finite automata*. Baader reduces Subsumption between $\mathcal{FL}_o$-concepts w.r.t. a TBox containing (possibly cyclic) definitions of the form $A \sqsubseteq C$ (which he calls terminological axioms) to decision problems for finite automata. In particular, he shows that Subsumption w.r.t. descriptive semantics can be decided in polynomial space using *Büchi automata*. Using results from [Baa90b], in [Neb91], a characterization of the above Subsumption problem w.r.t. descriptive semantics is given with the help of deterministic automata (whereas Büchi automata are nondeterministic). This also yields a PSPACE-algorithm for deciding Subsumption.

In [BBH+90] the attention is restricted to the language $\mathcal{ALC}$. In particular, that paper considers the problem of checking the satisfiability of a single equation of the form $C \equiv \top$, where $C$ is an $\mathcal{ALC}$-concept. This problem, called the *universal satisfiability problem*, is shown to be equivalent to checking the satisfiability of a TBox in $\mathcal{ALC}$ (see Proposition 7.3.12).

In [Baa90a], an extension of $\mathcal{ALC}$, called $\mathcal{ALC}_{reg}$, is introduced, which supports a constructor to express the transitive closure of roles. By means of transitive closure of roles it is possible to replace cyclic primitive concept specification (i.e. statements of the form $A \sqsubseteq D$) with equivalent acyclic ones. The problem of checking the satisfiability of an $\mathcal{ALC}_{reg}$-concept is solved in that paper. It is also shown that using transitive closure it is possible to reduce TBox Concept Satisfiability in $\mathcal{ALC}$ w.r.t. a free TBox into Pure Concept Satisfiability in $\mathcal{ALC}_{reg}$ (using the descriptive semantics). Since TBox Concept Satisfiability is trivially harder than checking the satisfiability of a TBox, that paper extends the result given in [BBH+90].

The technique exploited in [BBH+90] and [Baa90a] is based on the notion of *concept tree*. A concept tree is generated starting from a concept $C$ in order to check its satisfiability (or universal satisfiability). The way a concept tree is generated from a concept $C$ is similar in flavor to the way a complete constraint system is generated from the constraint system $\{x : C\}$. However, the extension of the concept tree method to deal with number restrictions and individuals in the knowledge base is neither obvious, nor suggested in the cited papers; on the other hand, the extension of the calculus based on constraint systems is immediate, provided that additional features have a counterpart in FOL.

In [Sch91] some results more general than those in [Baa90a] are obtained by considering languages more expressive than $\mathcal{ALC}_{reg}$ and dealing with the Pure Concept Satisfiability in such languages.

The results in [Sch91] are obtained by establishing a correspondence between concept languages and Propositional Dynamic Logics (PDL), and reducing the given problem to a satisfiability problem in PDL. Such an approach allows Schild to find several new results exploiting known results in the PDL framework. However, it cannot be used to deal with every concept language.
In fact, the correspondence is not established for the languages including some concept constructors having no counterpart in PDL. A proposal to extend the correspondence with PDL to deal with number restrictions and individuals in an ABox is given in [DL94a].

Recently, an algebraic approach to cycles has been proposed in [DMO92], in which (possibly cyclic) definitions are interpreted as determining an equivalence relation over the terms describing concepts. The existence and uniqueness of such an equivalence relation derives from Aczel’s results on non-well founded sets [Acz88]. In [DMO93], the same researchers prove that, on this approach, Subsumption is equivalent to Subsumption in greatest-fixed point semantics. The language analyzed is a small fragment of the one used in K-REP, and contains conjunction and existential-universal quantifications combined into one constructor (hence is similar to $\mathcal{FL}_0$). The difficulty of extending these results lies also in the fact that it is not clear how individuals can be interpreted in this algebraic setting. Moreover, we believe that constructive approaches like the algebraic one, give counterintuitive results when applied to non-constructive features of concept languages—as negation and number restrictions.

The role of the ABox in Concept Satisfiability and Subsumption

In [Neb90a] it is shown that the ABox plays no active role when checking Concept Satisfiability and Subsumption. More precisely, the following results hold for every satisfiable ABox $\mathcal{A}$:

$$
\langle T, \mathcal{A} \rangle \not\models C \iff \langle T, \emptyset \rangle \not\models C \\
\langle T, \mathcal{A} \rangle \models C \subseteq D \iff \langle T, \emptyset \rangle \models C \subseteq D
$$

Relations 3.9 and 3.10 show that the ABox (subject to its satisfiability) can be neglected when checking Concept Satisfiability and Subsumption.

It follows that Concept Satisfiability and Subsumption in knowledge bases comprising the ABox can be reduced to the corresponding ones without the ABox.

It is important to remind that Relations 3.9 and 3.10, together with Relations (3.2-3.4), are stated for pure languages, i.e. languages without $\mathcal{O}$ and $\mathcal{B}$. Those constructors, in fact, allowing the use of individuals in the TBox create an interaction between the individuals in the ABox with the individuals in the TBox (see Chapter 5).

3.2.3 Complexity of Consistency and Instance Checking

Consistency and Instance Checking in the case of empty ABox are problems of no real interest. In fact, the consistency of a TBox is either trivial or it can
be reduced to the other problems, depending upon the particular TBox mechanism involved. Instance Checking, instead, always reduces to the problem of checking whether the concept is equivalent to $\top$. We therefore consider only the cases in which the ABox is non-empty. However, before showing the complexity results for ABox- and KB-reasoning, we present a reasoning technique extensively used in actual systems to deal with Instance Checking.

Abstraction/Subsumption technique

Abstraction is a well known mechanism for reasoning about individuals in concept-based systems. It consists in retrieving all the assertions in the knowledge base $\Sigma$ relevant to a given individual $a$ and collecting them into a single concept. Such concept, that we denote with $C_{\Sigma[a]}$, can be obtained as the conjunction of two parts: One representing the concept memberships of $a$, denoted by $CC_{\Sigma[a]}$, and the other representing its role relationships, denoted by $RC_{\Sigma[a]}$.

Abstraction, together with Subsumption, allows us to perform Instance Checking. In fact, given the problem of checking whether $\Sigma \models C(a)$, with the abstraction process, we compute $C_{\Sigma[a]}$ and, after that, Instance Checking can be performed by checking whether $C$ subsumes $C_{\Sigma[a]}$. This technique, called Abstraction/Subsumption, has been broadly exploited in actual systems (see [Kin90, QK90, Neb90a]).

In some actual systems, for all individuals $a$, the concept $C_{\Sigma[a]}$ is explicitly stored and kept up-to-date when the knowledge base is modified. In order to minimize the space needed, the concept $C_{\Sigma[a]}$ is usually modified to represent only the most specific concepts of which $a$ is an instance. In this case, $C_{\Sigma[a]}$ is indicated as $MSC(a)$, and its computation becomes more complex, involving Subsumption tests among its conjuncts.

We do not pursue this idea, although in some cases we employ the Abstraction/Subsumption technique, combining it with the constraint systems calculus. We discuss in Chapter 4, the possible ways to compute $CC_{\Sigma[a]}$ and $RC_{\Sigma[a]}$ with the use of constraint systems. In Chapter 4 we also discuss the limits of this technique and we explain why it is not suitable to reason with any concept language.

Complexity of ABox-reasoning

Since Consistency and Instance Checking are at least as hard as Concept Satisfiability and Subsumption, respectively, then all the lower bounds for the latter problems directly apply to the former ones. In particular, those for Concept Satisfiability apply to Consistency and those for Subsumption apply for Instance Checking.
We see now an upper bound for $\mathcal{ALC}$ and $\mathcal{ALCN}$ obtained by Baader and Hollunder, within their work on the $\mathbb{KR}s$ system. In [BH91b] a PSPACE algorithm for checking Consistency in $\mathcal{ALC}$ is given, which is based on the constraint system calculus. That algorithm relies on the corresponding one for Pure Concept Satisfiability in $\mathcal{ALC}$ given in [SS91]. It also relies on the notion of pre-completion, which is extensively explained in Chapter 4.

Since for any $\mathcal{ALC}$-concept $C$ we have $\Sigma \models C(a)$ if and only if the knowledge base $\Sigma \cup \{\neg C(a)\}$ in $\mathcal{ALC}$ is unsatisfiable, it follows that Instance Checking in $\mathcal{ALC}$ can be done in polynomial time, too. Being Pure Concept Satisfiability in $\mathcal{ALC}$ already complete for PSPACE, it follows that those problems are PSPACE-complete, too.

Recently, Hollunder in [Hol93], proved that ABox Consistency is PSPACE-complete for $\mathcal{ALCN}$, too. The complexity of more expressive languages, e.g. $\mathcal{ALCN}^R$ is still under investigation.

Unfortunately, those are the only results available. In fact, the algorithms employed by the other actual systems (e.g. BACK, and LOOM), being incomplete, do not supply us any help, because we focus only on complete reasoning.

Complexity of KB-reasoning

In Section 3.2.2 we have shown that, by means of the transformation $U(\cdot)$, it is possible to reduce TBox-reasoning to pure-reasoning, in the case of acyclic TBox.

By means of $U(\cdot)$, it is also possible to reduce general Consistency and Instance Checking to the ABox-reasoning case of them. In fact, in those problems the acyclic TBox can be removed, in particular by substituting in every assertion of the ABox every concept name with its definition (see [Neb90a]). It follows that the results for ABox-reasoning therefore apply for KB-reasoning, too.

The problem of reasoning with a general TBox (or free TBox) and an ABox together has not been investigated in the literature. Some results are given in this thesis in Chapter 7.
Chapter 4

Complexity of Consistency and Instance Checking

We have seen in the previous chapter that Concept Satisfiability and Subsumption can be reduced to Pure Concept Satisfiability and Pure Subsumption, whereas Consistency and Instance Checking can be reduced to ABox Consistency and ABox Instance Checking.

Based on these observations, in this chapter and the following two, we focus on those special cases. Moreover, for the sake of simplicity, throughout this chapter and Chapters 5 and 6, we write Consistency and Instance Checking instead of ABox Consistency and ABox Instance Checking; similarly, when we write Concept Satisfiability and Subsumption we refer to Pure Concept Satisfiability and Pure Subsumption.

The central question we address in this chapter is to understand whether Instance Checking can be easily reduced to Subsumption.

The basic technique underlying most of the approaches developed so far, is the Abstraction/Subsumption one. This idea is applicable to a number of languages. However, we have discovered cases where, in order to check whether $\Sigma \models D(a)$, it is necessary to consider assertions about other objects in the knowledge base different from $a$, and the above method is no longer applicable.

In our analysis, we consider several languages for which Subsumption belongs to different complexity classes. In particular, we focus on $\mathcal{ALC}$, $\mathcal{ALN}$, $\mathcal{ALCE}$, $\mathcal{ALCR}$, and $\mathcal{ALC}$, which show a variety of different interesting behaviors. Languages including the constructors $O$ and $B$ are considered in next chapter.

The chapter is organized as follows. Section 4.1 deals with Consistency, and shows that this problem can be solved by exploiting the corresponding algorithms for Concept Satisfiability. Section 4.2 contains the complexity anal-
ysis of Instance Checking, providing both algorithms for computing Instance Checking via Subsumption, and hardness proofs, in those cases where Instance Checking belongs to a higher complexity class with respect to Subsumption. A final discussion summarizes the results on the computational complexity of Instance Checking in concept languages. Section 4.3 concludes the chapter with a few considerations on the implications of the results presented in the chapter for the research on concept languages.

The results of this chapter have been presented in [DLNS94], except for the first part of Section 4.2.2 that appeared in [Sch93]. The results for the language $\mathcal{AL}$ have been originally found in [LS91a], using a similar technique.

### 4.1 Complexity of Consistency

In this section, we deal with the computational complexity of Consistency, i.e., the problem of checking whether an knowledge base in $\mathcal{L}$ is consistent, where $\mathcal{L}$ is any of the concept languages considered above.

Before presenting the result of our analysis, we need to introduce some notions. Let $\Sigma$ be a knowledge base and $S_\Sigma$ the constraint system associated with it. A constraint system $S$ is said to be a pre-completion of $S_\Sigma$ if it is obtained from $S_\Sigma$ by the application of the non-generating rules, and none of these rules is applicable to $S$ (see [BH91b]). Note that any pre-completion $S$ of $S_\Sigma$ contains no variables, and, moreover, all the individuals in $S$ are also in $S_\Sigma$. Moreover, when $\Sigma$ is expressed in a language without disjunction, there is only one pre-completion of $S_\Sigma$.

If $S$ is any constraint system, and $b$ is an individual in $S$, then we denote by $CC_{S|b}$ the concept constituted by the conjunction of all the concepts $C$ such that $b : C$ is in $S$. If no constraint of the form $b : C$ is in $S$, then $CC_{S|b}$ is simply $\top$. Also, we denote by $RC_{S|b}$ the concept $\exists R_1 \land \cdots \land \exists R_m$, where the $R_i$'s ($i = 1, \ldots, m$) are all the roles such that $n_{R_i,S}(b) > 0$ (where $n_{R_i,S}(a)$ denotes the number of individuals $b$ such that $aRb \in S$). If for all $R_i$, $n_{R_i,S}(b) = 0$ then $RC_{S|b}$ is simply $\top$. Thus, $RC_{S|b}$ is the concept that accounts for the relationships involving $b$ in $S$. Finally, we denote $C_{S|b}$ the conjunction of $CC_{S|b}$ and $RC_{S|b}$.

We often use the concepts $CC_{S|b}$, $RC_{S|b}$, and $C_{S|b}$ when $S$ is a pre-completion of the constraint system $S_\Sigma$ derived from the knowledge base $\Sigma$. Notice that, when there is only one pre-completion $S$ of $S_\Sigma$, the concepts $CC_{S|b}$, $RC_{S|b}$, and $C_{S|b}$, where $b$ is an individual in $S_\Sigma$, are uniquely identified, and they correspond to $CC_{\Sigma|b}$, $RC_{\Sigma|b}$, and $C_{\Sigma|b}$ introduced in Chapter 3.

In the following, we make use of such notion of pre-completion of $S_\Sigma$ in order to derive complexity results regarding Instance Checking in $\Sigma$. For this reason, we are interested in evaluating the complexity of computing a pre-
completion. The next proposition shows that the size of any pre-completion of $S_\Sigma$ is polynomially bounded by the size of $\Sigma$.

**Proposition 4.1.1** Let $S$ be a constraint system and $S'$ be a pre-completion of $S$. Then $|S'|$ is bounded by $|S|^3$.

*Proof.* First notice that during the computation of the pre-completion no constraints of the form $vPw$ are generated, because the $\rightarrow_3$- and $\rightarrow_2$-rules are never applied. Moreover, the number of concepts appearing in new constraints which are introduced by the application of the rules to a constraint of the form $a : C$ is bounded by the number of subconcepts of $C$ and therefore by $|C|$. Thus, if all the constraints of the form $a : C$ in $S$ are $a_1 : C_1, \ldots, a_n : C_n$, then the number of new concepts that are introduced when computing the pre-completion of $S$ is bounded by $|C_1| + \cdots + |C_n|$. If there are $m$ individuals in $S$, the number of different constraints that can be added to $S$ is at most $m \times (|C_1| + \cdots + |C_n|)$. Since $|C_1| + \cdots + |C_n| < |S|$ and $m < |S|$, the number of constraints in the pre-completion of $S$ is bounded by $|S|^2$. The claim follows from the fact that the size of each constraint is obviously bounded by $|S|$.

It easily follows from Propositions 3.1.3 and 4.1.1 that any pre-completion of $S_\Sigma$ can be computed in polynomial time with respect to $|\Sigma|$, and, moreover, for every individual $b$ in $\Sigma$, both $CC_{S|b}$ and $RC_{S|b}$ can be computed in polynomial with respect to $|\Sigma|$.

The notions of pre-completion and $CC_{S|b}$ are useful for characterizing the consistency of a knowledge base, as shown in the following proposition, whose proof can be easily derived from the results in [BH91b].

**Proposition 4.1.2** Let $\mathcal{L}$ be one of the concept languages $\mathcal{ALC}$, $\mathcal{ALC}^N$, $\mathcal{ALC}^E$, $\mathcal{ALC}^R$, $\mathcal{ALC}^U$, $\mathcal{ALC}$. A knowledge base $\Sigma$ in $\mathcal{L}$ is consistent if and only if there is a clash-free pre-completion $S$ of $S_\Sigma$ such that for each individual $b$ in $\Sigma$, $CC_{S|b}$ is satisfiable.

This proposition ensures us that, when dealing with the languages we focus in this chapter, in order to check the consistency of $\Sigma$ one has to look for a clash-free pre-completion $S$ of $S_\Sigma$, extract the various concepts $CC_{S|b}$, and independently check them for satisfiability. It is worth notice that $RC_{S|b}$ plays no role in this case.

**Example 4.1.3** Let $\Sigma$ be the following knowledge base in $\mathcal{ALC}$:

$$\Sigma = \{\text{FRIEND}(\text{john}, \text{susan}),$$
$$\neg \text{Graduate}(\text{susan}),$$
$$\forall \text{FRIEND}(\text{Graduate} \sqcup \text{Happy})(\text{john})\}$$
The knowledge base $\Sigma$ specifies that Susan is a friend of John, Susan is not graduate, and all the friends of John are either graduate or happy.

The two pre-completions $S_1, S_2$ that can be obtained from $S_\Sigma$ are shown in Figure 4.1. It is easy to see that $S_1$ contains a clash, while $S_2$ does not. Moreover, the concepts

$CC_{S_2}[john] = \forall\text{FRIEND}.\text{(Graduate} \sqcup \text{Happy})$

$CC_{S_2}[\text{susan}] = \neg\text{Graduate} \sqcap (\text{Graduate} \sqcup \text{Happy}) \sqcap \text{Happy}$

are clearly satisfiable, implying that $\Sigma$ is consistent.

We now turn our attention to the computational complexity of Consistency. Proposition 4.1.2 shows that it depends upon two factors: the complexity of computing a pre-completion, and the complexity of Concept Satisfiability. Based on this property, the next proposition shows that, for the languages considered in this chapter, Consistency has the same computational complexity as Concept Satisfiability.

**Proposition 4.1.4** Let $\mathcal{L}$ be one of the concept languages $\mathcal{AL}, \mathcal{ALN}, \mathcal{ALE}, \mathcal{ALR}, \mathcal{ALU}, \mathcal{ALC}$. Then Consistency and Concept Satisfiability are in the same complexity class.

**Proof.** Notice first of all that, since Concept Satisfiability is reducible to Consistency, Consistency is at least as hard as Concept Satisfiability. The proof that Consistency is not harder than Concept Satisfiability is structured by cases, depending on the complexity of Concept Satisfiability:

1. Languages with polynomial Concept Satisfiability ($\mathcal{AL}, \mathcal{ALN}$). Since there is only one pre-completion for knowledge bases expressed in these languages, and Concept Satisfiability in both $\mathcal{AL}$ and $\mathcal{ALN}$ can be solved in polynomial time, it follows from Proposition 4.1.2 that we can easily devise a deterministic polynomial time algorithm for both Consistency in $\mathcal{AL}$ and Consistency in $\mathcal{ALN}$. Such an algorithm is shown in Figure 4.2, where $CSat_L$ denotes any polynomial algorithm for Concept Satisfiability in $\mathcal{L}$, where $\mathcal{L}$ is either $\mathcal{AL}$ or $\mathcal{ALN}$.
2. Languages with coNP-complete Concept Satisfiability ($\mathcal{ALC}, \mathcal{ALC}$). In order to derive the complexity of Consistency in $\mathcal{ALC}$ and Consistency in $\mathcal{ALC}$, it is convenient to refer to the complement of Consistency. Notice that there is only one pre-completion for knowledge bases expressed in these languages. This property allows us to derive an NP algorithm for checking the inconsistency of a knowledge base $\Sigma$ as follows. The algorithm first computes the pre-completion $S$ of $S_\Sigma$, and then chooses nondeterministically an individual $b$ of $S_\Sigma$, nondeterministically computes a trace of $\{x : CC_{S[b]} \}$ (with the $\rightarrow_T$-rule instead of the $\rightarrow_3$-rule), and checks whether such a trace contains a clash. Since the size of $CC_{S[b]}$ is polynomial with respect to the size of $\Sigma$, it follows that the above is an NP algorithm for checking the inconsistency of $\Sigma$.

3. Languages with NP-complete Concept Satisfiability ($\mathcal{ALU}$). An NP algorithm for checking the consistency of a knowledge base $\Sigma$ expressed in this language can be derived as follows. The algorithm simply computes nondeterministically a pre-completion $S$ of $S_\Sigma$, and then, for each individual $b_i$ in $S_\Sigma$ ($1 \leq i \leq n$), nondeterministically computes a completion $T_i$ of $\{b_i : CC_{S[b_i]} \}$. Finally, it checks whether the constraint system $T_1 \cup \cdots \cup T_n$ is clash-free. The answer will be TRUE if such a constraint system is clash-free. Again, since the size of each $CC_{S[b_i]}$ is polynomial with respect to $|\Sigma|$, this gives an NP algorithm for checking the consistency of $\Sigma$.

4. Languages with PSPACE-complete Concept Satisfiability ($\mathcal{ALC}$). An algorithm in PSPACE for checking the consistency of a knowledge base $\Sigma$ expressed in $\mathcal{ALC}$ can be easily derived from Proposition 4.1.2. The algorithm systematically searches a precompleteness $S$ of $S_\Sigma$ such that for each individual $b$, the satisfiability checking of $CC_{S[b]}$ (done with the PSPACE algorithm) returns true.

Notice that the result on Consistency in $\mathcal{ALC}$ have been reported in [BH91b].

4.2 Complexity of Instance Checking

In this section we investigate on the computational complexity of Instance Checking in the concept languages defined in the previous section. Specifically, we address Instance Checking starting from polynomial languages and then moving to languages where Concept Satisfiability and Subsumption belong to higher complexity classes. While in Section 4.1 we have considered the complexity of Consistency with respect to Concept Satisfiability, here we aim
Algorithm $Cons_{\mathcal{L}}(\Sigma)$
Input knowledge base $\Sigma$ in $\mathcal{L}$, where $\mathcal{L}$ is either $\mathcal{AL}$ or $\mathcal{ALN}$
Output TRUE if $\Sigma$ is consistent, FALSE otherwise
begin
    compute the pre-completion $S$ of $S_\Sigma$;
    if for each $b \in \mathcal{O}_\Sigma$, $CSat(CC_{\Sigma \mid b}) \equiv$ TRUE
    then return TRUE
    else return FALSE
end

Figure 4.2: The algorithm for Consistency in $\mathcal{AL}$ and $\mathcal{ALN}$

at comparing the complexity of Instance Checking with respect to Subsumption, based on the fact that Instance Checking is a more general problem than Subsumption.

Our ultimate goal is to understand whether Instance Checking can be solved by means of Subsumption algorithms. In fact, we will show that, in general, Instance Checking is more difficult than Subsumption, and there are cases where this is precisely demonstrated by the fact that Instance Checking belongs to a higher complexity class compared to Subsumption.

4.2.1 Languages with Polynomial Subsumption ($\mathcal{AL, ALN}$)

In this subsection we analyze the computational properties of Instance Checking for languages where both Concept Satisfiability and Subsumption can be solved in polynomial time. We show that Instance Checking in both $\mathcal{AL}$ and $\mathcal{ALN}$ retain tractability with respect to combined complexity. Obviously, since combined complexity is more general than knowledge base complexity, this result implies that Instance Checking in these languages is polynomial with respect to knowledge base complexity as well.

We begin our analysis with the language $\mathcal{AL}$. As already mentioned, for languages not including disjunction, only one pre-completion can be obtained from a knowledge base. A polynomial time algorithm for Instance Checking in $\mathcal{AL}$ can then be devised based on the following well known idea (Abstraction/Subsumption): in order to check whether $\Sigma \models D(a)$, where $\Sigma$ is consistent, first compute the pre-completion $S$ of $S_\Sigma$, and then build the concept $C_{\Sigma \mid a}$, representing all the relevant information about the individual $a$, according to $S$. Notice that concept $CC_{\Sigma \mid a}$ does not capture all the properties of
Algorithm \textit{Inst}_{\mathcal{AC}}(\Sigma, a, D)
\textbf{Input} knowledge base \Sigma in \mathcal{AC}, individual \(a\), \mathcal{AC}-concept \(D\)
\textbf{Output} \text{TRUE} if \(\Sigma \models D(a)\), \text{FALSE} otherwise
\begin{verbatim}
begin
  if \(\text{Cons}_{\mathcal{AC}}(\Sigma) = \text{FALSE}\) then return \text{TRUE};
  compute \(C_{\Sigma|a}\);
  if \(\text{Subs}_{\mathcal{AC}}(D, C_{\Sigma|a})\) then return \text{TRUE}
  else return \text{FALSE}
end
\end{verbatim}

Figure 4.3: The algorithm for Instance Checking in \mathcal{AC}

\(a\) in \(S\). Indeed, any assertion of the form \(R(a, b)\) in \(\Sigma\) implies that \(a\) is an instance of \(\exists R\), and this is captured by \(RC_{\Sigma|a}\). The solution is then obtained by checking whether \(D\) subsumes \(C_{\Sigma|a}\). The following proposition shows that such a method is correct.

**Proposition 4.2.1** Let \(\Sigma\) be a consistent knowledge base in \mathcal{AC}, \(a\) an individual, and \(D\) an \mathcal{AC}-concept. Then \(\Sigma \models D(a)\) if and only if \(D\) subsumes \(C_{\Sigma|a}\).

**Proof.**

\(\Rightarrow\) Suppose \(\Sigma \models D(a)\) and \(C_{\Sigma|a} \nsubseteq D\). It follows that the concept \(C_{\Sigma|a} \cap \neg D\) is satisfiable, and, therefore, there is a clash-free completion \(S'\) of the constraint system \(\{a : C_{\Sigma|a} \cap \neg D\}\). Now, let \(S\) be the completion of \(S_{\Sigma}\). Since \(\Sigma\) is consistent, \(S\) is clash-free. Consider the constraint system \(S \cup S'\). It is easy to see that \(S \cup S'\) is clash-free, and, moreover, it contains a completion of \(S_{\Sigma} \cup \{a : \neg D\}\), thus contradicting the hypothesis that \(\Sigma \not\models D(a)\).

\(\Leftarrow\) Suppose \(C_{\Sigma|a} \subseteq D\). It follows that every model of \(C_{\Sigma|a}(a)\) is also a model of \(D(a)\). Since every model of \(\Sigma\) is clearly a model of \(C_{\Sigma|a}(a)\) too, we can conclude that every model of \(\Sigma\) is also a model of \(D(a)\), and, therefore, \(\Sigma \models D(a)\).

In Figure 4.3 we show the algorithm \textit{Inst}_{\mathcal{AC}} for Instance Checking in \mathcal{AC}, where \(\text{Subs}_{\mathcal{AC}}\) is a function that takes two \mathcal{AC}-concepts as argument, \(E_1\) and \(E_2\), respectively, and checks in polynomial time whether \(E_1\) subsumes \(E_2\).

The correctness of \textit{Inst}_{\mathcal{AC}} follows from Proposition 4.2.1. With regard to the complexity, since both \(\text{Cons}_{\mathcal{AC}}(\Sigma)\) and computing the concept \(C_{\Sigma|a}\), can be done in polynomial time with respect to \(|\Sigma|\), and, moreover, \(\text{Subs}_{\mathcal{AC}}(E_1, E_2)\)
works in polynomial time with respect to \(|E_1|\) and \(|E_2|\), it follows that \(\text{Inst}_{\mathcal{AC}}\) is polynomial with respect to the combined complexity.

The interesting feature of \(\text{Inst}_{\mathcal{AC}}\) is to rely on the algorithm for Subsumption in \(\mathcal{AC}\), thus allowing for a careful reuse of the answers of previous queries. Indeed, observe that the only task that the algorithm performs besides Subsumption is to compute the pre-completion of \(S_\Sigma\) and the concept \(C_{\Sigma | \text{ta}}\), which is a minimal requirement for Instance Checking (see [Neb90a]).

From all the above observations, we can conclude that Instance Checking in \(\mathcal{AC}\) can be really solved by relying on the algorithm for Subsumption. Unfortunately, not all polynomial languages exhibit this property. In particular we now consider the language \(\mathcal{ACN}\), for which both Concept Satisfiability and Subsumption can be solved in polynomial time, and show that Instance Checking in \(\mathcal{ACN}\) is not easily reducible to Subsumption in \(\mathcal{ACN}\). Let us introduce this topic by means of an example showing that, by applying the method used for \(\text{Inst}_{\mathcal{AC}}\), one obtains an incomplete algorithm for \(\mathcal{ACN}\).

**Example 4.2.2** Let \(\Sigma\) be the following knowledge base in \(\mathcal{ACN}\):

\[
\Sigma = \{ \text{FRIEND}(\text{john}, \text{susan}), \text{FRIEND}(\text{john}, \text{peter}), \\
\text{Graduate}(\text{susan}), \text{Graduate}(\text{peter}),
\langle \leq 2 \text{ FRIEND}(\text{john}) \}
\]

The knowledge base \(\Sigma\) specifies that John has two friends, namely Susan and Peter, and that Susan and Peter are graduate. Furthermore the number of friends of John is limited to two.

Consider the \(\mathcal{ACN}\)-concept \(\forall \text{FRIEND}. \text{Graduate}\). It is easy to see that the following logical implication is valid \(\Sigma \models \forall \text{FRIEND}. \text{Graduate}(\text{john})\). The key observation here is that this conclusion cannot be inferred only from the assertions about John. Instead, one has to consider the assertions about the other individuals, in particular \(\text{Graduate}(\text{susan})\) and \(\text{Graduate}(\text{peter})\). In other words, we have that \(\Sigma \models \forall \text{FRIEND}. \text{Graduate}(\text{john})\), even though the concept \(\forall \text{FRIEND}. \text{Graduate}\) does not subsume \(C_{\Sigma | \text{john}}\) (which is \((\exists \text{FRIEND}) \cap \langle \leq 2 \text{ FRIEND} \rangle)\). \(\square\)

The example shows the reason why an algorithm of the kind of \(\text{Inst}_{\mathcal{AC}}\) is not adequate for dealing with \(\mathcal{ACN}\), namely the interaction between number restrictions and universal quantification. Nevertheless, Instance Checking in \(\mathcal{ACN}\) can be solved in polynomial time, as shown in the rest of this subsection.

First of all, notice that every \(\mathcal{ACN}\)-concept \(D\) can be rewritten into an equivalent concept of the form \(D_1 \cap \cdots \cap D_n\), where each \(D_i\) is conjunction-free (i.e. does not contain the symbol \(\cap\)), and is called a *conjunction-free*
4.2 Complexity of Instance Checking

component of $D$. This can be done by exploiting the equivalence:

$$\forall R. (C \sqcap D) \equiv \forall R. C \sqcap \forall R. D$$

It follows that $\Sigma \models D(a)$ if and only if for every conjunction-free component $D_i$ of $D$, $\Sigma \models D_i(a)$. Therefore, we can limit our attention to the case where the concept $D$ is conjunction-free, and in particular is of the form $\forall R_1, \forall R_2, \ldots, \forall R_n, \alpha$, where $\alpha$ is not of the form $\forall R. C$. Moreover, we distinguish between the case where $n = 0$ and the case where $n > 0$.

In the first case, by looking at the form of the concept $D$ —which is $\top, \bot, A, \neg A, (\geq n R)$, or $(\leq n R)$— one realizes that $\neg D$ (properly rewritten in a simple form) belongs to $\mathcal{ALN}$, and therefore $\Sigma \models D(a)$ can be checked by testing the knowledge base $\Sigma \cup \{\neg D(a)\}$ in $\mathcal{ALN}$ for inconsistency.

In the second case, the situation is complicated by the interaction between the universal quantification in the query and the number restrictions in the knowledge base (as pointed out by Example 4.2.2). The following proposition shows how concepts containing universal quantification can be handled (where $n_{R,\Sigma}(a)$ denotes the number of individuals $b$ such that $R(a, b) \in \Sigma$).

**Proposition 4.2.3** Let $\Sigma$ be a consistent knowledge base in $\mathcal{ALN}$, $a$ an individual, $R$ a role and $E$ a conjunction-free $\mathcal{ALN}$-concept. Let $m$ be the minimum of the set $\{k \mid (\leq k R) \text{ is a conjunct of } CC_{\Sigma|a}\}$, with the assumption that $m$ is $\omega$ (infinity) if no conjunct of the form $(\leq k R)$ is in $CC_{\Sigma|a}$. Then

1. If $n_{R,\Sigma}(a) < m$, then $\Sigma \models \forall R. E(a)$ if and only if $\forall R. E$ subsumes $CC_{\Sigma|a}$.

2. If $n_{R,\Sigma}(a) = m$, then $\Sigma \models \forall R. E(a)$ if and only if for all $b$ such that $R(a, b) \in \Sigma$: $\Sigma \models E(b)$.

**Proof.**

1. If $\forall R. E$ subsumes $CC_{\Sigma|a}$, then (analogously to Proposition 4.2.1) it follows that $\Sigma \models \forall R. E(a)$. On the other hand, assume that $CC_{\Sigma|a} \not\subseteq \forall R. E$. It follows that the concept $CC_{\Sigma|a} \sqcap \exists R. \neg E$ is satisfiable, and therefore, there is a clash-free completion $S'$ of the constraint system $\{a : CC_{\Sigma|a} \sqcap \exists R. \neg E\}$. Now, let $S$ be the clash-free completion of $S_{\Sigma}$. Consider the constraint system $S \cup S'$. If $n_{R,\Sigma \cup S'}(a) \leq m$, then it is easy to see that $S \cup S'$ is clash-free and, moreover, it contains a clash-free completion of $S_{\Sigma} \cup \{a : \exists R. \neg E\}$. Therefore $\Sigma \not\models \forall R. E(a)$. If $n_{R,\Sigma \cup S'}(a) > m$, then every variable $z$ such that $aRz$ holds in $S \cup S'$ can be safely replaced with the variable $y$ introduced by the application of the $\rightarrow_{\Sigma}$ rule to the constraint $a : \exists R. \neg E$. Since $n_{R,\Sigma}(a) < m$, by a suitable number of the above replacements, it is possible to derive from $S \cup S'$ a complete clash-free constraint system $S''$ that contains a clash-free completion of $S_{\Sigma} \cup \{a : \exists R. \neg E\}$, and is such that $n_{R, S''}(a) = m$. Therefore $\Sigma \not\models \forall R. E(a)$. 


2. If there is a \( b \) such that \( R(a, b) \in \Sigma \) and \( \Sigma \not\models E(b) \), then it is obvious that \( \Sigma \not\models \forall R.E(a) \). On the other hand, assume that for all \( b_i \) (1 \( \leq i \leq m \)) such that \( R(a, b_i) \in \Sigma, \Sigma \models E(b_i) \). Since the concept \( \leq m R \) is a conjunct of \( CC_{\Sigma|a} \), it follows that \( \Sigma \models \forall R.E(a) \).

The above proposition shows that in order to check whether \( \Sigma \models \forall R.D(a) \), we may have to consider every individual \( b \) such that \( R(a, b) \) is in \( \Sigma \), and check whether \( \Sigma \models D(b) \). Notice that a blind computation of all the required checks might result in a number of checks which is \( K^{|D|} \), where \( K \) is the cardinality of \( O_{\Sigma} \). However, we show that this can be avoided by means of a careful algorithm based on a form of dynamic programming.

The algorithm, called \( Inst_{ACN} \), is shown in Figure 4.4. It relies on the polynomial algorithm \( Subs_{ACN} \) to handle case 1 of Proposition 4.2.3, while uses dynamic programming in dealing with case 2. Specifically, \( Inst_{ACN} \) makes use of a data structure \( INST.OF \) which stores a value in the set \{unknown, true, false\} for every pair \((a, E)\), where \( a \) is an individual and \( E \) is a subconcept of the query \( D \). Informally speaking, \( INST.OF[a, E] \) is used to record the answer to the test \( \Sigma \models E(a) \): the value unknown represents that no answer has yet been computed for the test, whereas true and false have the obvious meaning of yes and no, respectively.

**Proposition 4.2.4** Let \( \Sigma \) be a knowledge base in \( ACN \), \( a \) be an individual, and \( D \) be a \( ACN \)-concept. Then \( Inst_{ACN}(\Sigma, a, D) \) terminates, returning true if \( \Sigma \models D(a) \), and false otherwise. Moreover, it runs in polynomial time with respect to \(|\Sigma|\) and \(|D|\).

**Proof.** The correctness of the algorithm easily follows from Proposition 4.2.3. Termination follows from the fact that in any recursive call of the algorithm the actual parameter corresponding to \( D \) decreases in length. With respect to complexity, first of all notice that \( Subs_{ACN} \) has polynomial time complexity. Hence, any recursive call issued during the execution of the algorithm \( Inst_{ACN}(\Sigma, a, D) \) performs a number of operations polynomially bounded by \(|\Sigma| \times |D| \). It remains to show that the number of recursive calls that are issued during \( Inst_{ACN}(\Sigma, a, D) \) is polynomial with respect to \(|\Sigma| \times |D| \). This follows from the fact that, due to the use of the data structure \( INST.OF \), for any pair \((b, E)\), where \( b \) is an individual in \( \Sigma \), and \( E \) is a subconcept of \( D \), at most one call to \( RESULT(a, E) \) can be issued during the execution of \( Inst_{ACN}(\Sigma, a, D) \). Therefore, the number of calls of \( RESULT \) is bounded by \(|O_{\Sigma}| \times |D| \).

### 4.2.2 Languages with NP-complete Subsumption (\( AC\mathcal{E}, AC\mathcal{R} \))

We start this subsection with the language \( AC\mathcal{E} \). In particular, we show that in \( AC\mathcal{E} \) Instance Checking and Subsumption do not belong to the same
Algorithm $Inst_{\mathcal{ALCN}}(\Sigma, a, D)$

Input knowledge base $\Sigma$ in $\mathcal{ALCN}$, individual $a$, $\mathcal{ALCN}$-concept $D$

Output TRUE if $\Sigma \models D(a)$, FALSE otherwise

begin
if $Cons_{\mathcal{ALCN}}(\Sigma) =$FALSE then return TRUE;
compute the pre-completion $S$ of $S_\Sigma$;
foreach $(b, E)$, where $b$ is an individual in $\Sigma$, and $E$ is a subconcept of $D$
do
$INST.OF[b, E] := unknown$;
let $D_1 \land \cdots \land D_p$ be the conjunction-free form of $D$
in return $RESULT(\Sigma, a, D_1) \land \cdots \land RESULT(\Sigma, a, D_p)$
end

Function $RESULT(\Sigma, b, D)$: boolean;

begin
if $INST.OF[b, D] =$unknown
then if $D = \forall R. E$
then begin
$m := \min \{ k \mid (\leq k R) \text{ is a conjunct of } CC_{\Sigma, b} \}$
if $n_{R, \Sigma}(b) < m$
then $INST.OF[b, D] := Subs_{\mathcal{ALCN}}(D, CC_{\Sigma, b})$
else let $b_1, \ldots, b_n$ ($n \geq 0$) be such that $R(b, b_i) \in \Sigma$;
in if $n = 0$
then $INST.OF[b, D] := TRUE$
else $INST.OF[b, D] :=$\begin{align*}
RESULT(\Sigma, b_1, E) \land \cdots \land RESULT(\Sigma, b_n, E)
\end{align*}
end
else $INST.OF := Cons_{\mathcal{ALCN}}(\Sigma \cup \{ \neg D(b) \})$;
return $INST.OF[b, D]$\end{align*}
end

Figure 4.4: The algorithm for Instance Checking in $\mathcal{ALCN}$
complexity class, neither with respect to knowledge base complexity nor with respect to combined complexity. Subsequently, we consider the language $ACR$ and we show that instead Instance Checking in $ACR$ is NP-complete with respect to both complexity measures, hence in the same complexity class as Subsumption in $ACR$.

**Knowledge base complexity of Instance Checking in $ALE$**

In this section we give a lower and an upper bound for the data complexity of Instance Checking in $ALE$. As shown above, Instance Checking in $ALE$ is NP-hard. We now prove that it is coNP-hard too. Since Subsumption in $ALE$ is NP-complete, a consequence of this result is that (assuming $NP \neq coNP$) Instance Checking for $ALE$ is strictly harder than Subsumption.

This unexpected result shows that the Instance Checking problem in $ALE$ suffers from a new source of complexity, which does not show up when checking Subsumption between $ALE$-concepts. This new source of complexity is related to the use of qualified existential quantification in the concept representing the query, which makes the behavior of the individuals dependent on the other individuals in the knowledge base. The following example enlights this point.

**Example 4.2.5** Let $\Sigma$ be the following knowledge base in $ALE$:

$$\Sigma = \{FRIEND(john, susan), FRIEND(john, peter),$$
$$LOVES(susan, peter), LOVES(peter, mary),$$
$$Graduate(susan), \neg\text{Graduate}(mary)\}$$

Consider now the following assertion

$$\beta = \exists\text{FRIEND}.(\text{Graduate}\land\exists\text{LOVES}.\neg\text{Graduate})(john).$$

Asking whether $\Sigma \models \beta$ means asking whether John has a graduate friend who loves a not graduate person. At the first glance, since Susan and Peter are the only known friends of John, it seems that the answer is to be found by checking whether either $\Sigma \models \text{Graduate}\land\exists\text{LOVES}.\neg\text{Graduate}(susan)$ or $\Sigma \models \text{Graduate}\land\exists\text{LOVES}.\neg\text{Graduate}(peter)$ is true. Since $\Sigma \not\models \text{Graduate}(peter)$, it follows that $\Sigma \not\models \text{Graduate}\land\exists\text{LOVES}.\neg\text{Graduate}(peter)$, and since $\Sigma \not\models \exists\text{LOVES}.\neg\text{Graduate}(susan)$ it follows that $\Sigma \not\models \text{Graduate}\land\exists\text{LOVES}.\neg\text{Graduate}(susan)$.

Reasoning in this way would lead to the answer NO. On the contrary, the correct answer is YES, and in order to find it, one needs to reason by case analysis. In fact, what is asked is if in every model $M$ of $\Sigma$ there is an individual, say $a$, such that $FRIEND(john, a), \text{Graduate}(a)$ and $\exists\text{LOVES}.\neg\text{Graduate}(a)$ are true in $M$. Obviously, in every model $M$ of $\Sigma$, either $\text{Graduate}(peter)$ or
\[ \neg \text{Graduate(peter)} \text{ is true. In the first case, it is easy to see that a is simply peter (and the not graduate person he loves is mary), while in the second case a is susan (and the not graduate person she loves is just peter). Therefore, such an individual a exists in every model of } \Sigma, \text{ and the answer is YES.} \]

In conclusion, even if none of the individuals related to the individual john through the role FRIEND is in the conditions requested, it happens that the combination of the assertions on the individuals (susan and peter) in the knowledge base is such that in every model one or the other is in that conditions. \hfill \square

The previous example shows that, in order to answer to a query involving qualified existential quantification, a sort of case analysis is required. We now show that this kind of reasoning makes Instance Checking in \( \mathcal{ALC} \) coNP-hard with respect to the knowledge base complexity.

The proof is based on a reduction from a suitable variation of the propositional satisfiability problem (SAT) to Instance Checking in \( \mathcal{ALC} \). We define 2+2-CN\( \mathcal{F} \) formula on an alphabet \( P \), a CN\( \mathcal{F} \) formula \( F \) such that each clause of \( F \) has exactly four literals: two positive and two negative ones, where the propositional letters are elements of \( P \cup \{ \text{true}, \text{false} \} \). Furthermore, we call 2+2-SAT the problem of checking whether a 2+2-CN\( \mathcal{F} \) formula is satisfiable.

**Theorem 4.2.6** 2+2-SAT is NP-complete.

**Proof.** The proof is obtained by a reduction from 3-SAT. Given a 3-CN\( \mathcal{F} \) formula \( F \) we obtain a 2+2-CN\( \mathcal{F} \) formula \( F' \) transforming each clause \( C \) of \( F \) according to the following rules:

\[
\begin{align*}
a \lor b \lor c & \implies (a \lor b \lor \neg d \lor \neg \text{true}) \land (c \lor d \lor \neg \text{true} \lor \neg \text{true}) \\
a \lor b \lor \neg c & \implies a \lor b \lor \neg c \lor \neg \text{true} \\
a \lor \neg b \lor \neg c & \implies a \lor \text{false} \lor \neg b \lor \neg c \\
\neg a \lor \neg b \lor \neg c & \implies (\text{false} \lor d \lor \neg a \lor \neg b) \land (\text{false} \lor \text{false} \lor \neg c \lor \neg d)
\end{align*}
\]

where for each clause \( C \) of \( F \), \( d \) is a new letter in \( P \) not appearing in \( F \). It is easy to see that \( F \) is satisfiable if and only if \( F' \) is satisfiable. \hfill \square

Given a 2+2-CN\( \mathcal{F} \) formula \( F = C_1 \land C_2 \land \cdots \land C_n \), where \( C_i = L_{1i}^+ \lor L_{2i}^- \lor \neg L_{1i}^- \lor \neg L_{2i}^+ \), we associate with it a knowledge base \( \Sigma_F \) in \( \mathcal{ALC} \) and a concept \( Q \) as follows. \( \Sigma_F \) has one individual \( t \) for each letter \( L \) in \( F \), one individual \( c \) for each clause \( C_i \), one individual \( f \) for the whole formula \( F \), plus two individuals true and false for the corresponding propositional constants. The roles of \( \Sigma_F \) are \( Cl, P_1, P_2, N_1, N_2 \), and the only concept name is \( A \).

\[
\Sigma_F = \{ A(\text{true}), \neg A(\text{false}) \},
\]
\[ Cl(f, c_1), Cl(f, c_2), \ldots, Cl(f, c_n),
\]
\[ P_1(c_1, l^1_{1+}), P_2(c_1, l^1_{2+}), N_1(c_1, l^1_{1-}), N_2(c_1, l^1_{2-}),
\]
\[ \ldots
\]
\[ P_1(c_n, l^n_{1+}), P_2(c_n, l^n_{2+}), N_1(c_n, l^n_{1-}), N_2(c_n, l^n_{2-}) \}
\]

\[ Q = \exists Cl(\exists P_1, \neg A \cap \exists P_2, \neg A \cap \exists N_1, A \cap \exists N_2, A). \]

Intuitively, the membership to the extension of A or \( \neg A \) corresponds to the truth values true and false respectively and checking if \( \Sigma_F \models Q(f) \) corresponds to checking if in every truth assignment for \( F \) there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e. a clause that is not satisfied.

**Lemma 4.2.7** A \( 2+2\)-CNF formula \( F \) is unsatisfiable if and only if \( \Sigma_F \models Q(f) \).

**Proof.**

"⇒" Suppose \( F \) is unsatisfiable. Notice first that \( \Sigma_F \) is always satisfiable independently of \( F \). Consider a model \( I \) of \( \Sigma_F \) (which always exists), and let \( \delta_I \) be the truth assignment for \( F \) such that \( \delta_I(l) = true \) if and only if \( l^I \notin A^I \), for every letter \( l \). Since \( F \) is unsatisfiable, there exists a clause \( C \) that is not satisfied by \( \delta_I \). It follows that the individuals related to \( c_i \) through the roles \( P_1, P_2 \) are in the extension of \( (\neg A)^I \) and the individuals related to \( c_i \) through the roles \( N_1, N_2 \) are in the extension of \( A^I \). Thus \( c_i^I \in (\exists P_1, \neg A \cap \exists P_2, \neg A \cap \exists N_1, A \cap \exists N_2, A)^I \), and consequently \( f^I \notin Q^I \). Therefore, since this argument holds for every model \( I \) of \( \Sigma_F \), we can conclude that \( \Sigma_F \models Q(f) \).

"⇐" Suppose \( F \) is satisfiable, and let \( \delta \) be a truth assignment satisfying \( F \). Let \( I_\delta \) be the interpretation for \( \Sigma_F \) defined as follows:

- \( A^I_\delta = \{ l^I_\delta \mid \delta(l) = true \} \),
- \( \rho^I_\delta = \{(a^I_\delta, b^I_\delta) \mid \rho(a, b) \in \Sigma_F \} \) for \( \rho = Cl, P_1, P_2, N_1, N_2 \).

By definition of \( \rho^I_\delta \) (for \( \rho = Cl, P_1, P_2, N_1, N_2 \) we see that \( I_\delta \) is a model of \( \Sigma_F \). On the other hand, since \( F \) is satisfiable, for every clause in \( F \) there exists either a positive literal interpreted as true or a negative one interpreted as false. It follows that, for every individual \( c_i \), there exists either a role \( (P_1 \) or \( P_2 \) such that the object related to \( c_i \) by means of that role is in the extension of \( A \) or there exists a role \( (N_1 \) or \( N_2 \) such that the object related to \( c_i \) by means of that role is in the extension of \( \neg A \); hence \( f^I_\delta \notin Q^I_\delta \). Therefore, we can conclude that \( \Sigma_F \models Q(f) \). \( \Box \)
**Theorem 4.2.8** Instance Checking in $\mathcal{ALE}$ is coNP-hard in the size of the knowledge base.

**Proof.** The claim follows from Theorem 4.2.6 and Lemma 4.2.7 and from the fact that, given a 2+2-CNF formula $F$, $Q$ is fixed independently of $F$ and $\Sigma_F$ can be computed in polynomial time with respect to the size of $F$. 

Since the language $\mathcal{ALE}$ involves name negation and the above reduction makes use of them it may seem that the coNP-hardness arises from the interaction of qualified existential quantification with the name negation. On the contrary, we are able to show that the coNP-hardness is caused by the qualified existential quantification alone. In fact, if we consider the language $\mathcal{FL}^- \ (\mathcal{FL}^- +$ qualified existential quantification), we are able to prove that Instance Checking in $\mathcal{FL}^-$ is coNP-hard too. The intuition is that in $\mathcal{FL}^-$ it is possible to require a reasoning by case analysis too. In particular, this is done by considering two $\mathcal{FL}^-$-concepts of the form $\exists R$ and $\forall R.C$ (instead of $A$ and $\neg A$ in $\mathcal{ALE}$), and exploiting the fact that their interpretation covers the entire domain, i.e. $\exists R \cup \forall R.C \equiv \top$. More in detail, the same reduction used to prove the coNP-hardness in $\mathcal{ALE}$, can be used for $\mathcal{FL}^-$ by replacing $Q$ with the $\mathcal{FL}^-$-concept $Q'$ obtained substituting any occurrence of $A$ with $\exists R$ and each occurrence of $\neg A$ with $\forall R.C$. The proof that any 2+2-CNF formula $F$ is unsatisfiable if and only if $\Sigma_F \models Q'(f)$ can be obtained following the same line of the proof of Lemma 4.2.7.

The new source of complexity is related to the use of qualified existential quantification in the query concept. Our result can be explained as follows: when answering queries formulated by concepts with such a construct, one has to consider that the properties of the individuals in the knowledge base strongly depend upon the interaction with other individuals. Such an interaction cannot be ignored when answering queries with qualified existential quantification, whereas it plays a limited role when answering queries with restricted existential quantification (i.e. $\exists R$). This explains why Instance Checking in $\mathcal{AL}$ and $\mathcal{ALN}$ do not suffer from this computational problem. Let us illustrate the new form of complexity by means of another example. The example we present is taken from [Lev88, Sec. 10] and shows how certain logic puzzles can be expressed through an Instance Checking in $\mathcal{ALE}$.

**Example 4.2.9** There are five blocks in a stack, where the second from the top is green, and the fourth is not green. Such situation could be represented by the following knowledge base in $\mathcal{ALE}$ (where $ob$ represents the observer watching the stack):

$$\Sigma = \{\text{ON}(a,b), \text{ON}(b,c), \text{ON}(c,d), \text{ON}(d,e)\},$$
\[\text{Green}(b), \neg\text{Green}(d),\]
\[\text{SEES}(ob,a), \text{SEES}(ob,b), \text{SEES}(ob,c), \text{SEES}(ob,d), \text{SEES}(ob,e)\]

Consider the query: \(\exists \text{SEES}. (\text{Green} \sqcap \exists \text{ON}. \neg\text{Green})(ob)\).

Asking whether \(\Sigma \models \exists \text{SEES}. (\text{Green} \sqcap \exists \text{ON}. \neg\text{Green})(ob)\) corresponds to ask if the observer sees a green block directly on top of a non-green one. As pointed out in [Lev88], the answer is not immediately obvious. If one thinks about it, though, one realizes that the answer is YES.

In order to discover this, one needs to reason by case analysis. In fact, the problem asks if in every model \(\mathcal{M}\) of \(\Sigma\) there is an individual, say \(z\), such that \(\text{SEES}(ob,z), \text{Green}(z)\) and \(\exists \text{ON}. \neg\text{Green}(z)\) are true in \(\mathcal{M}\). Obviously, in every model \(\mathcal{M}\) of \(\Sigma\), either \(\text{Green}(c)\) or \(\neg\text{Green}(c)\) is true. In the first case, it is easy to see that \(z\) is simply \(c\) (and the non-green object it is on is \(d\)), while in the second case \(z\) is \(b\) (and the non-green object it is on is \(c\)).

Therefore, even if none of the individuals related to the individual \(ob\) through the role \(\text{SEES}\) is in the extension of \((\text{Green} \sqcap \exists \text{ON}. \neg\text{Green})\) in every model of the knowledge base, it happens that the combination of the assertions on the individuals \(b\) and \(c\) in the knowledge base implies that in every model at least one of them is in that extension. \(\square\)

In the following we give an upper bound to the knowledge base complexity of Instance Checking in \(\text{ALE}\). In particular, we prove that it is in the class \(\Pi^P_1\). Remind that the class \(\Pi^P_1\), also denoted by \(\text{coNP}^P\), consists of the problems whose complement can be solved by a nondeterministic polynomial time algorithm exploiting a nondeterministic polynomial oracle.

**Lemma 4.2.10** Let \(\Sigma\) be a consistent knowledge base in \(\text{ALE}\), \(a\) be an individual, and \(D\) be a \(\text{ALE}\)-concept. Then checking if \(\Sigma \not\models D(a)\) can be done by a non-deterministic algorithm, which runs in polynomial time with respect to \(|\Sigma|\) and exploits a nondeterministic polynomial time oracle.

**Proof.** We know that \(\Sigma \models D(a)\) if and only if the knowledge base \(\Sigma \cup \{\neg D(a)\}\) is inconsistent. In general, \(\neg D\) is not an \(\text{ALE}\)-concept, since it contains the negation of a general concept. Therefore, \(\Sigma \cup \{\neg D(a)\}\) is not a knowledge base in \(\text{ALE}\), but a knowledge base in \(\text{ALC}\). In Section 4.1 an algorithm is presented for checking the consistency of a knowledge base in \(\text{ALC}\). That algorithm consists of a \(\text{NPTIME}\) procedure to compute a pre-completion, which exploits a \(\text{PSPACE}\) subprocedure to compute Concept Satisfiability. However in our case only the assertion \(\neg D(a)\) may contain negations of general concepts and we are interested in the complexity with respect to \(\Sigma\) and not with respect to \(D\), i.e. the size of \(D\) can be considered fixed. For this reason, any time we use
the subprocedure it is called to check the satisfiability of a concept such that
its part containing general concepts has fixed size. It follows, according to the
algorithm given in Section 4.1, that it can be computed in polynomial time
by an alternating Turing Machine (TM) such that the alternation between ∧-

nodes and ∨-nodes exists only down to a fixed depth and below that level only

∨-nodes are present. An alternating TM respecting this limitation has a fixed
number of ∧-nodes and therefore can be simulated by a non-deterministic TM
which works in polynomial time too.

\[ \text{Theorem 4.2.11 Instance Checking in } \mathcal{AL}E \text{ is in } \Pi_2^p \text{ with respect to the}

knowledge base complexity} \]

\[ \text{Proof. Easily follows from Lemma 4.2.10} \]

Notice that the complexity of Instance Checking with respect to the knowl-

dge base complexity is not completely placed in the complexity hierarchy. We

conjecture that it is \( \Pi_2^p \)-complete.

\[ \text{Combined complexity of Instance Checking in } \mathcal{AL}E \]

In this section we prove that Instance Checking in \( \mathcal{AL}E \) is PSPACE-complete

with respect to combined complexity, and, therefore, is even more complex

compared to knowledge base complexity.

We first focus on proving that the problem is PSPACE-hard. The proof

is based on a polynomial reduction from the PSPACE-complete problem of
deciding the validity of Quantified Boolean Formulae (QBF for short), which

is briefly recalled here.

A \textit{literal} is a nonzero integer. A \textit{clause} is a nonempty finite set \( c \) of literals

such that \( l \in c \) implies \( -l \notin c \). A \textit{prefix} from \( m \) to \( n \), where \( m \) and \( n \) are

positive integers such that \( m \leq n \), is a sequence

\[ (Q_m^m) (Q_{m+1}^m + 1) \cdots (Q_n^n), \]

where each \( Q_i \) is either \( \forall \) or \( \exists \). A \textit{quantified boolean formula} is a pair

\( P, M \), where, for some \( n \), \( P \) is a prefix from \( 1 \) to \( n \) and \( M \) is a finite nonempty

set of clauses containing only literals between \( -n \) and \( n \).

Let \( P \) be a prefix from \( m \) to \( n \). A \( P \)-assignment is a mapping

\[ \{m, m + 1, \ldots, n\} \rightarrow \{t, f\}. \]

An assignment \( \alpha \) \textit{satisfies} a literal \( l \) if \( \alpha(l) = t \) if \( l \) is positive and \( \alpha(-l) = f \) if

\( l \) is negative. An assignment \textit{satisfies} a clause if it satisfies at least one literal

of the clause.

Let \( P \) be a prefix from \( m \) to \( n \). A set \( A \) of \( P \)-assignments is \textit{canonical} for

\( P \) if it satisfies the following conditions:
1. \( A \) is nonempty

2. if \( P = (\exists m)P' \), then all assignments of \( A \) agree on \( m \) and, if \( P' \) is nonempty, \( \{ \alpha_{|m+1,..,n} | \alpha \in A \} \) is canonical for \( P' \)

3. if \( P = (\forall m)P' \), then
   
   (a) \( A \) contains an assignment that satisfies \( m \) and, if \( P' \) is nonempty,  
   \( \{ \alpha_{|m+1,..,n} | \alpha \in A \text{ and } \alpha(m) = t \} \) is canonical for \( P' \)
   
   (b) \( A \) contains an assignment that satisfies \( -m \) and, if \( P' \) is nonempty,  
   \( \{ \alpha_{|m+1,..,n} | \alpha \in A \text{ and } \alpha(m) = f \} \) is canonical for \( P' \).

A quantified boolean formula \( P,M \) is valid if there exists a set \( A \) of \( P \)-assignments that is canonical for \( P \) such that every assignment in \( A \) satisfies every clause of \( M \). It is well known that deciding the validity of quantified boolean formulae is a PSPACE-complete problem [GJ79, page 172].

In [SS91] it is proven that QBF can be reduced to the satisfiability of an \( \mathcal{ALC} \)-concept of the form

\[
C \cap C^1_0 \cap \cdots \cap C^n_0
\]

where \( C \) and \( C_0^i \) are defined as follows. The concept \( C \) is obtained from the prefix \( P \) using the equations

\[
C = [P] = \left\{ \begin{array}{ll}
(\exists R.A) \cap (\exists R.\neg A) \cap (\forall R,[P']) & \text{if } P = \forall m.P' \\
((\exists R.A) \cup (\exists R.\neg A)) \cap (\forall R,[P']) & \text{if } P = \exists m.P'
\end{array} \right.
\]

where \( P' \) is a non-empty prefix, and

\[
C = [P] = \left\{ \begin{array}{ll}
(\exists R.A) \cap (\exists R.\neg A) & \text{if } P = \forall m \\
(\exists R.A) \cup (\exists R.\neg A) & \text{if } P = \exists m.
\end{array} \right.
\]

Observe that exploiting the equivalence

\[
\exists R.A \cup \exists R.\neg A \equiv \exists R
\]

the concept \( C \) can be rewritten in an equivalent \( \mathcal{ALC} \)-concept \( D \).

The concept \( C_0^i \) is obtained from the clause \( c_i \) as follows. Let

\[
k = \max_{l \in N_i} |l|.
\]

Then, for \( l \in 1..(k - 1) \), define

\[
C_{i-1}^l = \left\{ \begin{array}{ll}
A \cup \forall R.C_i^l & \text{if } l \in c_i \\
\neg A \cup \forall R.C_i^l & \text{if } -l \in c_i \\
\forall R.C_i^l & \text{if neither } l \text{ nor } -l \text{ is in } c_i,
\end{array} \right.
\]
and define
\[ C^i_{k-1} = \begin{cases} \forall R.A & \text{if } k \in c_i \\ \forall R.\neg A & \text{if } -k \in c_i. \end{cases} \]

Notice that each \( C^i \) does not contain the symbol "\( \land \)" and therefore, for each \( i \), the negation of \( C^i \) can be rewritten into an equivalent \( \mathcal{ALC} \)-concept, denoted by \( D_i \). Therefore the concept \( C \cap C^1 \cap \cdots \cap C^n \) is equivalent to the \( \mathcal{ALC} \)-concept \( D \cap \neg D_1 \cap \cdots \cap \neg D_n \).

The problem of checking the satisfiability of \( D \cap \neg D_1 \cap \cdots \cap \neg D_n \) is now reduced to the problem of checking if \( \Sigma \models E_0(a) \), where \( a \) is an individual, \( \Sigma \) is a knowledge base in \( \mathcal{ALC} \), and \( E_0 \) is an \( \mathcal{ALC} \)-concept. \( \Sigma \) and \( E_0 \) are computed from \( D \) and \( D_i \) (for \( i = 1, 2, \ldots, n \)), and their sizes are polynomially related to those of \( D \) and the various \( D_i \).

In the following we assume \( Q \) to be a new role symbol, and \( E_n \) to be a new atomic concept. \( E_0 \) is defined by the following equations:
\[ E_i = \exists Q_i(D_{i+1} \cap E_{i+1}) \quad i = 0, 1, \ldots, n - 1. \]
The knowledge base \( \Sigma \) is defined as follows
\[ \Sigma = \{ Q(a, a), D_1(a), D_2(a), \ldots, D_n(a), \\
Q(a, b), E_1(b), E_2(b), \ldots, E_n(b), D(b) \}. \]

It can be verified that \( \Sigma \) and \( E_0 \) contain only \( \mathcal{ALC} \)-concepts. Therefore, \( \Sigma \) is a knowledge base in \( \mathcal{ALC} \), and \( E_0 \) is an \( \mathcal{ALC} \)-concept. It is also easy to see that both \( |\Sigma| \) and \( |E_0| \) is polynomially bounded by \( |D \cap \neg D_1 \cap \cdots \cap \neg D_n| \).

Our aim is to show that \( \Sigma \models E_0(a) \) if and only if \( D \cap \neg D_1 \cap \cdots \cap \neg D_n \) is satisfiable. To this end, we define the following constraint system \( T \), which will be needed in the sequel:
\[ T = \{ a: \neg E_1, \ldots, a: \neg E_n \} \cup \{ b: \neg D_1, b: \neg D_2, \ldots, b: \neg D_n \} \cup S_\Sigma \cup \{ a: \neg E_0 \}. \]

**Lemma 4.2.12** Let \( S_\Sigma \cup \{ a: \neg E_0 \} \) and \( T \) be defined as stated above. Then \( S_\Sigma \cup \{ a: \neg E_0 \} \) has at least one clash-free completion if and only if \( T \) has one.

**Proof.** Let \( S_0 = S_\Sigma \cup \{ a: \neg E_0 \} \). Define the following sequence of constraint systems
\[ S_i = S_{i-1} \cup \{ a: \neg E_i, b: \neg D_i \} \quad i = 1, \ldots, n \]
Observe that \( S_n = T \). We first prove that \( S_0 \) has at least one clash-free completion if and only if \( S_1 \) has one. Let us apply the completion rules to \( S_0 = S_\Sigma \cup \{ a: \neg E_0 \} \). Since \( aQa, aQb \) and \( a: \neg E_0 \) are in \( S_0 \), and since \( \neg E_0 \) can be rewritten as \( \forall Q_a(\neg D_1 \cup \neg E_1) \), the application of the \( \neg \)-rule adds to \( S_0 \) the constraints \( a: (\neg D_1 \cup \neg E_1) \) and \( b: (\neg D_1 \cup \neg E_1) \). Now, applying the \( \to \)-rule
to the first constraint, two different constraint system can be obtained, one containing \( a : \neg D_1 \), and the other one containing \( a : \neg E_1 \). Since \( a : D_1 \) is in \( S_0 \), every completion of the first system contains a clash. Hence we look at the second one, and apply the \(-\rightarrow\_l\) rule to the constraint \( b : (\neg D_1 \sqcap \neg E_1) \). Again, we obtain two constraint systems, one containing \( b : \neg D_1 \) and the other one containing \( b : \neg E_1 \). This time every completion of the second one contains a clash, because the constraint \( b : E_1 \) is in \( S_0 \). This shows that \( S_0 \) has at least one clash-free completion if and only if \( S_1 = S_0 \cup \{ a : \neg E_1, b : \neg D_1 \} \) has one. Following the same reasoning lines, one can easily prove that for each \( i \in 1..n \), \( S_i \), \( \neg \) has at least one clash-free completion if and only if \( S_i \) has one. Therefore, we can conclude that \( S_0 \) has at least one clash-free completion if and only if \( T \) has one.

Notice that \( T \) is a pre-completion of \( S_\Sigma \cup \{ a : \neg E_0 \} \), since the \(-\rightarrow\_l, \rightarrow\_v\), and \(-\rightarrow\_l\) rules cannot be applied to \( T \). Thus, Proposition 4.1.2 ensures that \( T \) has at least one clash-free completion if and only if the two concepts \( CC_{T\alpha} \), \( CC_{T\beta} \) are satisfiable (\( a \) and \( b \) are the only individuals in \( T \)), where

\[
CC_{T\alpha} = D_1 \sqcap D_2 \sqcap \cdots \sqcap D_n \sqcap \neg E_0 \sqcap \cdots \sqcap \neg E_n
\]

\[
CC_{T\beta} = D \sqcap \neg D_1 \sqcap \cdots \sqcap \neg D_n \sqcap E_0 \sqcap \cdots \sqcap E_n.
\]

Notice also that \( CC_{T\alpha} \) is clearly satisfiable, because all existential quantifications (in the \( D_i \)'s concepts) involve role \( R \), while all universal quantifications (in the \( \neg E_i \)'s concepts) involve role \( Q \). On the other hand, in \( CC_{T\beta} \), \( E_0 \sqcap \cdots \sqcap E_n \) is obviously satisfiable and it involves only the role \( Q \), while \( D \sqcap \neg D_1 \sqcap \neg D_2 \sqcap \cdots \sqcap \neg D_n \) involves only the role \( R \). Therefore, we can conclude that \( CC_{T\beta} \) is satisfiable if and only if \( D \sqcap \neg D_1 \sqcap \neg D_2 \sqcap \cdots \sqcap \neg D_n \) is satisfiable. From all the above observations, we can derive the following proposition.

**Lemma 4.2.13** Let \( C \sqcap C^1_0 \sqcap \cdots \sqcap C^n_0 \), \( \Sigma \), and \( E_0 \) be defined as stated above. Then \( C \sqcap C^1_0 \sqcap \cdots \sqcap C^n_0 \) is unsatisfiable if and only if \( \Sigma \models E_0(a) \).

**Proof.** We know that \( \Sigma \models E_0(a) \) if and only if every completion of \( S = S_\Sigma \cup \{ a : \neg E_0 \} \) contains a clash. Furthermore, from Lemma 4.2.12, every completion of \( S \) contains a clash if and only if every completion of \( T \) contains a clash. From the above observations, we know that every completion of \( T \) contains a clash if and only if \( D \sqcap \neg D_1 \sqcap \neg D_2 \sqcap \cdots \sqcap \neg D_n \) is unsatisfiable. Since \( D \) is equivalent to \( C \) and \( \neg D_i \) is equivalent to \( C^i_0 \) (for each \( i = 1, 2, \ldots, n \)), it follows that \( D \sqcap \neg D_1 \sqcap \neg D_2 \sqcap \cdots \sqcap \neg D_n \) is unsatisfiable if and only if \( C \sqcap C^1_0 \sqcap \cdots \sqcap C^n_0 \) is unsatisfiable.

**Theorem 4.2.14** Instance Checking in \( \text{ACL} \) is \( \text{PSPACE}\)-complete with respect to combined complexity.
4.2 Complexity of Instance Checking

Proof. We already observed that both \(|\Sigma|\) and \(|E_0|\) are polynomially bounded by the size of \(C \cap C_0^1 \cap \cdots \cap C_0^n\). Proposition 4.2.13, together with the reduction of QBF to the satisfiability of \(C \cap C_0^1 \cap \cdots \cap C_0^n\) given in [SS91], ensures us that Instance Checking in \(\mathcal{ALE}\) is PSPACE-hard. Since Instance Checking in \(\mathcal{ALE}\) is a special case of the PSPACE-complete problem Instance Checking in \(\mathcal{ACL}\) (see Subsection 5.4), it follows that Instance Checking in \(\mathcal{ALE}\) is also in PSPACE. Hence Instance Checking in \(\mathcal{ALE}\) is PSPACE-complete. □

Instance Checking in \(\mathcal{ACR}\)

We now show that the source of complexity singled out in the previous subsection for \(\mathcal{ALE}\) does not arise for the other NP-complete language \(\mathcal{ACR}\). Notice that the NP-hardness of Subsumption in \(\mathcal{ACR}\), and therefore the NP-hardness of Instance Checking in \(\mathcal{ACR}\), follows from NP-hardness of Subsumption in \(\mathcal{ALE}\), for the reason that, when checking Concept Satisfiability, qualified existential quantification can be simulated in \(\mathcal{ACR}\) using role conjunction (see [DLN91a]). In particular, checking whether the \(\mathcal{ALE}\)-concept \(C\) is satisfiable can be reduced to the problem of checking whether \(C'\) is satisfiable, where \(C'\) is the \(\mathcal{ACR}\)-concept obtained from \(C\) by substituting every subconcept \(E\) of the form \(\exists R . D\) with \((\exists (R \cap Q)) \cap (\forall Q . D)\), where \(Q\) is a new role name used only in \(E\). Since \(|C'|\) is polynomially bounded by \(|C|\), this means that, as far as satisfiability is concerned, qualified existential quantification can be expressed in \(\mathcal{ACR}\), even if it not explicitly present among the language constructors.

The above observations allow us to derive the NP-hardness of Instance Checking in \(\mathcal{ACR}\) w.r.t. knowledge base complexity (and hence, combined complexity too). However, they do not imply that the source of complexity discussed in Subsections 5.2.1 and 5.2.2 plays a role in Instance Checking in \(\mathcal{ACR}\). In fact, such a source of complexity arises from the use of qualified existential quantification in the query concept. Thus, the question we have to address is whether \(\mathcal{ACR}\) allows one to express such a constructor in the query concepts.

We answer negatively to the above question by showing that Instance Checking in \(\mathcal{ACR}\) can be done in \(\mathcal{NP}\) with respect to combined complexity. The fact that qualified existential quantification cannot be expressed in an \(\mathcal{ACR}\) query, is confirmed in Chapter 6, where it is shown that Instance Checking using \(\mathcal{ACR}\) as query language to a knowledge base in \(\mathcal{AC}\) can be done in polynomial time (whereas the same problem is \(\mathcal{coNP}\)-hard when the query language is \(\mathcal{ALE}\)).

Notice that only one pre-completion can be obtained from a knowledge base \(\Sigma\) in \(\mathcal{ACR}\). The next proposition shows that, as for \(\mathcal{AC}\), checking whether \(\Sigma\) logically implies \(D(a)\) can be reduced to checking whether \(D\) subsumes \(C_{\Sigma[a]}\). Its proof is similar to that of Proposition 4.2.1.
Proposition 4.2.15 Let $\Sigma$ be a consistent knowledge base in $\mathcal{ACR}$, a an individual, and $D$ an $\mathcal{ACR}$-concept. Then $\Sigma \models D(a)$ if and only if $D$ subsumes $C_{\Sigma[a]}$.

An NP algorithm for checking the inconsistency of $S_\Sigma \cup \{a : \neg D\}$ can be defined as follows. Notice first of all that checking the inconsistency of $\Sigma$ can be done in nondeterministic polynomial time (see Section 4). So, let us assume that $\Sigma$ is consistent. The algorithm computes in polynomial time the pre-completion $S$ of $S_\Sigma$, and then checks whether $D$ subsumes $C_{\Sigma[a]}$. Such a check can be done in nondeterministic polynomial time with respect to $|D|$ and $|C_{\Sigma[a]}|$, and therefore, with respect to $|D|$ and $|\Sigma|$. It follows that we can encode the whole process into a nondeterministic Turing machine whose time complexity is polynomial with respect to $|D|$ and $|\Sigma|$. This shows that Instance Checking in $\mathcal{ACR}$ is in NP with respect to combined complexity (and therefore with respect to knowledge base complexity, too).

Theorem 4.2.16 Instance Checking in $\mathcal{ACR}$ is NP-complete with respect to both knowledge base complexity and combined complexity.

4.2.3 Languages with coNP-complete Subsumption ($\mathcal{ACL}$)

As mentioned in Chapter 2, the presence of constructors expressing a form of disjunction in general makes Subsumption coNP-hard. Moreover, in Section 4.1 we have shown that checking the consistency of a knowledge base expressed in a language with disjunction is NP-hard. It follows that Instance Checking is coNP-hard too.

In this section we address the question of whether the new source of complexity singled out in Subsection 4.2.2 makes the problem harder. We do so by considering the language $\mathcal{ACL}$, among whose constructors there is not the qualified existential quantification, and for which Subsumption is coNP-complete.

The result of our analysis is that Instance Checking in $\mathcal{ACL}$ is in the same complexity class as Subsumption in $\mathcal{ACL}$, namely it is coNP-complete. Therefore, the presence of assertions on individuals does not add any complexity to the reasoning in $\mathcal{ACL}$. In order to show this, let us first consider the following proposition.

Proposition 4.2.17 Let $\Sigma$ be an knowledge base in $\mathcal{ACL}$, a an individual, and $D$ an $\mathcal{ACL}$-concept. Then $\Sigma \models D(a)$ if and only if for every pre-completion $S$ of $S_\Sigma$, either there is no clash-free completion of $S$, or $D$ subsumes $C_{S[a]}$.

Proof,
"⇒" Assume that there is a pre-completion $S$ of $S_{\Sigma}$ such that there exists a clash-free completion $S'$ of $S$, and also $D$ does not subsume $C_{S_{\Sigma}}$. The last hypothesis implies that there is a clash-free completion $S''$ of $\{a : C_{S_{\Sigma}} \cap \neg D\}$. Consider now the constraint system $S' \cup S''$. It can be easily verified that such a constraint system contains a clash-free completion of $S_{\Sigma} \cup \{a : \neg D\}$.

"⇐" Assume that for every pre-completion $S$ of $S_{\Sigma}$, either there is no clash-free completion of $S$, or $D$ subsumes $C_{S_{\Sigma}}$. Suppose that $\Sigma \not\models D(a)$, implying that there is a clash-free completion $S'$ of $S_{\Sigma} \cup \{a : \neg D\}$. Note that $S'$ contains both a clash-free pre-completion $S$ of $S_{\Sigma}$, and a clash-free completion of $\{a : CC_{S_{\Sigma}}^1, \ldots, a : CC_{S_{\Sigma}}^n, a : \neg D\}$, where $CC_{S_{\Sigma}}^1, \ldots, CC_{S_{\Sigma}}^n$ denote all the conjuncts of $CC_{S_{\Sigma}}$. It can be easily verified that the existence of a clash-free completion of $\{a : CC_{S_{\Sigma}}^1, \ldots, a : CC_{S_{\Sigma}}^n, a : \neg D\}$ implies the existence of a clash-free completion of $\{a : CC_{S_{\Sigma}}^1 \cap RC_{S_{\Sigma}}^1 \cap \neg D\}$. Hence, $D$ does not subsume $CC_{S_{\Sigma}} \cap RC_{S_{\Sigma}}$.

From the above proposition, we can derive a nondeterministic polynomial time algorithm for checking whether $\Sigma \not\models D(a)$ (the complement of Instance Checking in $ALU$) as follows. The algorithm first nondeterministically computes a pre-completion $S$ of $\Sigma$, and then, for each individual $b$, if $b \neq a$, computes a completion of $CC_{S_{\Sigma}} b$, else computes a completion of $CC_{S_{\Sigma}} \cap \neg D$. The answer will be TRUE if none of the computed completions contains a clash. Since the size of the pre-completion $S$ of $\Sigma$, the size of $CC_{S_{\Sigma}} b$ and the size of $CC_{S_{\Sigma}} \cap \neg D$ are polynomially bounded by $|\Sigma|$ and $|D|$, it follows that Instance Checking in $ALU$ is in coNP with respect to combined complexity (and therefore to knowledge base complexity, too).

### 4.2.4 Languages with PSPACE-complete Subsumption ($ALC$)

In this section we deal with the language $ALC$, where, by means of the negation of concepts, both qualified existential quantification and disjunction can be expressed. Therefore both sources of complexity discussed in Section 3 are present in this language. As shown in Chapter 3, Concept Satisfiability and Subsumption in $ALC$ are PSPACE-complete problems.

Differently from the languages considered so far, in $ALC$ Instance Checking easily reduces to Consistency. In fact, since $ALC$ admits general negations, $\Sigma \models D(a)$ if and only if the knowledge base $\Sigma \cup \{\neg D(a)\}$ is inconsistent. This simple consideration implies that the Instance Checking problem in $ALC$ has PSPACE combined complexity. It follows that for this language, Instance Checking is in the same complexity class as Subsumption.

Even though in $ALC$ Instance Checking and Subsumption are in the same complexity class, Instance Checking is not directly reducible to Subsumption as for $ALU$, i.e. Proposition 4.2.17 does not hold for $ALC$. This is easily derivable from the fact that $ALC$ is a sublanguage of $ALC$. The reason why
<table>
<thead>
<tr>
<th></th>
<th>Concept satisfiability</th>
<th>Subsumption</th>
<th>Consistency</th>
<th>inst. checking KB compl.</th>
<th>inst. checking comb. compl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{ALC}$</td>
<td>$P$ [SS91]</td>
<td>$P$ [DLNN91a]</td>
<td>$P$</td>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>$\mathcal{ALCV}$</td>
<td>$P$ [DLNN91a]</td>
<td>$P$ [DLNN91a]</td>
<td>$P$</td>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>$\mathcal{ALE}$</td>
<td>coNP [DHL+92]</td>
<td>NP [DHL+92]</td>
<td>coNP</td>
<td>NP-hard in $\Pi_2^p$</td>
<td>PSPACE</td>
</tr>
<tr>
<td>$\mathcal{ACR}$</td>
<td>coNP [DLNN91a]</td>
<td>NP [DLNN91a]</td>
<td>coNP</td>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>$\mathcal{ACU}$</td>
<td>NP [DLNN91a]</td>
<td>coNP [DLNN91a]</td>
<td>NP</td>
<td>coNP</td>
<td>coNP</td>
</tr>
<tr>
<td>$\mathcal{ALC}$</td>
<td>PSPACE [SS91]</td>
<td>PSPACE [SS91]</td>
<td>PSPACE [BH91b]</td>
<td>PSPACE [BH91b]</td>
<td>PSPACE [BH91b]</td>
</tr>
</tbody>
</table>

Table 4.1: Complexity of reasoning in pure languages

the new source of complexity singled out in Subsection 4.2.2 does not affect the complexity class of reasoning on knowledge bases in $\mathcal{ALC}$, is different from the case of $\mathcal{ACU}$, and is based on the fact that Concept Satisfiability in $\mathcal{ALC}$ already suffers from a source of complexity which is of the same nature as the new one.

### 4.3 Discussion

In figure 4.1 we summarize the results of our analysis together with previous known results. For each of the languages considered in this chapter, the table shows the complexity of Concept Satisfiability, Subsumption, Consistency, and Instance Checking, the last one with respect to both knowledge base complexity and combined complexity.

Although not exhaustive with respect to all possible languages, our analysis singles out several interesting properties.

First, there are languages (e.g. $\mathcal{ALE}$) for which knowledge base complexity and combined complexity of Instance Checking belong to different complexity classes. This fact confirms that, in order to have a significant complexity measure of the performance of the system, we must carefully consider which input data have critical size for the deductive services.

Most importantly, while Consistency is in the same complexity class as
4.3 Discussion

Concept Satisfiability for all the considered languages, different results are obtained for instance checking depending upon the language used to express the knowledge base. In particular:

1. Languages with polynomial Subsumption, such as $\mathcal{AL}$ and $\mathcal{ALN}$, preserve their tractability, although, in general, some additional work with respect to Subsumption is required.

2. Languages with qualified existential quantification, such as $\mathcal{AL\mathcal{E}}$, suffer from an additional source of complexity which does not show up when checking Subsumption. This new source of complexity is related to the use of qualified existential quantification in the concept representing the query, which allows for a form of navigation through the assertions in the knowledge base.

3. For other languages where Subsumption is NP-complete, such as $\mathcal{ALC}$, the source of complexity singled out for $\mathcal{AL\mathcal{E}}$ does not arise. In fact, although $\mathcal{ALC}$ can simulate qualified existential quantification in the assertions, qualified existential quantification cannot be expressed in the query concept, neither explicitly nor implicitly. It follows that Instance Checking in $\mathcal{ALC}$ is NP-complete, exactly like Subsumption.

4. With regard to other languages, Instance Checking is again in the same complexity class as Subsumption. This happens either because no navigation through the knowledge base is allowed (e.g. $\mathcal{ALU}$), or because the language is rich enough (e.g. $\mathcal{ALC}$), so that Subsumption is already computational demanding.
Chapter 5

Reasoning with Mixed Concept Languages

With the term mixed languages we call those languages including at least one constructor that allows one to refer to the individuals into a concept expression. The constructors of that kind that we consider are $\mathcal{O}$ and $\mathcal{B}$.

Mixed languages have received very little attention from the theoretical point of view, although they are generally considered useful for practical systems (see e.g. [BMP+91]).

The demand for the constructors $\mathcal{O}$ and $\mathcal{B}$ in concept-based systems is due to the significant increase of the expressiveness of the language they provide, as shown in Section 5.1. It is also confirmed by the fact that they are both included in the recent proposal for a standard concept-based system in [PS93] ($\mathcal{O}$ was also included in the previous proposal [BBH+91]).

In this chapter, we investigate on the consequences of introducing $\mathcal{O}$ and $\mathcal{B}$ in the concept language, and in general of admitting the individuals in it. In particular, we give a survey of the various issues associated with the use of $\mathcal{O}$ and $\mathcal{B}$ (Section 5.1). We briefly describe some of the strategies chosen by the implementors of the actual systems in order to deal with $\mathcal{O}$ and $\mathcal{B}$ (Section 5.2). We show the relationship between the various reasoning services in mixed languages (Section 5.3). We present several complexity results (Section 5.4) and, finally, in Section 5.5 we draw some conclusions.

The results of this chapter are all included in [Sch94]. Lemma 5.4.12 has been originally stated in [LS91a].

5.1 Peculiarities of Mixed Languages

In this section we investigate the effects of the use of mixed languages in the reasoning process. First, we provide an example of knowledge base built using
a mixed language. Then, in Sections 5.1.1–5.1.6, we give an overview of the most relevant issues related to $\mathcal{O}$ and we give an intuition of how these issues can make the reasoning process more complex than in the corresponding pure language without $\mathcal{O}$ (the so-called underlying language). In Section 5.1.7, we consider the issues related to $\mathcal{B}$.

**Example 5.1.1** Let $\Sigma$ be the following knowledge base in $\mathcal{ALEO}$:

$$\Sigma = \{ \exists \text{FRIEND.}\{\text{susan, peter}\}(\text{john}),$$
$$\quad \forall \text{FRIEND.\Married(\text{john})},$$
$$\quad \neg \text{Married(\text{peter})} \}$$

It is easy to see that $\Sigma$ is consistent. Moreover, some non-trivial conclusion can be drawn from $\Sigma$. For example, we can prove that $\Sigma \models \text{Married(\text{susan})}$ and $\Sigma \models \text{FRIEND(\text{john, susan})}$. In fact, due to the last two assertions, Peter cannot be a friend of John. Therefore, according to the first assertion, the friend of John must be Susan and, consequently, she must be married, i.e. both $\text{FRIEND(\text{john, susan})}$ and $\text{Married(\text{susan})}$ are logically implied by $\Sigma$.$\square$

5.1.1 Implicit Assertions

One of the characteristics of concept languages is the ability of describing incomplete knowledge. In particular, by means of existential quantification, it is possible to express information about objects that exist but whose identity is not known by the knowledge base. With regards to these unknown objects, it is also possible to state their membership to some concept. In particular, when $\mathcal{O}$ is used, it is possible to state the membership of an unknown object to a set of individuals. A consequence of this is that the unknown object is forced to be one of the individuals of the set.

For example, consider the following assertion in $\mathcal{ALEO}$:

$$\exists R. (A \cap \{a\})(d).$$

It explicitly states the membership of $d$ in $\exists R. (A \cap \{a\})$, but it also implicitly states that $a$ must be in the extension of $A$. In fact, it says that there exists an object in $A \cap \{a\}$, therefore this object must be $a$ and it must be in the extension of $A$; that is equivalent to stating the assertion $A(a)$.

In the above example we have considered a collection formed by a single individual. If we consider collection formed by more the one element then the resulting implicit assertion can be disjunctive. For example, if we state the existence of an object in the concept $A \cap \{a, b\}$, then the resulting implicit assertion is $A(a) \vee A(b)$.

The following example shows how the implicit assertions play a role in the semantics of a concept.
Example 5.1.2 Consider the following $\mathcal{ALC}$-concept formed by a conjunction of three existential quantifications

$$\exists R, (A \sqcap \{a, b\}) \sqsubseteq \exists R, (\neg A \sqcap \{a\}) \sqsubseteq \exists R, (\neg A \sqcap \{b\}) \sqsubseteq \exists R, (A \sqcap \{a\}) \sqsubseteq \exists R, (\neg A \sqcap \{b\}).$$

Suppose that we want to check its satisfiability. The standard approach to this problem is to separately check for the satisfiability of the three concepts involved in the existential quantifications, namely $A \sqcap \{a, b\}$, $\neg A \sqcap \{a\}$, and $\neg A \sqcap \{b\}$, for example using the trace rules. It is easy to see that such technique fails to recognize that the whole concept is unsatisfiable. In fact, although each of the conjuncts is separately satisfiable, the conjunction of their implicit assertion (i.e. $A(a) \lor A(b)$, $\neg A(a)$, and $\neg A(b)$) is unsatisfiable. \hfill \Box

5.1.2 Mixing Terminological and Assertional Knowledge

Another important characteristic of concept languages is that the reasoning process in the terminological component is in general not influenced by the assertional knowledge.

In Chapter 3, we have stated that Subsumption can be reduced to Subsumption w.r.t. a knowledge base with an empty ABox. The above property is crucial for the efficiency of reasoning in concept-based KR-systems. In fact, it allows for the maintenance of a static hierarchy of concepts, which is not influenced by the evolution of the ABox.

Unfortunately, such nice property does not hold when the language includes $\mathcal{O}$, as shown in the following example.

Example 5.1.3 Let $\Sigma = \{A(a), A(b)\}$. It is easy to see that

$$\forall R, \{a, b\} \not\subseteq \forall R, A.$$

In fact, given an interpretation $\mathcal{I}$ such that $R^\mathcal{I} = \{(d, a^\mathcal{I})\}$ and $A^\mathcal{I} = \emptyset$, then $d \in (\forall R, \{a, b\})^\mathcal{I}$ and $d \not\in (\forall R, A)^\mathcal{I}$. On the other hand

$$\Sigma \models (\forall R, \{a, b\} \subseteq \forall R, A).$$

That is because in every model of $\Sigma$ all the objects related only with $a$ and $b$ by means of $R$ are obviously related only to object in $A$. \hfill \Box

This is the reason why we call “mixed” the languages with individuals: They mix the intensional knowledge with the extensional one.
5.1.3 Abstraction

The problem of exploiting the Abstraction/Subsumption technique for Instance Checking is that, in general, it is not possible to completely fit the information relevant to an individual into a single concept of the language. For example, given the following knowledge base \( \Sigma = \{ R(a, a), B(a) \} \), the abstraction for \( a \) in \( \text{ACE} \) returns \( \text{MSC}(a) = B \sqcap \exists R.B \).

In \( \text{MSC}(a) \), the information that the individual related to \( a \) is exactly \( a \) itself is lost. In general, any time an individual is referred twice in the knowledge base, the connection between the two occurrences may be lost.

For this reason, the algorithms for Instance Checking based on abstraction are, in general, incomplete. For instance, in the above example, the Abstraction/Subsumption technique fails to draw the conclusion that \( \Sigma \models \exists R. \exists R.B(a) \).

As pointed out in [MB92] there are even other drawbacks about using the Abstraction/Subsumption technique. However they are out of the scope of this thesis.

Nevertheless, if the language includes \( O \) it is possible to make a lostless abstraction. In the previous example, if the language is \( \text{ACE}O \), the abstraction for \( a \) gives \( \text{MSC}(a) = \{ a \} \sqcap B \sqcap \exists R.\{ a \} \), and it is easy to see that the inference \( \Sigma \models \exists R. \exists R.B(a) \) is captured since \( \{ a \} \sqcap B \sqcap \exists R.\{ a \} \sqsubseteq \exists R. \exists R.B \) holds.

It follows that the use of \( O \) gives the possibility to complete reasoning using the Abstraction/Subsumption technique (see [DLN90, DE92] for a detailed discussion on this topic).

5.1.4 Number Restrictions and Negation

The ability offered by the constructor \( O \) to express concepts of a fixed extension gives also the possibility to express implicit number restrictions on the roles. If, for example, we assert the membership of an individual \( d \) to the concept \( \forall R.\{ a, b, c \} \) it implies that \( d \) is also in the extension of \( (\leq 3 R) \). For this reason, an inconsistency can be generated by the conjunction of two concept of the form \( \forall R.\{ a_1, \ldots, a_n \} \) and \( (\geq m R) \), in the case \( m > n \).

As pointed out in [BMP+91], using \( O \) it is possible to express the negation of a concept with respect to another concept. Let clarify this point by means of the following example (which is a slight modification of the example in [BMP+91, page 44]).

**Example 5.1.4** Consider the following three concepts

\[
\begin{align*}
C & = \forall R.\{ a, b \} \sqcap (\geq 1 R) \sqcap (\leq 1 R), \\
D_1 & = \forall R.\{ a \} \sqcap (\geq 1 R), \\
D_2 & = \forall R.\{ b \} \sqcap (\geq 1 R). 
\end{align*}
\]
The concept $C$ describes the individuals which have a single filler for the role $R$ and such filler is $a$ or $b$; $D_1$ and $D_2$ describe the individuals which have a single filler for the role $R$ and such fillers are respectively $a$ and $b$. Therefore, we have that $D_1 \cap D_2 = \bot$ and $D_1 \cup D_2 = C$, i.e. $D_1$ and $D_2$ are complementary with respect to $C$. \hfill \Box

### 5.1.5 Logical Connectives

Example 5.1.2 shows that, exploiting the implicit assertion, it is possible to express logical connectives between assertions. The explicit use of such connectives is usually not allowed in concept based systems.

Conversely, if the concept language includes $\mathcal{O}$ and $\mathcal{C}$, all the connectives can be simulated by atomic assertions. In order to show this point, we consider the language $\mathcal{ALCO}$ and we consider a complex-ABox in $\mathcal{ALCO}$, as defined in Chapter 2.

We now prove that in $\mathcal{ALCO}$ every complex-ABox $\Pi$ can be transformed in a simple-ABox $\Sigma = \Phi(\Pi)$ such that $\Pi$ is consistent if and only if $\Sigma$ is consistent.

Consider a generic assertion $\alpha$ in $\Pi$. Without loss of generality we suppose that $\alpha$ is in conjunctive normal form (CNF). Therefore, $\alpha$ has the form $c_1 \land \cdots \land c_m$, where each $c_i$ has the form $l_1 \lor \cdots \lor l_n$, and each $l_j$ has the form $p$ or $\neg p$ (where $p$ has either the form $C(a)$ or $R(a, b)$).

As a notation, if $\alpha$ has the form $C(a)$, we call $C_\alpha$ the concept $C$ involved in $\alpha$ and $a_\alpha$ the individual $a$. Furthermore, we assign to each clause in each assertion an individual $i_k$ that does not appear in $\Pi$; where $k$ is an integer that takes a different value for each clause. The transformation $\Phi$ is then defined by the following rules (where $Q$ is a role not appearing in $\Pi$):

\begin{align*}
\Phi(R(a, b)) &= \exists R. \{ b \}(a) && (5.1) \\
\Phi(C(a)) &= C(a) && (5.2) \\
\Phi(\neg p) &= \neg \Phi(p) && (5.3) \\
\Phi(l_1 \lor \cdots \lor l_n) &= \exists Q. (C_{\Phi(l_1)} \cap \{ a_{\Phi(l_1)} \}) \cup \cdots \cup \\
&\quad \quad \quad \exists Q. (C_{\Phi(l_n)} \cap \{ a_{\Phi(l_n)} \}) (i_k) && (5.4) \\
\Phi(c_1 \land \cdots \land c_m) &= \{ \Phi(c_j) \mid j = 1, \ldots, m \} && (5.5) \\
\Phi(\Pi) &= \bigcup_{\alpha \in \Pi} \Phi(\alpha) && (5.6)
\end{align*}

**Example 5.1.5** Consider the following complex knowledge base $\Pi$ in $\mathcal{ALCO}$:

\[ \Pi = \{ \exists R. D(a) \lor R(b, c), \neg R(a, b) \} \]
Applying the reduction \( \Phi \), we obtain:

\[
\Phi(\Pi) = \{ \exists Q. (\exists R. D \cap \{ a \} \cup \exists Q. ((\exists R. \{ c \} \cap \{ b \}))(i_1), \exists Q. ((\neg \exists R. \{ b \}) \cap \{ a \}))(i_2) \}
\]

\[\square\]

**Lemma 5.1.6** A complex knowledge base \( \Pi \) in \( \mathcal{ALCO} \) is consistent if and only if the knowledge base \( \Phi(\Pi) \) in \( \mathcal{ALCO} \) is consistent.

**Proof.** We prove the claim by showing that all the rules 5.1-5.6 are consistency preserving. In particular, it is easy to see that for each rule, but rule (5.4), \( \Phi(\Pi) \) is trivially equivalent to \( \Pi \). Therefore in order to prove the claim it is sufficient to show that \( \Pi \cup \{ l_1 \lor \cdots \lor l_n \} \) is consistent if and only if \( \Pi \cup \{ \Phi(l_1 \lor \cdots \lor l_n) \} \) is consistent.

\(\Rightarrow\) Suppose \( \Pi \cup \{ \Phi(l_1 \lor \cdots \lor l_n) \} \) consistent. Let \( I \) be a model of \( \Pi \cup \{ \Phi(l_1 \lor \cdots \lor l_n) \} \). Since \( I \) satisfies \( \Phi(l_1 \lor \cdots \lor l_n) \) there must be \( j \in \{ 1, \ldots, n \} \) such that \( I \models \exists Q C_{\Phi(l_j)} \cap \{ a_{\Phi(l_j)} \}(i_k) \). Hence, in \( \Delta^I \) there must exist an element \( d \) such that \( (i_k^I, d) \in Q^I \) and \( d \in (C_{\Phi(l_j)} \cap \{ a_{\Phi(l_j)} \})^I \). Therefore \( d = a_{\Phi(l_j)}^I \) and consequently \( a_{\Phi(l_j)}^I \in C_{\Phi(l_j)}^I \). It follows that \( I \) is a model of \( l_j \), therefore it is a models of \( l_1 \lor \cdots \lor l_n \) too.

\(\Leftarrow\) Suppose \( \Pi \cup \{ l_1 \lor \cdots \lor l_n \} \) consistent. Let \( I \) be a model of \( \Pi \cup \{ l_1 \lor \cdots \lor l_n \} \). There must exist one \( j \in \{ 1, \ldots, n \} \) such that \( I \models l_j \). Let \( I' \) be the interpretation that coincides with \( I \) except that \( (i_k^{I'}, a_{\Phi(l_j)}^{I'}) \in Q^I \). Since \( i_k \) does not appear in \( \Pi \cup \{ l_1 \lor \cdots \lor l_n \} \), it follows that \( I' \) is a model of \( \Pi \). Since \( a_{\Phi(l_j)}^I \in C_{\Phi(l_j)}^I \), it follows that \( a_{\Phi(l_j)}^{I'} \in C_{\Phi(l_j)}^{I'} \). and therefore \( a_{\Phi(l_j)}^{I'} \in (C_{\Phi(l_j)} \cap \{ a_{\Phi(l_j)}^I \})^I \). In conclusion, since \( (i_k^{I'}, a_{\Phi(l_j)}^{I'}) \in Q^I \) holds by construction of \( I' \), we have that \( I' \) is a model of \( \Pi \cup \{ \Phi(l_1 \lor \cdots \lor l_n) \} \) and therefore \( \Pi \cup \{ \Phi(l_1 \lor \cdots \lor l_n) \} \) is consistent.

\[\square\]

**Theorem 5.1.7** Consistency in \( \mathcal{ALCO} \) and Consistency of complex knowledge bases in \( \mathcal{ALCO} \) are problems polynomially reducible to each other.

**Proof.** Consistency is trivially reducible to complex knowledge base Consistency, being a particular case of it. The other direction is proved by Lemma 5.1.6 and the observation that \( \Phi \) is polynomial.

\[\square\]
5.1.6 Discussion on Reasoning with $O$

The above (not exhaustive) list of issues helps in understanding why reasoning with $O$ is generally hard. This hardness has a twofold explanation: On the one side, it is related to the implicit disjunction carried by the use of sets with more than one object. On the other side, it is due to the identification of unknown objects with individuals.

It is well known, that concept-based assertions can be translated into first-order formulae. The above explanation can be clarified looking at the translation in first-order formulae of assertions with $O$. For example, an assertion of the form

$$\exists R,\{a_1, \ldots, a_n\}(b)$$

is translated into the formula

$$R(b, x) \land (x = a_1 \lor \cdots \lor x = a_n).$$

This formula explicitly contains both disjunction and equality; they can be easily recognized as the causes of the hardness of reasoning with $O$.

At this point, it is important to note that, in some cases, the standard semantics gives results which are hard to be intuitively understood. In particular, the implicit assertion are difficult to be recognized and their role in the semantics can be mistaken or overlooked.

These two facts together, i.e. hardness and lack of intuition, explain why $O$ is usually treated in a non-standard way in the actual systems, as shown in Section 5.2.

5.1.7 Reasoning with $B$

As shown in Chapter 2, a concept of the form $R: a$ is equivalent to the concept $\exists R,\{a\}$. Hence $B$ can be viewed as a limited form of the combination of $E$ and $O$.

Reasoning with $E$ has been proved to be generally hard (see [DHL+92, DLNS94]). However, the hardness of $E$ is related to the possibility of nesting arbitrary numbers of existential quantification. Since $B$ does not offer the possibility of nesting, the issues related to $E$ do not regard $B$.

Regarding the relation with $O$, the main difference between $O$ and $B$ is that $B$ involves always a single individual. Therefore, all the issues related to the use of sets with more then one individual are not concerned to $B$.

The other difference between $O$ and $B$ is that the set of individuals involved in $B$ is always in the scope of an existential quantification. Even though this is a obvious limitation of the use of singleton sets, we now show that the issues related to the use of single-element sets are pertinent to $B$, too.
For instance, using $B$, it is possible to express implicit assertions. As an example, it is easy to see that the assertion

$$(\forall R.C) \cap (R : b)(a)$$

carries the implicit assertion $C(b)$.

As said before, since $B$ involves only singleton sets, disjunctive assertions cannot be expressed with $B$. However, as shown in Section 5.1.5, disjunctive implicit assertions can be obtained anyway by using single-element sets together with $U$. For example the following assertion

$$((\forall R.A) \cap (R : a)) \sqcup ((\forall R.B) \cap (R : b))(c)$$

carries the implicit assertion $A(a) \lor B(b)$.

Moreover, the independence of reasoning about concepts from the ABox, does not hold also in presence of $B$, as shown by the following example.

**Example 5.1.8** Let $\Sigma = \{A(a)\}$. Then

$$(R : a) \not\in \exists R.A.$$ 

Conversely, we have that

$$\Sigma \models ((R : a) \sqsubseteq \exists R.A).$$

\[\square\]

In conclusion, reasoning with $B$ has all the characteristic of reasoning with $O$ related to the presence of individual. On the other hand, the issues related to the implicit disjunction of $O$ are obviously not pertinent to $B$. However, when the language is supported with explicit disjunction ($U$), the use of $B$ becomes complex like the use of $O$.

### 5.2 Actual Systems and Mixed Languages

In this section we briefly describe the methods chosen by the implementors of the actual systems for dealing with $O$ and $B$, and with individuals in the concept descriptions in general. For this purpose we have chosen to describe two systems, namely **CLASSIC** and **BACK**. A more detailed description of them can be found in [BMP+91, BP94] and [QK90] respectively.

In **CLASSIC**, individuals are treated with a non-standard semantics. The reason why the **CLASSIC** designers have left the standard semantics is mostly related to the drawbacks described in previous section (in particular in Sections 5.1.1 and 5.1.2), and to the computational intractability of Subsumption
(see Section 5.4), which, in their opinion (see [BP94]), is not relegated only to a few non-practical worst cases.

Roughly speaking, the individuals appearing in concept descriptions are interpreted as disjoint concept names, i.e. as subsets of the domain, instead of as single elements of it. This semantics eliminates the effects of implicit assertions. In fact, the existence of an object in the concept \( C \cap \{ a \} \) does not tell that \( a^T \) is in \( C^T \), but only that \( a^T \) and \( C^T \) intersect each other. The fact that \( a^T \) and \( C^T \) intersect each other does not guarantee that \( \{ a^T \} \subseteq C^T \) and does not exclude the possibility that even \( a^T \) and \( \neg C^T \) intersect each other.

Moreover, in \textsc{classic}, the assertions on the individuals are not taken into account while reasoning with concepts. In other words, even when a ABox is non-empty, Subsumption is considered always as the ABox were empty. That semantics is weaker than the standard one; in fact, it fails to draw several conclusions that are entailed in the standard semantics.

The following example is a modified version of the one appearing in [BP94, page 13].

\textbf{Example 5.2.1} Let \( \Sigma \) be the following knowledge base in \textsc{classic}:

\[ \Sigma = \{ \forall \text{FRIEND,}\{\text{susan}\}(\text{john}), \text{Married(susan)} \} \]

The proposed semantics fails to draw the correct conclusion that the following entailment holds: \( \Sigma \models \forall \text{FRIEND. Married(john)} \). \ \Box

In \textsc{back}, \( O \) and \( B \) are not allowed. However, in \textsc{back} it is possible to express collections of elements, but these elements, called \textit{attributes}, belong to an alphabet disjoint from the alphabet of the individuals. Moreover, the domain of interpretation of the concepts is disjoint form the domain of interpretation of the attributes. A collection of attributes is not considered a concept and it is not allowed to be in conjunction with any concept, but it can appear only in the range of the quantification of a role.

This treatment avoids the reasoning complications of Section 5.1. It is simple and efficient; in fact reasoning with collections of attributes in \textsc{back} is just a matter of computing intersection, union, and difference between sets. The possible usefulness of sets of non-individual elements is argued also in [Bra92], where they are called \textit{Host Individuals}. However, they miss the expressive power of the full use of \( O \).

\section{5.3 Reductions between Reasoning Services}

We analyze the relationship between the complexity of the various reasoning tasks in mixed languages.
Before starting, we need one round of definitions. We call $IN_C$ the set of individuals appearing in $C$. In a similar way, we call $IN_{\Sigma}$ the set of all the individuals appearing in $\Sigma$ (either within the concepts or as the individual of which the assertion states the membership).

We first discuss the validity for mixed languages of relations (3.2-3.4) stated in Chapter 3. It is easy to see that relations (3.2) and (3.4) hold for mixed languages as well. Conversely, relations (3.1) and (3.3) do not hold as they are, as shown by the following example for relation (3.1). The intuitive reason is that relations (3.1) and (3.3) involve assertions of the form $C(a)$, and, if $C$ involves some individuals, there might be an interaction between $C$ and $a$.

**Example 5.3.1** Consider the following $ACO$-concept:

$$\text{Student} \cap \{\text{mary, john, peter}\}.$$ 

It is trivially satisfiable, whereas the corresponding knowledge base:

$$\{\text{Student} \cap \{\text{mary, john, peter}\}(b)\}$$

is clearly inconsistent (due to the Unique Name Assumption). □

Nevertheless, a variant of relations (3.1) and (3.3) hold for mixed languages, as stated by the following lemma.

**Proposition 5.3.2** Let $\mathcal{L}$ be a mixed language, $C$ and $D$ two $\mathcal{L}$-concepts, and $b$ an individual not appearing in $C$ and $D$:

\[
\begin{align*}
C \neq \bot & \iff \exists a \in (IN_C \cup \{b\}) \mid \{C(a)\} \not\models (5.7) \\
C \subseteq D & \iff \forall a \in (IN_C \cup IN_D \cup \{b\}) \mid \{C(a)\} \models D(a) (5.8)
\end{align*}
\]

**Proof.**

(5.7) “$\models$” Suppose $C$ satisfiable and $\forall a \in (IN_C \cup \{b\}) \mid \{C(a)\}$ is inconsistent. Since $C$ is satisfiable, there exists an interpretation $I$ such that $d \in \Delta^I$ and $d \in C^I$. The element $d$ can be either the interpretation of an individual in $IN_C \cup \{b\}$ or not. We show that in both cases we reach a contradiction.

- $\exists a \in IN_C \cup \{b\} : d = a^I$. Contradicts the hypothesis that $\{C(a)\}$ is inconsistent.

(5.7) “$\models$” Suppose $C$ satisfiable and $\forall a \in (IN_C \cup \{b\}) \mid \{C(a)\}$ is inconsistent. Since $C$ is satisfiable, there exists an interpretation $I$ such that $d \in \Delta^I$ and $d \in C^I$. The element $d$ can be either the interpretation of an individual in $IN_C \cup \{b\}$ or not. We show that in both cases we reach a contradiction.

- $\exists a \in IN_C \cup \{b\} : d = a^I$. Contradicts the hypothesis that $\{C(a)\}$ is inconsistent.
\[ \forall a \in \text{IN}_C \cup \{b\} : d \neq a^T. \] Let \( I' \) be the interpretation that coincides with \( I \) except that \( b^{I'} = d \). Since \( b \) does not appear in \( C \), it follows that the interpretation of \( b \) as no influence on the interpretation of \( C \), and therefore \( b^{I'} \in C^{I'} \). This contradicts the hypothesis that \( \{C(b)\} \) is inconsistent.

\[ \Rightarrow \] Suppose \( \exists a \in (\text{IN}_C \cup \{b\}) \mid \{C(a)\} \) is consistent. It follows that there exists an interpretation \( I \) such that \( a^I \in C^I \), therefore \( C \) is satisfiable.

\((5.8)\)

\[ \Leftarrow \] Assume \( C \subseteq D \) and \( \exists a \in (\text{IN}_C \cup \text{IN}_D \cup \{b\}) : \{C(a)\} \not\models D(a) \). From \( \{C(a)\} \not\models D(a) \) it follows that there exists an interpretation \( I \) such that \( a^I \in C^I \), \( a^I \not\in D^I \). This contradicts the hypothesis that \( C \subseteq D \).

\[ \Rightarrow \] Assume \( \forall a \in (\text{IN}_C \cup \text{IN}_D \cup \{b\}) : \{C(a)\} \models D(a) \) and \( C \not\subseteq D \). From \( C \not\subseteq D \), it follows that there exists an interpretation \( I \) and an element \( d \) of \( \Delta^I \) such that \( d \in C^I \), and \( d \not\in D^I \). The element \( d \) can be either the interpretation of an individual in \( \text{IN}_C \cup \text{IN}_D \cup \{b\} \) or not. We show that in both cases we reach a contradiction.

- Suppose that \( d = a^I \) for some \( a \in \text{IN}_C \cup \text{IN}_D \cup \{b\} \); then \( a^I \in C^I \) and \( a^I \not\in D^I \) contradicting the hypothesis that \( \{C(a)\} \models D(a) \).
- Suppose that \( d \neq a^I \) for all \( a \in \text{IN}_C \cup \text{IN}_D \cup \{b\} \) then let \( I' \) be the interpretation that coincides with \( I \) except that \( b^{I'} = d \). Since \( b \) does not appear in \( C \) and \( D \), it follows that \( b^{I'} \in C^{I'} \), \( b^{I'} \not\in D^{I'} \). This contradicts the hypothesis that \( \{C(b)\} \models D(b) \).

\[ \square \]

The above result is stated for Pure Subsumption. Regarding ABox Subsumption, we know based on the observations of Section 5.1.2, that it does not easily collapse to (Pure) Subsumption. To the aim of studying all the reasoning services, the next proposition shows a bidirectional transformation between ABox Subsumption and Instance Checking in any language including \( \mathcal{O} \) and states its correctness.

**Proposition 5.3.3** Let \( \mathcal{L} \) be a concept language including \( \mathcal{O} \), \( \Sigma \) an knowledge base in \( \mathcal{L} \), \( C, D \) two \( \mathcal{L} \)-concepts, \( a \) an individual, and \( b \) an individual not in \( \text{IN}_\Sigma \cup \text{IN}_C \cup \text{IN}_D \), then:

\[ \Sigma \models D(a) \iff \Sigma \models \{a\} \subseteq D \]  \hspace{2cm} (5.9)

\[ \Sigma \models C \subseteq D \iff \forall a \in (\text{IN}_\Sigma \cup \text{IN}_C \cup \text{IN}_D \cup \{b\}) : \Sigma \cup \{C(a)\} \models D(a) \]  \hspace{2cm} (5.10)
Proof.

(5.9) If Σ is inconsistent, then both Σ ⊨ D(a) and Σ ⊨ \{a\} ⊆ D trivially hold. Therefore, we can suppose Σ consistent. If Σ is consistent then it has at least one model. Let \mathcal{I} be a generic model, obviously a^\mathcal{I} ∈ D^\mathcal{I} if and only if \{a^\mathcal{I}\} ⊆ D^\mathcal{I}. The claim follows.

(5.10) 

"⇐" Assume Σ ⊨ C ⊆ D and ∃a ∈ (IN_Σ ∪ IN_C ∪ IN_D ∪ \{b\}) : Σ ∪ \{C(a)\} ∭ D(a). From Σ ∪ \{C(a)\} ∭ D(a) it follows that there exists an interpretation \mathcal{I} such that \mathcal{I} is a model of Σ, and a^\mathcal{I} ∈ C^\mathcal{I}, a^\mathcal{I} \notin D^\mathcal{I}. This contradicts the hypothesis that Σ ⊨ C ⊆ D.

"⇒" Assume ∀a ∈ IN_Σ ∪ IN_C ∪ IN_D ∪ \{b\} : Σ ∪ \{C(a)\} ⊨ D(a) and Σ ∤ C ⊆ D. From Σ ∤ C ⊆ D, it follows that there exists an interpretation \mathcal{I} and an element d of \Delta^\mathcal{I} such that \mathcal{I} is a model of Σ, d ∈ C^\mathcal{I}, and d ∉ D^\mathcal{I}. The element d can be either the interpretation of an individual in IN_Σ ∪ IN_C ∪ IN_D ∪ \{b\} or not. We show that in both cases we reach a contradiction.

- Suppose that d = a^\mathcal{I} for some a ∈ IN_Σ ∪ IN_C ∪ IN_D ∪ \{b\}; then a^\mathcal{I} ∈ C^\mathcal{I} and a^\mathcal{I} ∉ D^\mathcal{I} contradicting the hypothesis that Σ ∪ \{C(a)\} ⊨ D(a).
- Suppose that d ≠ a^\mathcal{I} for all a ∈ IN_Σ ∪ IN_C ∪ IN_D ∪ \{b\} then let \mathcal{I}' be the interpretation that coincides with \mathcal{I} except that b^\mathcal{I}' = d. Since b does not appear in Σ, C, and D, it follows that \mathcal{I}' is a model of Σ, and b^\mathcal{I}' ∈ C^\mathcal{I}', b^\mathcal{I}' ∉ D^\mathcal{I}'. This contradicts the hypothesis that Σ ∪ \{C(b)\} ⊨ D(b).

□

Proposition 5.3.3 states that in order to solve ABox Subsumption, we make a linear number of Instance Checking tests, one for each individual appearing in the concepts and in the knowledge base plus one new. It is interesting to observe that, when \mathcal{O} is not used, it is sufficient to make a single Instance Checking test, by considering only the new individual b. The following example shows that, on the contrary when \mathcal{O} is used, this is not true. Let Σ = \emptyset, it is easy to see that Σ ∤ (C ∪ \{a\} ⊆ C). However, due to the Unique Name Assumption, \{C ∪ \{a\}(b)\} ⊨ C(b) holds. In fact, the latter relation holds for every individual except a, therefore it is necessary to include a in the set of individuals we consider.

Next we show that in the languages with \mathcal{O} and \mathcal{B}, Consistency can be reduced to Concept Satisfiability. In order to achieve this result, we present the transformation \phi from a knowledge base to a concept defined as follows:
Let $L$ be a concept language including $O$ and $B$, $\Sigma$ a knowledge base in $L$, $C, D$ two $L$-concepts, and $a, b$ two individuals, then:

$$
\phi(C(a)) = \exists Q_i \cap \forall Q_i, (\{a\} \cap C)
$$

$$
\phi(R(a, b)) = \exists Q_i \cap \forall Q_i, (\{a\} \cap R; b)
$$

$$
\phi(\Sigma) = \forall \{\alpha \in \Sigma\} \phi(\alpha)
$$

where $i$ has a different value for each assertion and $Q_i$ does not appear in $\Sigma$ for each $i$. Intuitively, $\phi$ "encodes" the knowledge base $\Sigma$ in the implicit assertions of the concept $\phi(\Sigma)$. Such encoding is done in a way that the only possible cause of unsatisfiability of $\phi(\Sigma)$ comes from the implicit assertions. It follows that $\phi(\Sigma)$ is satisfiable if and only if $\Sigma$ is consistent, as stated by the following lemma.

**Lemma 5.3.4** Given a language $L$ and a knowledge base $\Sigma$ in $L$, then $\Sigma$ is consistent if and only if $\phi(\Sigma)$ is satisfiable.

*Proof.*

"$\Rightarrow$" Suppose $\phi(\Sigma)$ satisfiable and let $I$ be one of its models. There exists an element $d$ in $\Delta_I$ such that $d \in \phi(\Sigma)^I$. By definition of $\phi$, for each assertion in $\Sigma$ of the form $C(a)$ (resp. $R(a, b)$) we have that $(d, a^I) \in Q^I_k$, for some $k$, and $a^I \in C^I$ (resp. $(a^I, b^I) \in R^I$). Hence $I$ is a model of $\Sigma$.

"$\Leftarrow$" Suppose $\Sigma$ consistent and let $I$ be one of its models. Since for each $i$, $Q_i$ does not appear in $\Sigma$, we can assume, without loss of generality, that $Q^I_i = \emptyset \times \emptyset$ for each $i$. Let $I'$ be the interpretation such that $\Delta_I' = \Delta_I \cup \{d\}$ and $\cdot^I' = \cdot^I$ except that for each conjunct of $\phi(\Sigma)$ we have $(d, a^I) \in Q^I_k$. It is easy to see that $d \in (\phi(\Sigma))^I'$ and therefore $\phi(\Sigma)$ is satisfiable.

\[\Box\]

Exploiting the same idea of Lemma 5.3.4, the next lemma shows that in the languages with $O$ and $B$, Instance Checking can be reduced to Subsumption.

**Lemma 5.3.5** Given a language $L$, a knowledge base $\Sigma$ in $L$, an individual $a$, and an $L$-concept $C^2$ then $\Sigma \models C(a)$ if and only if $\phi(\Sigma) \cap \{a\} \subseteq C$

*Proof.*

1 If $L$ includes $E$ then the concept $\exists Q_i \cap \forall Q_i, (\ldots)$ can be simplified in $\exists Q_i, (\ldots)$ and the condition on $i$ can be dropped. In fact this condition is needed to avoid the interaction between the universal and existential quantifications coming up from different assertions.

2 We assume that $Q_i$ (for each $i$) does not appear in $C$. Otherwise it is possible to modify $\Phi$ in a way such that this condition is fulfilled.
“⇒” Assume $\Sigma \models C(a)$ and $\phi(\Sigma) \cap \{a\} \not\subseteq C$. Since $\phi(\Sigma) \cap \{a\} \not\subseteq C$, there exists an interpretation $I$ and an element $d \in \Delta^I$ such that $d \in (\phi(\Sigma) \cap \{a\})^I$ and $d \notin C^I$. From $d \in (\phi(\Sigma) \cap \{a\})^I$, it follows that $a^I = d$ and therefore $a^I \in (\phi(\Sigma))^I$ and $a^I \notin C^I$. Since $a^I \in (\phi(\Sigma))^I$ holds, $I$ is a model of $\phi(\Sigma)$ and therefore (see proof of Lemma 5.3.4) it is a model of $\Sigma$ too. Since $a^I \notin C^I$ and $I$ is a model of $\Sigma$, the assumption $\Sigma \models C(a)$ is contradicted.

“⇐” Assume $\phi(\Sigma) \cap \{a\} \subseteq C$ and $\Sigma \nvdash C(a)$. From $\Sigma \nvdash C(a)$ it follows that there exists an interpretation $I$ such that $I$ satisfies $\Sigma$ and $a^I \notin C^I$. Since $I$ satisfies $\Sigma$, there exists $I'$ that satisfies $\phi(\Sigma)$ (see proof of Lemma 5.3.4). Suppose $d \in \phi(\Sigma)^{I'}$. Since for each $i$, $Q_i$ does not appear in $C$ and the only necessary property of $d$ concern $Q_i$, it is possible to assume that $d \notin C^{I'}$. Let $I''$ be the interpretation obtained from $I'$ assigning $a^{I''} = d$. Then $a^{I''} \notin C^{I''}$ and $a^{I''} \in \phi(\Sigma)^{I''}$. Therefore $a^{I''} \in (\phi(\Sigma) \cap \{a\})^{I''}$ contradicting the assumption $\phi(\Sigma) \cap \{a\} \subseteq C$.

\begin{flushright}
\Box
\end{flushright}

**Theorem 5.3.6** In any language including $O$ and $B$, Consistency is polynomially reducible to Concept Satisfiability and Instance Checking is polynomially reducible to Subsumption.

**Proof.** Follows from Lemmata 5.3.4 and 5.3.5 and the observation that $\phi$ is polynomial. \(\Box\)

Summarizing the results obtained so far, for concept languages with $O$ and $B$, Instance Checking and ABox Subsumption are reducible to each other, Consistency is reducible to Concept Satisfiability and Instance Checking is reducible to Subsumption.

The results of Theorem 5.3.6 are important because they relate ABox problems with pure ones. To this regard, in Chapter 4, it was already proved that for a large class of languages Concept Satisfiability and Consistency are in the same complexity class, and therefore the latter is reducible to the former. However, that result is achieved considering each language separately. The result obtained here is stronger, in the sense that it is proved independently of the single language (provided that $O$ and $B$ are included). This relationship is crucial for the design of efficient reasoning algorithms and it is still not completely clear.

In Chapter 4, it was also shown that in $\mathcal{ALC}$ Instance Checking and Subsumption are in different complexity classes. In particular, it is shown that in $\mathcal{ALC}$ Instance Checking is PSPACE-complete while Subsumption is NP-complete. Therefore (assuming $NP \neq PSPACE$), such result proves that,
5.4 Complexity Results for Mixed Languages

In this section, we investigate the complexity of the above problems in the specific languages. To this aim, we consider various languages that do not use $\mathcal{O}$ and $\mathcal{B}$ and the corresponding languages obtained by adding them. In particular we focus on the pure languages $\mathcal{ALC}$, $\mathcal{ALE}$, and $\mathcal{AL}$, which are a good representative of the various degrees of expressiveness (and complexity), and we achieve some complexity results on the corresponding languages with $\mathcal{O}$ and $\mathcal{B}$. We concentrate on the subsumption problem. Results for the other reasoning tasks easily follow.

5.4.1 Reasoning in $\mathcal{ALCO}$

The calculus we presented in Chapter 3 can be turned into an actual procedure for reasoning in $\mathcal{ALCO}$. However, such a procedure should store, in a suitable data structure, the whole constraint system involved in the computation. Unfortunately, the constraint system might have exponential size, therefore a naive procedure that merely implements the calculus requires at least exponential space.

On the other hand, we have seen that every reasoning task in $\mathcal{ALC}$ is PSPACE-complete. We have also seen that the basic idea underlying the algorithms for $\mathcal{ALC}$ is that, although the whole constraint system might have exponential size, it is possible to keep track of only a trace of it at a time, and traces have polynomial size.

However, when the language is $\mathcal{ALCO}$, the above idea is no longer valid. In fact, because of the presence of implicit assertions, the traces are not independent. On the contrary, the satisfiability of one trace can depend on the constraints in the other traces, as shown by Example 5.1.2. In the calculus this possibility is taken into account by the $\rightarrow_{\mathcal{O}}$-rule, which allows for the substitution of variable in different traces with the same individual.

The next example shows that a naive trace-based calculus would fail in making inferences in $\mathcal{ALCO}$.

**Example 5.4.1** Given the following knowledge base in $\mathcal{ALCO}$

$$\Sigma = \{ \forall R.A(a), \forall Q.A(a) \},$$
consider the query

\[ D = (\forall R. \{ b \} \cup \forall Q. \{ b \}) \]

In order to check whether \( \Sigma \models D(a) \), we consider the constraint system \( S = S_\Sigma \cup \{ a : \neg D \} \), which is the following

\[ S = \{ a : \forall R. A, a : \forall Q, \neg A, a : \exists R. \{ b \} \cap \exists Q, \{ b \} \}. \]

Applying the completion rules to \( S \) we obtain the following constraint system \( S' = S \cup \{ a Rx, x : A, x : \{ b \}, a Q y, y : \neg A, y : \{ b \} \} \). Now the application of the \( \rightarrow \sigma \)-rule to both the constraints \( x : \{ b \} \) and \( y : \{ b \} \) leads to the clash, proving that \( \Sigma \models D(a) \).

Conversely, both the traces derivable from \( S \), i.e. \( S_1 = S \cup \{ a Rx, b : A, b : \{ b \} \} \) and \( S_2 = S \cup \{ a Q y, b : \neg A, b : \{ b \} \} \), are satisfiable. \( \square \)

For the above reason, in this section we modify the calculus in such a way that it can be turned into an effective procedure. We also prove that our procedure is correct and actually works in polynomial space. Since reasoning in \( \mathcal{ALC} \) is already a PSPACE-complete problem, the proposed procedure turns out to be optimal (at this level of granularity).

The idea is that, although the number of variable might be exponential, the individuals are polynomial in number. Therefore, the complete information about individuals can be store in polynomial space. Exploiting this property, the calculus is equipped with a completion rule that guesses nondeterministically for each individual the set of concepts of which the individual is in the extension (as proposed also in [HB91]). In this way, it is possible to use the trace rules, allowing though only the substitutions that agree with the given guess.

The modified calculus is obtained by replacing the \( \rightarrow \sigma \)-rule with the rule given below and adding a new clash case. Before presenting the rule and the clash case, we need the following definition.

**Definition 5.4.2** Given a constraint system \( S \), we denote by \( \mathcal{O}_S \) the set of individuals appearing in \( S \), we call subconcept of \( S \) any subconcept of a concept \( C \) appearing in \( S \). We call \( \text{Sub}(S) \) the set of all the subconcepts of \( S \). Given an individual \( a \in \mathcal{O}_S \) we define the set of concepts \( S_a \) in the following way

\[ S_a = \{ C \mid (a : C) \in S \}. \]

The new completion rule is:

- \( S \rightarrow_{\text{guess}} \{ a : D \} \cup S \)
  - if \( a \in \mathcal{O}_S \), \( C \in \text{Sub}(S) \), \( D = C \) or \( D = \neg C \), and neither \( a : C \) nor \( a : \neg C \) is in \( S \);
We add the following condition to the list of clashes:

7. there exists a variable \( x \) such that \( x: \{a_1, \ldots, a_n\} \) is in \( S \), the \( \rightarrow \text{guess} \)-rule is not applicable, and \( S_{a_i} \not\subseteq S_x \) for all \( i \in \{1, \ldots, n\} \).

Intuitively, the condition \( S_{a_i} \subseteq S_x \) ensures that \( x \) can be successfully substituted with \( a_i \) without leading to unsatisfiability. Conversely, if \( S_{a_i} \not\subseteq S_x \) then the constraint system obtained substituting \( x \) with \( a_i \) is unsatisfiable, as proved later by Lemma 5.4.5. And so, if all possible substitution for \( x \) lead to an unsatisfiable constraint system then the constraint \( x: \{a_1, \ldots, a_n\} \) represents a clash.

The following definition is needed for the design of an algorithm based on the modified calculus.

**Definition 5.4.3** Given a constraint system \( S \), we call \( T_S \) a constraint system (nondeterministically) obtained from \( S \) by the exhaustive applications of the \( \rightarrow \text{guess} \)-rule only.

Before presenting the algorithm, we need one more definition. A constraint \( \sigma \) is closed in a constraint system in the following three cases:

- \( \sigma \) is \( s:C \cap D \), and both \( x:C \) and \( x:D \) are in \( S \);
- \( \sigma \) is \( s:C \cup D \), and either \( x:C \) or \( x:D \) is in \( S \);
- \( \sigma \) is \( s: \exists R.C \), and there exists \( t \) such that \( sRt \) and \( t:C \) are in \( S \);

Intuitively, a constraint is closed if no rule applies to it. A constraint in \( S \) is open if it not closed.

The algorithm for Concept Satisfiability is shown in Figure 5.1. It faithfully follows the modified calculus, except that closed constraints are removed and the traces are checked independently. However, notice that the information regarding the individuals is passed to each trace, in order to make it available to each trace.

For the proof of the correctness of the algorithm we need the following lemmata.

**Lemma 5.4.4** Given a constraint system \( S \), \( S \) is satisfiable if and only if there exists a \( T_S \) such that \( T_S \) is satisfiable.

**Proof.** \( \Rightarrow \) Suppose \( S \) satisfiable. Let \( I \) be a model of \( S \). Let \( T_S \) be obtained in the following way: For each pair \( C, a \) such that \( C \in \text{Sub}(S) \) and \( a \in \Omega_S \), if \( a^T \in C^T \) then \( a:C \in T_S \) otherwise \( a:C \not\in T_S \). \( T \) trivially satisfies \( T_S \). The claim follows.

\( \Leftarrow \) Suppose there exists a satisfiable \( T_S \), and let \( I \) be a model of \( T_S \). \( I \) is obviously a model of \( S \), and therefore \( S \) is satisfiable. \( \square \)
Algorithm $Sat(C)$;
Input $\mathcal{ALCO}$-concept $C$;
Output TRUE if $C \not= \bot$; FALSE otherwise
begin
  $S := \{x : C\}$;
  loop
    compute a new $T_S$;
    if $\text{clash free}(T_S)$
      then return TRUE
  endloop;
  return FALSE
end.

Function $\text{clash free}(S : \text{constraint system}) : \text{boolean}$;
begin
  if $(s : A, s : \neg A \in S)$ or $(s : \bot \in S)$ or
  $(a : \{a_1, \ldots, a_n\} \in S \text{ with } a \not= a_i \text{ (for all } i = 1, \ldots, n))$ or
  $(a : \neg\{a_1, \ldots, a_n\} \in S \text{ with } a = a_i \text{ (for some } i = 1, \ldots, n))$
    then return false
  elseif $x : \{a_1, \ldots, a_n\} \in S$
    then return (exists $i : S_x \subseteq S_{a_i}$)
  elseif $s : C_1 \cap C_2 \in S$
    then return $\text{clash free}(S \setminus \{s : C_1 \cap C_2\} \cup \{s : C_1, s : C_2\})$
  elseif $s : C_1 \cup C_2 \in S$
    then return $\text{clash free}(S \setminus \{s : C_1 \cup C_2\} \cup \{s : C_1\})$ or
      $\text{clash free}(S \setminus \{s : C_1 \cup C_2\} \cup \{s : C_2\})$
  elseif $s : \exists P.C_1 \in S$
    then return $\text{clash free}(\{x : C_1\} \cup \{x : D \mid s : \forall P.D \in S\} \cup \{s : \exists P.C_1\})$ and
      $\text{clash free}(S \setminus \{s : \exists P.C_1\})$
  else return true
end;

Figure 5.1: The algorithm for Concept Satisfiability in $\mathcal{ALCO}$
Lemma 5.4.5 Given a constraint system $S$, let $S'$ be a constraint system obtained from a $T_S$ by a sequence of applications of the completion rules. Let $x$ be a variable in $S'$ and $a$ an individual in $O_S$. Then $S'[x/a]$ is satisfiable if and only if $S'$ is satisfiable and $S'_x \subseteq S'_a$.

Proof. 

$\Rightarrow$

- Suppose $S'_x \not\subseteq S'_a$. Then there exists a concept $C$ such that $x : C \in S'$ and $a : C \not\in S'$. It follows that $a : C$ is not in $T_S$, hence, from the definition of $T_S$, $a : \neg C$ is in $T_S$. Since no constraint can be eliminated by the application of the rules, $a : \neg C$ is in $S'$ too. It follows that both $x : C$ and $a : \neg C$ are in $S'$, thus both $a : C$ and $a : \neg C$ are in $S'[x/a]$. It follows that $S'[x/a]$ is unsatisfiable.

- Suppose $S'$ is unsatisfiable and $S'[x/a]$ satisfiable. Let $I, \alpha$ be a pair that satisfies $S'[x/a]$. Obviously $x$ do not appear in $S'[x/a]$, therefore the pair $I, \alpha'$, where $\alpha'$ coincides with $\alpha$ except that $\alpha(x) = a^I$, satisfies $S'$, contradicting the hypothesis.

$\Leftarrow$

Suppose $S'[x/a]$ is unsatisfiable, $S'$ satisfiable, and $S'_x \subseteq S'_a$. Let $I_{S'}, \alpha_{S'}$ be the canonical model and the canonical $I_{S'}$-assignment of $S'$. Consider the $I_{S'}$-assignment $\alpha'$ that coincides with $\alpha_{S'}$ except that $\alpha'(x) = a^I$. Since $S'_x \subseteq S'_a$, $I_{S'}, \alpha'$ is also a model of $S'$ and it is therefore a model of $S'[x/a]$ contradicting the hypothesis.

$\square$

It follows from Lemma 5.4.5 that checking the satisfiability of a constraint system of the form $S[x/a]$ can be done without making the substitution explicitly, but simply checking the condition $S_x \subseteq S_a$, as done in the algorithm $Sat(C)$ of Figure 5.1. A further consequence of this fact is that each trace can be checked for a clash independently. In fact, it can be easily verified, that no clash can involve two variables belonging to different traces.

Theorem 5.4.6 Algorithm $Sat(C)$ is correct and terminating.

Proof. Follows from Lemmata 5.4.4 and 5.4.5 and from the results in [SS91, BH91b] on the independence of the traces in $\mathcal{ALC}$. $\square$

We now turn the attention to the complexity of the algorithm. First of all, notice that the size of each $T_S$ is polynomially bounded by the size of $C$, as stated by the following lemma.
Lemma 5.4.7 Given a constraint system $S = \{x : C\}$, for each $T_S$ the following relation holds:

$$|T_S| = O(|C|^3)$$

Proof. Each $T_S$ contains exactly $|\text{Sub}(S)| \times |\mathcal{O}_S|$ constraints. The size of each constraint in $T_S$ is obviously bounded by of the size of $S$. Therefore $|T| \leq |S| \times |\text{Sub}(S)| \times |\mathcal{O}_S|$. Since $|\text{Sub}(S)|$ and $|\mathcal{O}_S|$ are bounded by $|S|$ and $|S|$ is equal to $|C|$ (but a constant value), the claim follows.

Lemma 5.4.8 Algorithm Sat($C$) works in polynomial space.

Proof. Easily follows from Lemma 5.4.7 and the results for $\mathcal{ALC}$. □

Since Subsumption in $\mathcal{ALCO}$ is reducible to Concept Satisfiability, we can conclude with the following theorem, stating that adding the $O$ constructor to the language $\mathcal{ALC}$ does not change the complexity class of the reasoning services.

Theorem 5.4.9 Subsumption in $\mathcal{ALCO}$ is a PSPACE-complete problem.

5.4.2 Reasoning in $\mathcal{ALCO}$

In Chapter 4, it is proven that Instance Checking in $\mathcal{ALC}$ is PSPACE-complete. It follows that Instance Checking in $\mathcal{ALCO}$ is PSPACE-hard, and therefore (Proposition 5.3.5) Subsumption in $\mathcal{ALCO}$ is PSPACE-hard too. Since Subsumption in $\mathcal{ALC}$ is PSPACE-complete, it is in PSPACE for $\mathcal{ALCO}$ too (remind that $\mathcal{ALCO}$ is a superlanguage of $\mathcal{ALC}$).

Theorem 5.4.10 Subsumption in $\mathcal{ALCO}$ is a PSPACE-complete problem.

5.4.3 Reasoning in $\mathcal{ALC}$

In this section we state that Subsumption in $\mathcal{ALC}$ is coNP-complete, besides the fact that in $\mathcal{AL}$ both Subsumption and Instance Checking are in P, as proved in [DLN91a] and Chapter 4, respectively. Since $\mathcal{ALC}$ does not have $\mathcal{E}$, $\mathcal{ALC}$ is not equivalent to $\mathcal{ALCB}$ and therefore (unlike previous sections) the results obtained for the language with $\mathcal{O}$, are not directly extended to the language with $\mathcal{O}$ and $\mathcal{E}$. However, it can be easily proved that the results we obtain here for $\mathcal{ALC}$ are valid for $\mathcal{ALCO}$, too.

In order to prove the coNP-hardness of Subsumption, we now prove the NP-hardness of Concept Satisfiability which obviously implies that Subsumption is coNP-hard.
This proof is based on a reduction from SAT, the satisfiability problem for a propositional conjunctive normal form (CNF) formula, to the Concept Satisfiability problem in \( \mathcal{ACO} \).

We define a pos-neg CNF-formula, a CNF-formula \( \Gamma' \) such that every clause of \( \Gamma' \) is either positive (i.e. it is constituted by positive literals) or negative (i.e. it is constituted by negative literals).

Notice, first of all, that any CNF formula \( \Gamma \) can be transformed in polynomial time into a pos-neg CNF formula \( \Gamma' \) such that \( \Gamma \) is satisfiable if and only if \( \Gamma' \) is so. This is done by replacing every clause of \( \Gamma \) by two clauses, as follows:

\[
p_1 \lor \ldots \lor p_n \lor \sim q_1 \lor \ldots \lor \sim q_m \Rightarrow
\]
\[
(p_1 \lor \ldots \lor p_n \lor r), \quad (\sim q_1 \lor \ldots \lor \sim q_m \lor \sim r)
\]

where \( r \) is a new propositional variable.

Let \( \Psi \) be the transformation from a pos-neg CNF formula \( \Gamma = \alpha_1^+ \land \cdots \land \alpha_n^+ \land \alpha_1^- \land \cdots \land \alpha_m^- \) to the \( \mathcal{ACO} \)-concept \( \Psi(\Gamma) = C_1^+ \cap \cdots \cap C_n^+ \cap C_1^- \cap \cdots \cap C_m^- \), specified by the following equations:

\[
C_1^+ = \exists R_1^+ \cap \forall R_1^- . (\text{obj}(\alpha_1^+) \cap A),
\]
\[
C_1^- = \exists R_1^- \cap \forall R_1^+ . (\text{obj}(\alpha_1^-) \cap \sim A)
\]

where \( \alpha_i^+ \) (resp. \( \alpha_i^- \)) denotes a positive (resp. negative) clause, \( A \) is a concept name, \( R_h^+ \) and \( R_h^- \) (for \( h = 1, \ldots, n \) and \( k = 1, \ldots, m \)) are roles, and \( \text{obj}(\alpha) \) denotes the concept \( \{p_1, \ldots, p_k\} \), where \( p_1, \ldots, p_k \) are all the propositional letters in the clause \( \alpha \). In other words, we associate with every propositional letter of \( \Gamma \) an individual with the same name, and with every clause \( \alpha \) of \( \Gamma \) the collection of individuals \( \text{obj}(\alpha) \).

For example, if \( \Gamma = (p \lor q) \land (\sim p \lor \sim r) \), then the corresponding \( \mathcal{ACO} \)-concept is:

\[
\Psi(\Gamma) = \exists R_1^+ \cap \forall R_1^- . (\{p, q\} \cap A) \cap \exists R_1^- \cap \forall R_1^+ . (\{p, r\} \cap \sim A).
\]

**Lemma 5.4.11** A pos-neg CNF formula \( \Gamma \) is satisfiable if and only if the corresponding \( \mathcal{ACO} \)-concept \( \Psi(\Gamma) \) is satisfiable.

**Proof.**

\( \Leftarrow \)  Suppose \( \Psi(\Gamma) \) satisfiable. Then there exists an interpretation \( I \) and an object \( d \) such that \( d \in (\Psi(\Gamma))^I \). Looking at the structure of \( \Psi(\Gamma) \), one can verify that there are \( n + m \) objects \( d_1^+, \ldots, d_n^+, d_1^-, \ldots, d_m^- \) in \( \Delta^I \) such that

\[
(d, d_1^+) \in (R_1^+)^I, \ldots, (d, d_n^+) \in (R_n^+)^I,
\]
\[
(d, d_1^-) \in (R_1^-)^I, \ldots, (d, d_m^-) \in (R_m^-)^I;
\]
and for each $i \in \{1, \ldots, n\}$, $d^+_i \in (\text{obj}(\alpha^+_i))^T$ and $d^-_i \in A^T$, and for each $i \in \{1, \ldots, m\}$, $c^-_i \in (\text{obj}(\alpha^-_i))^T$ and $c^-_i \in (-A)^T$. Now construct a truth assignment $J$ for $\Gamma$ as follows: for every letter $p$, if $p^T \in A^T$, then $J(p) = \text{true}$, else $J(p) = \text{false}$. Due to the above properties, for every clause $\alpha^+_i$ (resp. $\alpha^-_j$), $J$ assigns true (resp. false) to at least one literal in $\alpha^+_i$ (resp. $\alpha^-_j$), and therefore $J$ satisfies every clause of $\Gamma$.

\[ \Rightarrow \] Suppose $\Gamma$ satisfiable. There exists one truth assignment $J$ that satisfies $\Gamma$. Let $\mathcal{I}$ be the interpretation such that:

- $A^T = \{ p \mid J(p) = \text{true} \}$
- for each $i \in \{1, \ldots, n\}$, $(R^+_i)^T = \{(d, q^T)\}$, where $q$ is a literal in $c_i$ such that $J(q) = \text{true}$
- for each $i \in \{1, \ldots, m\}$, $(R^-_i)^T = \{(d, q^T)\}$, where $\sim q$ is a literal in $c_i$ such that $J(q) = \text{false}$

It is easy to see that $\mathcal{I}$ is a model for $\Psi(\Gamma)$.

\[ \square \]

**Lemma 5.4.12** Concept Satisfiability in $\mathcal{ALC}$ is coNP-hard.

**Proof.** Follows from Lemma 5.4.11 and the fact that the reduction $\Psi$ is clearly polynomial with respect to the size of $\Gamma$.

We now show that Concept Satisfiability in $\mathcal{ALC}$ is in NP. This result is achieved showing that the algorithm obtained by the application of the completion rule runs in nondeterministic polynomial time.

The first step is to show that, for any $\mathcal{ALC}$-concept $C$, each completion of the constraint system $\{x : C\}$ has polynomial size. In [SS91] it is shown that for every $\mathcal{AL}$-concept $C$, the constraint system $\{x : C\}$ has the unique completion (up to variable renaming), which has linear size w.r.t. $|C|$. However, in $\mathcal{ALC}$, the size of the completion can be bigger, as shown by the following example:

$$C = \{a\} \cap \exists R \cap \forall R, \{a\} \cap \forall R, \forall R, \cdots \forall R, A$$

The only completion of the constraint system $\{x : C\}$ has quadratic size w.r.t. the length $n$ of the chain of universal quantifications $\forall R, \forall R, \cdots \forall R, A$. In fact, such a completion contains the constraints $a : \forall R, \forall R, \cdots \forall R, A, \ldots, a : \forall R, A$, $a : A$ for any size of the chain from 1 to $N$.

The example shows that the introduction of constraints on the individuals can increase the size of the single completion. Nevertheless, we show that its size is still polynomial. In particular, given the constraint system $\{x : C\}$, we associate with it a constraint system called $S_C$ and we show the two following properties: (i) the completion of $S_C$ is polynomial with respect to $|C|$ and (ii)
the completion of \( S_C \) is bigger that the completion of the constraint system \( \{x : C\} \).

**Definition 5.4.13** Given a constraint system \( \{x : C\} \), we call \( S_C \) the constraint system obtained adding to \( \{x : C\} \) all the constraints \( a : E \), for every individual \( a \) in \( C \) and every subconcept \( E \) of \( C \) (including \( C \) itself). If \( C \) does not contain any individual then \( S_C = \{x : C\} \).

Since both the number of subconcept of \( C \) and the number of individual in \( C \) are linear w.r.t. \(|C|\) it follows that \( S_C \) has a quadratic number of constraints w.r.t. \(|C|\). Therefore the size of \( S_C \) is almost cubic w.r.t. \(|C|\). Since \( S_C \) contains all the constraints of the form \( a : E \), the application of the \( \rightarrow \sigma \)-rule to a constraint system \( S \) obtained from \( S_C \) does not add any new constraint to \( S \). Therefore \( S_C \) has a single completion \( S' \). Now, it is easy to see that each completion of \( \{x : C\} \) is smaller than \( S' \); in fact \( S_C \) contains at least the constraint \( x : C \).

We now show that \( S' \) is polynomial w.r.t. \( S_C \) (and therefore w.r.t. \( C \)). To this aim, in Chapter 4, it is shown that for every knowledge base \( \Sigma \) in \( \mathcal{ALC} \), the unique completion of the constraint system \( S_\Sigma \), has polynomial size w.r.t. \(|\Sigma|\). Since the \( \rightarrow \sigma \)-rule does not add any constraint to \( S_C \), it follows that the completion of \( S_C \) is equal to the completion obtained for an \( \mathcal{ALC} \) constraint system. Therefore from the above results concerning \( \mathcal{ALC} \), it follows that the completion of \( S_C \) has polynomial size. Hence every completion of \( \{x : C\} \) has polynomial size too.

The fact that each completion has polynomial size ensures that it can be computed in polynomial time. In fact, the application of each rule either increase the size of the constraint system or increase the number of constraints on the individuals (the \( \rightarrow \sigma \)-rule). Since the number of constraints on the individuals is obviously bound by the size of the constraint system it follows that only a polynomial number of applications of the rules can be done. Since each completion is obtained doing a polynomial guess in the application of the \( \rightarrow \sigma \)-rule, it follows that the whole algorithm works in nondeterministic polynomial time. Therefore we have proved the following lemma:

**Lemma 5.4.14** Concept Satisfiability in \( \mathcal{ALC} \) is an NP problem.

**Theorem 5.4.15** Subsumption in \( \mathcal{ALC} \) is coNP-complete.

**Proof.** The NP-hardness follows from Lemma 5.4.12. Regarding the upper bound, using the technique in [DLNN91a, Sec. 5], Lemma 5.4.14 can be easily extended to state that Subsumption in \( \mathcal{ALC} \) is in NP too.

In [LS91a], it is observed that it is not necessary to have a constructor for name negation for intractability, but it suffices to have the possibility to
express disjointness between concepts. Therefore the intractability is directly extended to several other languages. For example, it applies to CLASSIC, which extend $\mathcal{FL}^-$ in several ways including $\mathcal{O}$ and $\mathcal{N}$. In fact using $\mathcal{N}$, it is possible to express the following two concepts which are obviously disjoint: $(\leq 2R)$ and $(\geq 3R)$.

On the other hand, if the underlying language does not include any form of disjointness, reasoning with $\mathcal{O}$ is in general polynomial. For example, in [FMV90], it is shown that in the language $\mathcal{OOL}$, which extends $\mathcal{FL}^-$, Subsumption can be checked in polynomial time.

Table 5.1 summarize the results obtained in this section together with previous known results on the underlying languages. In the table, each entry means that the problem is complete for the given class, except for P that has the simple meaning that the problem is in the class P.

### Table 5.1: Complexity of reasoning in mixed languages

<table>
<thead>
<tr>
<th></th>
<th>without $\mathcal{O}$</th>
<th>with $\mathcal{O}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}$</td>
<td>Subsumption</td>
<td>instance checking</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>$\mathcal{P}$ [SS91]</td>
<td>$\mathcal{P}$ (Chapter 4)</td>
</tr>
<tr>
<td>$\mathcal{ALC}$</td>
<td>$\mathcal{ALC}$</td>
<td>$\mathcal{ALC}$</td>
</tr>
<tr>
<td>$\mathcal{ALC}^+$</td>
<td>$\mathcal{NP}$ [DHL+92]</td>
<td>$\mathcal{NP}$ (Chapter 4)</td>
</tr>
<tr>
<td>$\mathcal{ALE}$</td>
<td>$\mathcal{NP}$</td>
<td>$\mathcal{PSPACE}$</td>
</tr>
<tr>
<td>$\mathcal{AL}$</td>
<td>$\mathcal{PSPACE}$ [SS91]</td>
<td>$\mathcal{PSPACE}$ [BH91b]</td>
</tr>
</tbody>
</table>

5.5 Discussion

We have shown an extended analysis of the various issues related to the use of concept constructors involving individuals. This analysis gives an insight of the problem of reasoning with individuals and allows to understand the intuitive aspects which makes reasoning difficult.

In addition, in Chapter 3 we have extended the calculus for reasoning with $\mathcal{O}$ and $\mathcal{B}$.

Another result of this chapter are a set of complexity results which formally confirm that reasoning with individuals is generally hard. In fact, in some languages, they increase the complexity of reasoning ($\mathcal{ALC,ALE}$). Whereas, in those cases in which reasoning is in the same complexity class as the underlying language ($\mathcal{ALC}$), the algorithms are generally more complex and less efficient.
(in terms of both time and space) than in the underlying language.

We have also identified an intuitive explanation of this intractability: On one side, it is related to the implicit disjunction carried by the use of sets with more than one object. On the other side, it is due to the implicit equality associated with individuals in concept expressions.

In our opinion, the solutions proposed in actual systems to overcome the computational intractability are not completely satisfying. Therefore a deeper insight of the problem can also be useful for the development of better incomplete reasoners and deviant semantics.
Chapter 6

Generalized Instance Checking

The results of Chapters 3, 4, and 5 show that tractability of reasoning is achieved only using limited languages, such as $\mathcal{AL}$ and $\mathcal{ALN}$. These languages are generally considered too poor to be useful in practical cases.

As argued also in [BN92], the developers of actual systems reacted to this situation in three different ways: There are systems, such as CLASSIC ([BBMA89a, BP94]) that support only a very limited language, but employ almost (see Section 5.2) complete reasoning methods. Systems such as BACK [QK90] and LOOM [MB87] provide a powerful language, but the reasoning is incomplete, which means that not all the existing subsumption relationships are detected. The only system that does not make this compromise, i.e., that provides complete algorithms for a very expressive language, is KRIS [BH91b]. On the other hand, a query in KRIS may take exponential time.

The three approaches described above are all reasonable, and they seem to be the only possible ways to overcome the intractability of reasoning in concept languages. However before committing ourselves to one or the other of those, we enquire whether it is still possible to gain some expressiveness giving up neither the tractability nor the completeness of reasoning.

To this aim, in Section 6.1 we propose a way to gain expressiveness making use of two different concept languages, one to express the knowledge base and one to formulate the queries. In Section 6.2, using the same idea, we introduce an epistemic operator in the query language, which allows us to go beyond the first order setting of concept language. Such operator, not only increases the expressivity without substantially increasing the complexity of reasoning but also helps in solving some semantic issues, in particular concerning the semantics of the existential quantification.

The results of Section 6.1 come from [LS91a, LS91b], whereas those of
Section 6.2 are taken from [DLN+92, DLN+93, DLN+94].

6.1 Enriching the Query Language

Our goal in this section is to explore the possibility to develop the query language without substantially increasing the complexity of reasoning.

The main idea is that, regarding the problem of checking if \( \Sigma \models D(a) \), we can allow the knowledge base \( \Sigma \) and the query \( D \) to be expressed in two different languages. We call assertional language the language used to express \( \Sigma \) and query language the language used for \( D \).

It is obvious that the query language should be at least as expressive as the assertional language, otherwise we couldn’t be able even to extract from the knowledge base the information that has been asserted.

We consider a query language that is more powerful than the assertional language, and we try to identifying an optimal compromise between expressive power and computational tractability based on both the assertional and the query language.

The main result of this section is to show that one can use a rich (intractable) query language without falling from the computational cliff, provided that a tractable language is used for expressing the knowledge base.

It is worth mentioning that the idea of using a query language richer than the assertional language is not new. For example, relational databases, which are built up by means of a very limited data definition language, are queried using a full first order language, called relational calculus.

In order to apply this idea in the context of concept-based knowledge bases, we make use of \( \mathcal{AL} \) for defining the knowledge base, and of \( \mathcal{ALE\!RHO} \) (in a limited form, see below) for query formulation. For the sake of simplicity, from this point on, \( \mathcal{ALE\!RHO} \) is called \( \mathcal{QL} \).

Another result of our work is that \( \mathcal{QL} \) is almost optimal with respect to the tractability of query answering. In particular, by analyzing the other constructors, we show that, if one aims at retaining tractability, none of them could be added to \( \mathcal{QL} \). We have therefore shown that, like for the assertional language, there are inherent limits to the expressive power of the query language.

6.1.1 Subsumption between Concepts of Different Languages

The first result that we achieved is that, when using a tractable language for the subsumee, it is possible to enrich the language of the subsumer without endangering the tractability of the Subsumption problem. To the best of our knowledge, this fact was never noticed before in the research on terminological reasoning. In particular, we study Pure Subsumption in the hypothesis that
\[
x:\text{\textsc{Child}.Graduated}\cap \exists \text{\textsc{Child}}\land \forall \text{\textsc{Child}.Graduated},
\]
\[
x:\text{\textsc{Child}.Graduated}\cap \exists \text{\textsc{Child}}, x:\text{\textsc{Child}.Graduated}, (\text{by } \neg \land)
\]
\[
x: \exists \text{\textsc{Child}}, x:\text{\textsc{Child}.Graduated}, (\text{by } \rightarrow \land)
\]
\[
x: y: \text{Graduated}, (\text{by } \neg \lor)
\]
\[
y: \neg \text{Graduated} (\text{by } \neg \lor)
\]

Figure 6.1: The completion of the constraint system of Example 6.1.1

the candidate subsumee \( C \) is an \( \text{AL} \)-concept, and the candidate subsumer \( D \) is a \( Q\text{AL} \)-concept.

Reminding that a concept \( C \) is subsumed by \( D \) if and only if \( C \cap \neg D \) is unsatisfiable, we can reduce Subsumption between a \( Q\text{AL} \)-concept \( D \) and an \( \text{AL} \)-concept \( C \) to unsatisfiability of the constraint system \( \{ x : C \cap \neg D \} \).

The set of completion rules for a constraint system of the above form is constituted by the rules for \( \text{AL} \) (i.e., \( \rightarrow \land, \rightarrow \lor \), and \( \rightarrow \exists \)), together with the rules \( \rightarrow \land D, \rightarrow \lor D \), and \( \rightarrow \exists D \), that take care of the constructors of \( \neg D \) (rewritten in simple form). Notice that the \( \rightarrow \exists \)-rule is never applied since the constraint system can contain only constraints of the form \( x : \neg\{a_1, \ldots, a_n\} \) and not of the form \( x : \{a_1, \ldots, a_n\} \).

Due to the presence of the \( \rightarrow \lor \)-rule, several complete constraint systems can be obtained from \( \{ x : C \cap \neg D \} \).

It is easy to see that, starting from \( \{ x : C \cap \neg D \} \), in a finite number of applications of the rules, all the completions are computed, and easily checked for clash. It follows that the above completion rules provide an effective procedure to check Subsumption between \( D \) and \( C \). Our task is now to show that such procedure works in polynomial time.

**Example 6.1.1** Let \( D \) be the following \( Q\text{AL} \)-concept \( \exists \text{\textsc{Child}.Graduated} \) and \( C \) be the \( \text{AL} \)-concept \( \forall \text{\textsc{Child}.Graduated} \cap \exists \text{\textsc{Child}} \). It is easy to verify that \( C \) is subsumed by \( D \). The complete constraint system obtained from \( S_1 = \{ x : C \cap \neg D \} \) is shown in Figure 6.1. \( \square \)

Theorem 6.1.3 states that the procedure based on constraint systems works in polynomial time. In order to prove it, we need the following lemma.

**Lemma 6.1.2** Given a constraint system \( S = \{ x : C \cap \neg D \} \), where \( C \) is an \( \text{AL} \)-concept and \( D \) is a \( Q\text{AL} \)-concept, let \( S' \) be a constraint system obtained from \( S \) by the application of the completion rules and let \( E \) be a subconcept of \( \neg D \) of the form \( \exists R.E' \) or \( \forall R.E' \). For each variable \( y \) in \( S' \) we have the following properties:
1. at most one constraint of the form $y : E$ is in $S'$;

2. for no other variable $z$, such that $y$ and $z$ are direct successors of the same variable, there exists a constraint of the form $z : E$ in $S'$.

Proof. We prove the results by induction showing that it holds for the root $x$ and if it holds for a variable, it holds for every direct successor if it. Since $\mathcal{QL}$ contains no disjunction, the simple concept obtained rewriting $\neg D$ contains no conjunction. It follows that the $\rightarrow_\exists$-rule never applies to the constraint $x : \neg D$. Therefore, only the $\rightarrow_\forall$-rule can generate new constraints on $x$ involving subconcepts of $\neg D$. However, a chain of zero or more applications of the $\rightarrow_\forall$-rule can generate only one constraint which is not of the form $D' \subseteq D''$, proving the claim 1 for $x$. The claim 2 trivially holds for $x$ since it has no direct predecessor. Suppose that claims 1 and 2 hold for $y$. There are two cases:

1. a constraint of the form $y : \exists R.E'$ is in $S'$. Since there are no constraints of the form $y : \forall R.E'$ in $S'$, the variable $z$ created by the application of the $\rightarrow_\exists$ to the constraint $y : \exists R.E'$ is involved only in the constraint $z : E'$ (along with constraint involving subconcepts of $C$) (proving claim 1). In addition, even though $y$ can have others direct successor, since $E'$ is inside an existential quantification it is not propagated to any of them (proving claim 2).

2. a constraint of the form $y : \forall R.E'$ is in $S'$. Since there are no constraint of the form $y : \exists R.E'$ in $S'$, only constraints of the form $\exists Q$ can be in $S'$. Since the existential quantification is unqualified, even if there are more than one constraint of that form involving $R$ (e.g. $\exists (P \cap R)$), only one variable $z$ is created and the constraint $z : E'$ is transmitted to it (proving claim 1 and claim 2).

\begin{theorem}
Let $C$ be an $\mathcal{AL}$-concept, and let $D$ be a $\mathcal{QL}$-concept. Then the set of all the completions of the constraint system \{x : C \cap \neg D\} can be computed in polynomial time with respect to $|C \cap \neg D|$. 
\end{theorem}

Proof. Since both the selection and the application of the completion rules can be done in polynomial time (Proposition 3.1.3), it is sufficient to show that (i) every completion has a size which is polynomially bounded by $|C \cap \neg D|$, and (ii) the number of completions obtainable from $x : C \cap \neg D$ is bounded by $|C \cap \neg D|$.

With regard to point (i), it is sufficient to show that the number of variables in any completion $S'$ is bounded by $|C \cap \neg D|$. Lemma 6.1.2 ensures that, among all the variables in $S'$, there is only a chain of variables that involves
subconcepts of $\neg D$. In addition, the length of the chain is linear in the size of $\neg D$. Therefore, since the number of variable generated from $\{x : C\}$ is polynomial, the claim follows.

With regard to the point $(ii)$, since for each concept $E$ appearing in $D$, only one variable $x$ may exist such that $x : E$ is in $S'$, it follows that number of complete constraint systems that can be obtained from $\{x : C \cap \neg D\}$ is bounded by the number of occurrences of the symbol $\cap$ in $D$, and therefore is bounded by $|C \cap \neg D|$.

From the above theorem it follows that checking Subsumption between a $\mathcal{QL}$-concept and an $\mathcal{AL}$-concept can be done in polynomial time. This result will be exploited in next section to devise an Instance Checking procedure.

### 6.1.2 Instance Checking using the Enriched Query Language

In this section we propose a query answering method that allows one to pose queries using the language $\mathcal{QL}$ to a knowledge base in $\mathcal{AL}$.

Since we aim at the tractability of query answering, we need to impose a certain restriction on the $\mathcal{QL}$-concepts used for formulating the queries. Such restriction is as follows\(^1\): A $\mathcal{QL}$-concept $Q$ is said to be safe if for every subconcept of $Q$ of the form $\exists R. D$ that is not in the scope of a universal quantification, $D$ does not contain any existential quantification.

The above restriction, that is further discussed in Section 6.1.3, allows us to devise a polynomial algorithm for query answering. Throughout this section we assume to deal with queries represented by safe $\mathcal{QL}$-concepts.

**Example 6.1.4** In order to show the expressive power of our query language, we present some queries that can be formulated using $\mathcal{QL}$.

All the individuals having at least one child that works only in the factory and who is susan or peter or mary:

$$\exists \text{CHILD}.(\{\text{susan, peter, mary}\} \uparrow \forall \text{WORKSIN.\{factory\}}$$

All the individuals having a graduated grandchild:

$$\exists (\text{CHILD} \circ \text{CHILD}). \text{Graduated}$$

Is peter or john a bad father (i.e., someone that is not friend of any of his children) ?:

$$\{\text{peter, john}\} \uparrow \forall (\text{CHILD} \uparrow \text{FRIEND})$$

Notice that a method merely based on Abstraction/Subsumption would not work in our case, because of the presence of the qualified existential quantification in $\mathcal{QL}$. For example, in order to answer the query $\Sigma \models \exists P_1 \circ$

\(^1\)This definition of safe is given in [MM93], and it corrects the previous one given in [LS91b]
it is not sufficient to look at \( CC_{\Sigma^2} \), but it is necessary to consider the assertions involving the roles \( P_1 \) and \( P_2 \) in the knowledge base. For this reason, our method relies on an ad hoc technique that, by navigating through the role assertions, takes into account the whole knowledge about the individuals.

We present now an ad hoc technique to solve the problem, which has been developed in [LS91a] and has been revised and corrected in [LS91b]. We prove its correctness and its tractability. The technique has been also used in [MM93] to solve problems more complex than Instance Checking (i.e. the Retrieval Problem).

Such technique is only partially based on the calculus. A procedure completely based on the calculus, i.e. check the satisfiability of the constraint system \( S_{\Sigma} \cup \{ \alpha : \neg C \} \), could be used as well. The correctness of the procedure would be ensured by the correctness of the calculus. However proving its tractability seems to be more complex than in our case.

Notice that if a knowledge base \( \Sigma \) in \( \mathcal{AL} \) is unsatisfiable, then any query to \( \Sigma \) gets a positive answer. As we said in Section 4.1, checking the satisfiability of \( \Sigma \) can be done in polynomial time by computing the (unique) completion of \( S_{\Sigma} \).

In the sequel we make use of a function \( \text{ALL} \) that, given an object \( a \), a \( \mathcal{QL} \)-role \( Q \) of the form \( P_1 \sqcap \ldots \sqcap P_n \), and a knowledge base \( \Sigma \) in \( \mathcal{AL} \), computes the concept \( \text{ALL}(a, Q, \Sigma) = C_1 \sqcap \ldots \sqcap C_m \), where \( C_1, \ldots, C_m \) are all the concepts appearing in some constraint of the form \( \alpha : \forall P.C_i \) in the completion of \( S_{\Sigma} \) such that \( P \in \{ P_1, \ldots, P_n \} \). If no such a concept exists, we assume \( \text{ALL}(a, Q, \Sigma) = \top \). In other words, \( \text{ALL}(a, Q, \Sigma) \) represents the concept to which every object related to \( a \) through \( Q \) must belong, according to the assertions in \( \Sigma \).

Our method heavily relies on the following theorem, which states necessary and sufficient conditions for an assertion to be logically implied by a knowledge base.

**Theorem 6.1.5** Let \( \Sigma \) be a satisfiable knowledge base in \( \mathcal{AL} \) and \( S \) the completion of \( S_{\Sigma} \); let \( a, a_1, \ldots, a_n \) be individuals, \( A \) be a concept name, \( R \) be a \( \mathcal{QL} \)-role, \( Q \) be a \( \mathcal{QL} \)-role of the form \( P_1 \sqcap \ldots \sqcap P_m \), and \( D_1, D_2 \) be \( \mathcal{QL} \)-concepts. Then the following properties hold:

1. \( \Sigma \models \{ a_1, \ldots, a_n \}(a) \) if and only if \( a \in \{ a_1, \ldots, a_n \} \);
2. \( \Sigma \models A(a) \) if and only if \( a : A \in S \) and \( \Sigma \models \neg A(a) \) if and only if \( a : \neg A \in S \);
3. \( \Sigma \models D_1 \sqcap D_2(a) \) if and only if \( \Sigma \models D_1(a) \) and \( \Sigma \models D_2(a) \);
4a. \( \Sigma \models \forall Q.D(a) \) if and only if \( D \) subsumes \( \text{ALL}(a, Q, \Sigma) \).
4b. \( \Sigma \models \forall (Q \circ R). D(a) \) if and only if \( \forall R. D \) subsumes \( ALL(a, Q, \Sigma) \);

5. \( \Sigma \models \exists R. D(a) \) if and only if there is a \( b \) such that
\( aRb \) holds in \( S \) and \( \Sigma \models D(b) \)

Proof. The proofs of 1, 2 and 3 are straightforward. With regard to 4a, assume that \( D \) subsumes \( ALL(a, Q, \Sigma) \), and suppose that \( \Sigma \not\models \forall Q. D(a) \), i.e. \( \Sigma \cup \{ \exists Q. \neg D(a) \} \) is satisfiable. This implies that there is a model \( T \) of \( \Sigma \) with an element \( d \in \Delta^T \) such that \( d \) is related to \( a \) by means of \( Q \), and \( d \in (\neg D)^T \); but, based on the definition of \( ALL \), it follows that \( d \in (ALL(a, Q, \Sigma))^T \), contradicting the hypothesis that \( ALL(a, Q, \Sigma) \) is subsumed by \( D \). On the other hand, assume that \( \Sigma \models \forall Q. D(a) \), i.e. \( \Sigma \cup \{ \exists Q. \neg D(a) \} \) is unsatisfiable, implying that \( S_\Sigma \cup \{ aQz, z : \neg D \} \) is unsatisfiable, where \( z \) is a new variable. Note that \( S_\Sigma \cup \{ aQz, z : \neg D \} \) is unsatisfiable if and only if \( S_\Sigma \cup \{ aQz, z : \neg D, z : ALL(a, Q, \Sigma) \} \) is unsatisfiable. Now, it is possible to verify that, since \( \Sigma \) is satisfiable and \( z \) do not appear in \( \Sigma \), this may happen only because the constraint system \( \{ z : ALL(a, Q, \Sigma), z : \neg D \} \) is unsatisfiable, which means that \( D \) subsumes \( ALL(a, Q, \Sigma) \). Similar arguments can be used for the proof of 4b.

With regard to 5, it is easy to verify that if there is a \( b \) such that \( aRb \) holds in \( S \) and \( \Sigma \models D(b) \), then \( \Sigma \models \exists R. D(a) \). On the other hand, assume that \( \Sigma \models \exists R. D(a) \), and suppose that for no \( b_i \) \((i = 1, \ldots, n) \) such that \( aRb_i \) holds in \( S \), \( \Sigma \models D(b_i) \). This implies that for each \( b_i \), \( \Sigma \cup \{ \neg D(b_i) \} \) is satisfiable. Now it is possible to prove that there exist \( n \) interpretations \( M_1, \ldots, M_n \) such that for each \( i \in \{1, \ldots, n\} \), \( M_i \) is a model of \( \Sigma \cup \{ \neg D(b_i) \} \), and \( M_1 \cup \cdots \cup M_n \) is a model of \( \Sigma \cup \{ \forall R. \neg D(a) \} \), contradicting the hypothesis that \( \Sigma \models \exists R. D(a) \).

It is worth noting that the above theorem would be no longer valid if we include the full existential quantification in the query language. In fact, consider a query of the form \( \exists R.C \) and an object \( a \) and suppose that the completion of \( S_\Sigma \) includes for \( a \) only the two following constraints regarding \( R \): \( aRb_1 \) and \( aRb_2 \). In addition suppose that \( \Sigma \cup \{ \neg C(b_1) \} \) and \( \Sigma \cup \{ \neg C(b_2) \} \) are both satisfiable. In this case it may happen that \( \Sigma \cup \{ \neg C(b_2) \} \cup \{ \neg C(b_1) \} \) is unsatisfiable. For example, if \( C \) as the form \( (\exists R.A) \land \neg A \) and \( \Sigma \) includes \( R(b_1, b_2) \), in all the models of \( \Sigma \cup \{ \neg C(b_1) \} \), all the objects related to \( b_1 \) by means of the role \( R \) (including \( b_2 \)) are forced to be in the extension of \( \neg A \). Therefore, in this case, it is not possible to find a model of \( \Sigma \cup \{ \neg C(b_2) \} \cup \{ \neg C(b_1) \} \), and thus a model of \( \Sigma \cup \{ \forall R. \neg C(a) \} \), as stated in point 5 of the theorem.

Based on the properties stated in the above theorem, we can directly develop a sound and complete algorithm for query answering. The algorithm is called \( \text{Inst}_{AC/\mathcal{L}} \) and is shown in Figure 6.2, in which \( \text{Comp}(\Sigma) \) denotes the completion of \( S_\Sigma \). The following theorem states the soundness and the
Algorithm $\text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D)$

**Input** knowledge base $\Sigma$ in $\mathcal{AC}$, object $a$, $\mathcal{QL}$-concept $D$;

**Output** one value in $\{true, false\}$;

begin

case $D$ of

   \{\text{a}_1, \ldots, \text{a}_n\} : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := a \in \{\text{a}_1, \ldots, \text{a}_n\}

   A : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := a : A \in \text{COMP}(\Sigma);

   \neg A : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := a : \neg A \in \text{COMP}(\Sigma);

   D_1 \cap D_2 : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := \text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D_1) \land \text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D_2);

   \forall Q . D_1 : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := D_1 \text{ subsumes ALL}(a, Q, \Sigma);

   \forall (Q \circ R) . D_1 : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := \forall R . D_1 \text{ subsumes ALL}(a, Q, \Sigma);

   \exists R . D_1 : \text{Inst}_{\mathcal{AC}/\mathcal{QL}} := \exists b \text{ such that } (aRb \text{ holds in } \text{COMP}(\Sigma)) \land \text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, b, D_1);

caseend

end

Figure 6.2: The algorithm for Instance Checking in $\mathcal{AC}/\mathcal{QL}$

The completeness of the algorithm. Moreover, it shows that the time complexity of the algorithm is polynomial.

**Theorem 6.1.6** Let $\Sigma$ be a satisfiable knowledge base in $\mathcal{AC}$, $a$ be an individual, and $D$ be a $\mathcal{QL}$-concept. Then $\text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D)$ terminates, returning true if $\Sigma \models D(a)$, and false otherwise. Moreover, it runs in polynomial time with respect to $|\Sigma|$ and $|D|$.

**Proof.** The correctness of the algorithm easily follows from Theorem 4.1. With respect to termination it is sufficient to observe that in any recursive call of the algorithm the actual parameter corresponding to $D$ decreases in length. With respect to complexity, notice first of all that the number of recursive calls that are issued during $\text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D)$ is linear with respect to $|D|$. Moreover both the subsumption check between $D$ and $\text{ALL}(a, Q, \Sigma)$, and the check if $aRb$ holds in $\Sigma$ are polynomial with respect of the size of $\Sigma$ and $D$. It follows that any recursive call issued during the execution of $\text{Inst}_{\mathcal{AC}/\mathcal{QL}}(\Sigma, a, D)$ performs a number of operations bounded by the size of $D$ and the size of $\text{COMP}(\Sigma)$, which is polynomially related to $|\Sigma|$.
6.1.3 Limits to the Tractability of the Query

In this section we consider some possible extensions of the query language and analyze their effect on the tractability of query answering.

The first observation regards the possibility of using the full power of qualified existential quantification. In [LS91a], we used the full language $\mathcal{QL}$ for query formulation, and we claimed that a polynomial algorithm similar to the one presented in Section 4 was sound and complete for query answering. Unfortunately, as pointed out in [LS91b], while that algorithm is sound, it is in fact not complete if the concept representing the query is not safe. The reason is that the proof of point 5 of Theorem 6.1.5 is no longer valid if we include nested qualified existential quantification in the query language. In fact, we can prove that query answering with full $\mathcal{QL}$ is co-NP-hard.

Such result easily follows from Lemma 4.2.7. In fact, the reduction shows that the coNP-hardness arises even if the knowledge base is expressed using a simple language, even simpler than $\mathcal{AL}$. This implies that, in order to obtain tractability, it is sufficient to enrich only the query language with the qualified existential quantification, keeping a simple and tractable assertional language. Moreover the intractability arise even considering the data complexity. It is worthwhile to notice that the algorithm proposed in [LS91a] fails to get the correct answer YES for the query $\beta$ to the knowledge base $\Sigma$ of Example 4.2.5.

The second observation is that if $C$ is equivalent to $\top$, then for any knowledge base $\Sigma$, it holds that $\Sigma \models C(a)$. It follows that query answering is at least as hard as the so-called top-checking problem for the query language, i.e. checking whether a concept is equivalent to the universal concept $\top$.

In some language, top-checking is intractable. We show in the following that this is the case already for $\mathcal{FLU}^-$, that is obtained from $\mathcal{FL}^-$ simply by adding disjunction of concepts.

The proof is based on a reduction from the satisfiability problem for a propositional conjunctive normal form (CNF) formula and the top-checking problem in $\mathcal{FLU}^-$. Let $\Phi$ be the following transformation from a propositional CNF formula $\Gamma = \alpha_1 \land \cdots \land \alpha_m$ to an $\mathcal{FLU}^-$-concept $\Phi(\Gamma)$ ($p_i$ denotes a propositional letter, $l_i$ a literal, and $\alpha_i$ a clause):

\[
\begin{align*}
\Phi(p_i) &= \exists R_{p_i} \\
\Phi(\neg p_i) &= \forall R_{p_i} \cdot \neg A \\
\Phi(l_i \lor \cdots \lor l_n) &= \Phi(l_i) \lor \cdots \lor \Phi(l_n) \\
\Phi(\alpha_1 \land \cdots \land \alpha_m) &= \Phi(\alpha_1) \land \cdots \land \Phi(\alpha_m)
\end{align*}
\]

For example if $\Gamma = (\neg p \lor q) \land (\neg q \lor r)$, then the corresponding $\mathcal{FLU}^-$-concept is $\Phi(\Gamma) = (\forall R_p \cdot A \land \exists R_q) \lor (\forall R_q \cdot A \land \exists R_r)$. 
Theorem 6.1.7 A propositional CNF formula $\Gamma$ is satisfiable if and only if the corresponding $\mathcal{FLU}^-$-concept $\Phi(\Gamma)$ is not equivalent to $\top$.

Proof. Assume that $\Gamma$ is satisfiable, and let $M$ be one of its models. Let $I$ be the interpretation for $\Phi(\Gamma)$ defined as follows:
- if $M(p_i) = \text{true}$ then $R_{p_i}^I = \emptyset$, 
- if $M(p_i) = \text{false}$ then $R_{p_i}^I = \{(d, d_{p_i})\}$ and $d_{p_i} \notin A^I$.

It follows that for every literal $l_k$ (positive or negative), if $M(l_k) = \text{true}$ then $d \notin \Phi(l_k)^I$ and if $M(l_k) = \text{false}$ then $d \in \Phi(l_k)^I$. Now, since in every clause $\alpha_j$ of $\Gamma$ there is a literal whose value in $M$ is true, it follows that in the corresponding concept $\Phi(\alpha_j)$ of $\Phi(\Gamma)$ there is at least one conjunct $\Phi(l_i)$ such that $d \notin \Phi(l_i)^I$. Therefore, for every $j$ we have that $d \notin (\Phi(\alpha_j))^I$ thus $d \notin (\Phi(\Gamma))^I$, and therefore $\Phi(\Gamma)$ is not universal.

On the other hand, assume that $\Phi(\Gamma)$ is not universal, and let $I$ be an interpretation with an element $d \in \Delta^I$ such that $d \notin (\Phi(\Gamma))^I$. Let $M$ be the truth assignment for $\Gamma$ defined as follows:
- $M(p_i) = \text{true}$ if $d \notin (\exists R_{p_i})^I$,
- $M(p_i) = \text{false}$ if $d \notin (\forall R_{p_i}, A)^I$.

Now, since for each $\alpha_j = (l_1 \vee \cdots \vee l_n), d \notin (\Phi(l_1 \vee \cdots \vee l_n))^I$, it follows that either there exists $h$ such that $l_h = p_i$ and $d \notin (\exists R_{p_i})^I$, or there exists $k$ such that $l_k = \neg p_i$ and $d \notin (\forall R_{p_i}, A)^I$. Therefore, the clause $\alpha_j$ is satisfied by $M$. Since this holds for every clause of $\Gamma$, we can conclude that $\Gamma$ is satisfiable. \qed

Theorem 6.1.8 Query answering using $\mathcal{FLU}^-$ as query language is co-NP-hard, independently of the assertional language.

Proof. Since the reduction $\Phi$ is clearly polynomial with respect to the size of $\Gamma$, the previous theorem shows that top-checking in $\mathcal{FLU}^-$ is co-NP-hard. The thesis follows from the fact that query answering is at least as hard as top-checking in the query language. \qed

Notice that the transformation $\Phi$ does not exploit the full power of $\mathcal{FLU}^-$, in fact the $\sqcup$ construct is used only at the upper level of the query. Indeed, the above theorem shows that answering queries of the form $C_1 \sqcup \cdots \sqcup C_n$, where each $C_i$ is an $\mathcal{FL}^-$-concept, is co-NP-hard, independently of the assertional language.

The above result allows us to derive the intractability of several other concept languages as query languages. For example, the co-NP-hardness clearly extends to both $\mathcal{ALU}$ and $\mathcal{ALC}$.

An analogous result can be achieved for the language $\mathcal{FL}$ [BL84], which extends $\mathcal{FL}^-$ with the construct $R : C$ for role restriction\(^2\). To see why, notice

\(^2\)The $R : C$ construct has the following semantics: $(R : C)^I = \{(a, b) \in \Delta^2 \mid (a, b) \in R^I \land b \in C^I\}$. 
that in \( \mathcal{FL} \) a concept of the form \( \forall (R : C) . D \) is equivalent to \( \top \) if and only if \( C \) is subsumed by \( D \). It follows that top-checking in \( \mathcal{FL} \) is at least as hard as Subsumption, which is a co-NP-hard problem.

Notice that all the intractability results obtained by the reduction to the top-checking problem concern with the combined complexity (and the query complexity). In fact, the knowledge base plays no role in the top-checking.

If we use the knowledge base complexity, we might get different results. For example, as a consequence of the results of Section 4.2.3, we have that if \( \Sigma \) is a knowledge base expressed in the language \( \mathcal{ALC} \), \( a \) is an individual, and \( D \) is an \( \mathcal{ALC} \)-concept, then checking if \( \Sigma \models D(a) \) can be done in polynomial time with respect to \( |\Sigma| \). Indeed, in this case, checking whether \( C_{\Sigma|a} \cap \neg D \) is unsatisfiable, can be done in polynomial time with respect to \( |C_{\Sigma|a} \cap \neg D| \) (although in exponential time with respect to \( |C_{\Sigma|a} \cap \neg D| \)). In other words, when \( \Sigma \) is expressed in \( \mathcal{ALC} \), Instance Checking using \( \mathcal{ALC} \) is still coNP-hard with respect to combined complexity, but is polynomial with respect to knowledge base complexity.

### 6.1.4 Discussion

We have considered several constructors and we have seen that none of them can be used in the query language without sacrificing tractability. This analysis does not cover the set of all the constructors considered in Chapter 2. For example, inverse roles and number restrictions have not been taken into account. This issue will be addressed in future work.

We have shown that it is possible to use a rich language for querying a concept-based knowledge base while keeping the deduction process still tractable. It is interesting to observe that this is one of the few “positive” results in the recent research on terminological reasoning, after a number of intractability results on reasoning about concepts.

In the future, we aim at addressing several open problems related to the use of concept languages as query languages. First of all, we aim at investigating the possible extensions of \( \mathcal{QL} \) which have been discussed above (number restrictions and inverse roles).

Second, we would like to investigate possible extensions of \( \mathcal{QL} \) that are still tractable w.r.t. the knowledge base complexity.

Finally, we aim at improving the efficiency of our method for query answering; in fact, the goal of our work was to show that the problem is tractable, but several optimization of the algorithm are needed in order to cope with sizable knowledge bases (e.g., [MM93]).
6.2 The Epistemic Operator

The strength of concept languages is that they are given a set-theoretic first-order semantics. However, the first-order semantics is, for other aspects, also their weakness; in fact it leaves out several aspects of practical systems. Therefore, it seems now appropriate to enrich such semantics both to explore novel language features and to account for some of those aspects that cannot be described in a standard first-order framework.

The need of some enrichment of this sort is discussed in the literature (see for example [DP91, Woo91]) and can be easily recognized by looking at recent concept-based systems such as CLASSIC and CLASP [YNM91]. Work in this direction has already begun with proposals of extending concept languages to deal with different forms of non-monotonic reasoning (see for example [BH92, QR92, PN93, BH93, Str93, PZ93]).

We proposed to enrich concept languages with an epistemic operator defined in the style of [Lev84, Rei90, Li91]. We show that answering queries formulated in the epistemic concept languages can be done by extending the calculus presented in Chapter 3. We also aim at discussing in more detail the advantages provided by such an extension for enhancing the capabilities of concept languages. In particular, we focus our attention on the use of the epistemic operator in order to: (1) define a more powerful query language; (2) be able to formulate queries requiring some forms of closed world reasoning. In the next chapter, we also show the use of the epistemic operator to formalize procedural and non-monotonic mechanisms.

With regard to Point (1), in Chapter 8, we provide several examples that show how the new query language allows one to address both aspects of the external world as represented in the knowledge base, and aspects of what the knowledge base knows about the external world.

With regard to Point (2), we show that a careful usage of the epistemic operator allows one to express queries whose processing forces the system to assume complete knowledge about (part of) the knowledge base. Note that this approach is different from assigning a closed world semantics to the knowledge base itself. In fact, the world closure is not in the semantics of the knowledge base, but a form of closure reasoning is achieved by the system when answering special kinds of queries.

6.2.1 Semantics

In this section we present the epistemic concept language $\mathcal{ALCK}$ which is an extension of $\mathcal{ALC}$ with an epistemic operator. Generally speaking, we follow [Rei90], and use $\mathcal{KC}$ to denote the set of individuals known to be instances of the concept $C$, i.e. that are in the extension on $C$ in every model for the
knowledge base. The syntax of $\mathcal{ALCK}$ is the following:

$$
C, D \rightarrow A | T | \bot | C \cap D | C \cup D | \neg C | \forall R.C | \exists R.C | KC
$$

$$
R \rightarrow P | KP
$$

The semantics of $\mathcal{ALCK}$ is an adaptation to the framework of concept languages of the one proposed in [Lev84, Rei90, Lif91]. As in the cited papers, some issues typical of first-order modal systems arise. Such issues concern the interpretation structures and are dealt with by the following assumptions:

- every interpretation is defined over a fixed domain, called $\Delta$ (Common Domain Assumption);
- for every interpretation the mapping from the individuals into the domain elements, called $\gamma$, is fixed (Rigid Term Assumption).

An epistemic interpretation is a pair $(I, W)$ where $I$ is an interpretation and $W$ is a set of interpretations such that the following equations are satisfied:

$$
\begin{align*}
\tau^I_W &= \Delta \\
\bot^I_W &= \emptyset \\
A^I_W &= A^I \\
P^I_W &= P^I \\
(C \cap D)^I_W &= C^I_W \cap D^I_W \\
(C \cup D)^I_W &= C^I_W \cup D^I_W \\
(\neg C)^I_W &= \Delta \setminus C^I_W \\
(\forall R.C)^I_W &= \{d_1 \in \Delta \mid \forall d_2. (d_1, d_2) \in R^I_W \rightarrow d_2 \in C^I_W\} \\
(\exists R.C)^I_W &= \{d_1 \in \Delta \mid \exists d_2. (d_1, d_2) \in R^I_W \land d_2 \in C^I_W\} \\
(KC)^I_W &= \bigcap_{J \in W} (C^J_W) \\
(KP)^I_W &= \bigcap_{J \in W} (P^J_W).
\end{align*}
$$

Notice that, since the domain is fixed independently of the interpretation, it is meaningful to refer to the conjunction of the extensions of a concept in different interpretations. It follows that $KC$ is interpreted in $W$ as the set of objects that are instances of $C$ in every interpretation belonging to $W$. In this sense, $KC$ represents those objects known to be instances of $C$ in $W$. Notice also that if one discards $K$ and $W$ in the equations, one obtains the standard semantics of $\mathcal{ALC}$.
A knowledge base $\Psi$ in $\mathcal{ALCK}$ is defined in the usual way as a pair (TBox, ABox), whose concepts and roles belong to the language $\mathcal{ALCK}$. The truth of inclusion statements and membership assertions in an epistemic interpretation is defined in a straightforward way. An epistemic model for $\Psi$ is a pair $(I, W)$, where $I \in W$ and $W$ is any maximal set of interpretations such that for each $\mathcal{J} \in W$, every sentence of $\Psi$ is true in $(\mathcal{J}, W)$.

Notice that the semantics of a knowledge base in $\mathcal{ALCK}$ could be equivalently defined in terms of an accessibility relation on a set of possible worlds. More specifically, the constraints posed by the semantic equations on $\mathcal{KC}$ and $\mathcal{KP}$, correspond to a structure of possible worlds each one connected with all the others. Therefore, the accessibility relation would be an equivalence relation, as in the modal system S5. However, the epistemic models of a knowledge base correspond to S5 models with a maximal set of worlds. In particular, if $\Sigma$ is a knowledge base in $\mathcal{ALC}$ and we denote with $M(\Sigma)$ the set of its models, then its epistemic models are all the pairs $(I, M(\Sigma))$ for every $I \in M(\Sigma)$.

A knowledge base $\Psi$ in $\mathcal{ALCK}$ is said to be consistent if there exists an epistemic model for $\Psi$, inconsistent otherwise. $\Psi$ logically implies an assertion $\sigma$, written $\Psi \models \sigma$, if $\sigma$ is true in every epistemic model for $\Psi$.

### 6.2.2 The Calculus for Answering Epistemic Queries

In this section, we use $\mathcal{ALCK}$ as a query language to knowledge bases in $\mathcal{ALC}$. We thus consider Instance Checking $\Sigma \models C(a)$ where $\Sigma$ is a knowledge base in $\mathcal{ALC}$ and $C$ is an $\mathcal{ALCK}$-concept.

The problem of designing methods for epistemic query answering is dealt with in [Lev84, Rei90]. In [Rei90], a procedure is presented that is sound and complete if the query satisfies some syntactic constraints. However, not all epistemic concepts belonging to $\mathcal{ALCK}$ satisfy such constraints (for example, the formula corresponding to $\exists P, -KC(a)$ is not admissible in [Rei90]). On the other hand, the method proposed in [Lev84] has been conceived within a more general framework, and its specialization to the case of concept languages does not provide an effective procedure. It follows that none of these approaches can be directly applied in our setting.

One way to answer epistemic queries posed to a knowledge base $\Sigma$ in $\mathcal{ALC}$ is to check whether the knowledge base in $\mathcal{ALCK}$ obtained by adding to $\Sigma$ the negation of the query is unsatisfiable. In this section we extend the calculus presented in Chapter 3 to deal with this problem.

Suppose to define a constraint system in the same way as Chapter 3, with the difference that concept and roles are those of $\mathcal{ALCK}$. In order to assign a meaning to constraints of that kind, we need the following definitions. An assignment $\alpha(\cdot)$ is a function from $\mathcal{O} \cup \mathcal{V}$ into $\Delta$, such that $\alpha|_{\mathcal{O}} = \gamma$, i.e. $\alpha$ is an extension of $\gamma$ to variables.
6.2 The Epistemic Operator

Let \((\mathcal{I}, \mathcal{W})\) be an epistemic interpretation, and let \(\alpha\) be an assignment. The triple \((\mathcal{I}, \mathcal{W}, \alpha)\) is said to satisfy the constraint \(s : C\) if \(\alpha(s) \in C_{\mathcal{I}, \mathcal{W}}\). Similarly, \((\mathcal{I}, \mathcal{W}, \alpha)\) satisfies the constraint \(sRt\) if \((\alpha(s), \alpha(t)) \in R_{\mathcal{I}, \mathcal{W}}\).

Let \(S\) be a constraint system, and let \(\Sigma\) be a knowledge base in \(\mathcal{ACL}\). \((\mathcal{I}, \mathcal{W}, \alpha)\) is a solution of \(S\) if \((\mathcal{I}, \mathcal{W}, \alpha)\) satisfies all its constraints. \(S\) is said to be \(\Sigma\)-solvable if there is a triple \((\mathcal{I}, \mathcal{M}(\Sigma), \alpha)\) that is a solution of \(S\). If there is no such solution, then \(S\) is said to be \(\Sigma\)-unsolvable.

The next proposition shows that answering a query posed to a knowledge base \(\Sigma\) can be reduced to checking a particular constraint system for \(\Sigma\)-solvability.

**Proposition 6.2.1** Let \(\Sigma\) be a knowledge base in \(\mathcal{ACL}\), \(C\) be an \(\mathcal{ACLK}\)-concept, and \(a\) an an individual. Then \(\Sigma \models C(a)\) if and only if \(S_\Sigma \cup \{a : \neg C\}\) is \(\Sigma\)-unsolvable.

An \(\mathcal{ACLK}\)-concept is said to be simple if every negation appearing in it is either of the form \(\neg A\) or of the form \(\neg KC\), where \(C\) is simple. Every \(\mathcal{ACLK}\)-concept can be rewritten in linear time into an equivalent simple concept.

We say that \(sRt\) holds in a constraint system \(S\) if either

1. \(R\) is \(P\), and \(sPt \in S\), or

2. \(R\) is \(KP\), \(s, t \in \mathcal{O}\), and \(sPt \in S\).

The set of completion rules are the same defined in Chapter 3 for \(\mathcal{ACL}\). The following proposition is the analog of Proposition 3.1.2 in the case of epistemic concept (and specialized to \(\mathcal{ACL}\)).

**Proposition 6.2.2** Let \(\Sigma\) be a knowledge base in \(\mathcal{ACL}\), and \(S, S'\) be two constraint systems. Then:

1. If \(S'\) is obtained from \(S\) by application of one of rules \(\rightarrow\Pi, \rightarrow\exists, \rightarrow\forall\), then \(S\) is \(\Sigma\)-solvable if and only if \(S'\) is \(\Sigma\)-solvable.

2. If \(S'\) is obtained from \(S\) by application of the \(\rightarrow\Pi\)-rule, then \(S\) is \(\Sigma\)-solvable if \(S'\) is \(\Sigma\)-solvable. Conversely, if \(S\) is \(\Sigma\)-solvable and the \(\rightarrow\Pi\)-rule is applicable to \(S\), then it can be applied in such a way that it yields a \(\Sigma\)-solvable constraint system.

We now introduce the notion of \(\Sigma\)-clash. Then, we exploit such a notion in order to derive a necessary and sufficient condition for a constraint system to be \(\Sigma\)-solvable.

**Definition 6.2.3** A constraint system \(S\) contains a \(\Sigma\)-clash if at least one of the following conditions holds:
1. S contains a constraint of the form s: ⊥;
2. S contains two constraints of the form s: A, s: ¬A;
3. S contains a constraint of the form a: KC, and there is at least one completion of S_Σ ∪ \{a: ¬C\} without Σ-clashes;
4. S contains a constraint of the form a: ¬KC, and every completion of S_Σ ∪ \{a: ¬C\} contains a Σ-clash;
5. S contains a constraint of the form aK Pb, and a Pb ∉ S_Σ;
6. S contains x: ¬KC, x: KC, xK Py, or sK Px, and for each a ∈ O_Σ ∪ \{ι\}, every completion of S[x/a] contains a Σ-clash, where ι is an individual in O \ O_Σ.

**Proposition 6.2.4** Let Σ be a knowledge base in ACC, and S be a constraint system. Then S is Σ-solvable if and only if there exists at least one completion of S that contains no Σ-clash.

**Proof.** The proof is by induction on the number k of occurrences of the epistemic operator in the constraint system. If k = 0, then the claim trivially follows from the results in Chapter 4. If k > 0, then let H_1 be the following induction hypothesis: “Any complete constraint system with h < k occurrences of the epistemic operator is Σ-solvable if and only if it contains no Σ-clash”. Let S be a complete constraint system with k occurrences of the epistemic operator. We must show that

1. if S contains no Σ-clash, then it is Σ-solvable, and
2. if S is Σ-solvable, then it contains no Σ-clash.

Proof of (1). Suppose S contains no Σ-clash. We construct a triple (I, M(Σ), α). First of all, α is defined by:

- α(s) = γ(s) if s is an individual;
- α(x) = γ(a) if x is involved in at least one epistemic constraint in S (i.e., such that x : ¬KC, x : KC, xK Py, or sK Px is in S), and a is any individual in O_Σ ∪ \{ι\} such that S[x/a] does not contain any Σ-clash (there must be such an a, otherwise S would contain a clash of type 6);
- α(x) = p if x is not involved in any epistemic constraint, and p is any domain object that is assigned by α neither to an individual in O_Σ nor to a variable y in S such that y ≠ x.

Secondly, I is defined as follows:
• for every concept name \( A \) and every domain object \( p \), let \( p \in A^I \) if and only if there is a \( s \) such that \( \alpha(s) = p \), and \( s: A \) is in \( S \);

• for every role name \( P \) and every pair \( p_1, p_2 \) of domain objects, let \((p_1, p_2) \in P^I \) if and only if there are \( s, t \) such that \( \alpha(s) = p_1 \), \( \alpha(t) = p_2 \), and \( sPt \) is in \( S \).

• the interpretations of complex concepts and roles are derived from the semantic equations given in Section 6.2.1.

We show that \((I, \mathcal{M}(\Sigma), \alpha)\) satisfies every constraint in \( S \), i.e. \( S \) is \( \Sigma \)-solvable.

Consider any constraint of the form \( sPt \). By the construction of \( \alpha \) and \( I \), it is easy to verify that \((\alpha(s), \alpha(t)) \in P^{I,\mathcal{M}(\Sigma)}\).

Consider any constraint of the form \( s: C \). We proceed by a secondary induction on the form of \( C \), considering as base cases \( T \), \( A \), \( \neg A \), \( KC \), and \( \neg KC \).

If \( C \) is of one of the forms \( T \), \( A \), \( \neg A \), then it follows by the construction of \( \alpha \) and \( I \) that \( \alpha(s) \in C^{I,\mathcal{M}(\Sigma)} \).

Consider any constraint of the form \( a : \neg KC \). Since \( S \) does not contain any \( \Sigma \)-clash, there is at least one completion of \( \Sigma \) \( \cup \{a : \neg C\} \) which does not contain any \( \Sigma \)-clash. Since the number of occurrences of the epistemic operator in \( \Sigma \) \( \cup \{a : \neg C\} \) is less than \( k \), by the induction hypothesis \( H_1 \) and by Proposition 6.2.2, \( \Sigma \) \( \cup \{a : \neg C\} \) is \( \Sigma \)-solvable, which means that there is a model \( J \) of \( \Sigma \), such that \( \gamma(a) \in (\neg C)^{J,\mathcal{M}(\Sigma)} \), and hence \( \gamma(a) \notin C^{J,\mathcal{M}(\Sigma)} \). Since \( J \in \mathcal{M}(\Sigma) \), \( \gamma(a) \notin \bigcap_{J \in \mathcal{M}(\Sigma)} C^{J,\mathcal{M}(\Sigma)} \), hence, by definition of \((KC)^{I,\mathcal{M}(\Sigma)} \), \( \gamma(a) \notin (KC)^{I,\mathcal{M}(\Sigma)} \). Therefore, \((I, \mathcal{M}(\Sigma), \alpha)\) satisfies \( a : \neg KC \).

Consider any constraint of the form \( a : KC \). Since \( S \) does not contain any \( \Sigma \)-clash, every completion of \( \Sigma \) \( \cup \{a : \neg C\} \) contains a \( \Sigma \)-clash. Since the number of occurrences of the epistemic operator in \( \Sigma \) \( \cup \{a : \neg C\} \) is less than \( k \), by the induction hypothesis \( H_1 \) and by Proposition 6.2.2, \( \Sigma \) \( \cup \{a : \neg C\} \) is unsolvable, which means that for every model \( J \) of \( \Sigma \), \( \gamma(a) \in C^{J,\mathcal{M}(\Sigma)} \), i.e. \( \gamma(a) \in \bigcap_{J \in \mathcal{M}(\Sigma)} C^{J,\mathcal{M}(\Sigma)} \), and hence \( \gamma(a) \in (KC)^{I,\mathcal{M}(\Sigma)} \). Therefore, \((I, \mathcal{M}(\Sigma), \alpha)\) satisfies \( a : KC \).

Consider any constraint of the form \( x : KC \). Let \( a \) be the individual in \( O_S \cup \{x\} \) such that \( \alpha(x) = \gamma(a) \) (notice that by the construction of \( \alpha \), \( S[x/a] \) does not contain any \( \Sigma \)-clash). Since \( S[x/a] \) does not contain any \( \Sigma \)-clash, every completion of \( S \backslash \{a : \neg C\} \) contains a clash. Recall from the previous case that the last condition implies \( \gamma(a) \in \bigcap_{J \in \mathcal{M}(\Sigma)} C^{J,\mathcal{M}(\Sigma)} \), and hence \( \gamma(a) \in (KC)^{I,\mathcal{M}(\Sigma)} \). It follows that \( \alpha(x) \in (KC)^{I,\mathcal{M}(\Sigma)} \), that is, \((I, \mathcal{M}(\Sigma), \alpha)\) satisfies \( x : KC \). The other forms of epistemic constraints, namely \( x : \neg KC \), \( xKPy \), and \( xKPx \) can be treated analogously.

Now consider the following induction hypothesis \( H_2 \): “For every proper subconcept \( D \) of a concept \( C \), every constraint \( t : D \) is satisfied by \((I, \mathcal{M}(\Sigma), \alpha)\)."
Suppose $s : C$ is in $S$ and $C$ has the form $D \cap E$. Since $S$ is complete, both $s : D$ and $s : E$ are in $S$. By the induction hypothesis $H_2$, $(I, \mathcal{M}(\Sigma), \alpha)$ satisfies both constraints, and therefore, $(I, \mathcal{M}(\Sigma), \alpha)$ satisfies $s : D \cap E$, too. The remaining forms of constraints, namely, $E \cup D, \exists R.D$, and $\forall R.D$, can be treated analogously.

In conclusion, we have shown that the triple $(I, \mathcal{M}(\Sigma), \alpha)$ is a solution of $S$, and therefore $S$ is $\Sigma$-solvable.

Proof of (2). Suppose $S$ contains a $\Sigma$-clash. We now consider each type of clash in turn, and show that if $S$ contains a clash of that type, then it is $\Sigma$-unsolvable.

1. If $S$ contains a clash of type 1 or 2, then it is clearly $\Sigma$-unsolvable.

3. Suppose $S$ contains a constraint of the form $a : KC$, and there is at least one completion of $S_{\Sigma} \cup \{a : \neg C\}$ without $\Sigma$-clash. By the induction hypothesis $H_1$ and by Proposition 6.2.2, $S_{\Sigma} \cup \{a : \neg C\}$ is $\Sigma$-solvable, i.e., there is a triple $(I, \mathcal{M}(\Sigma), \alpha)$ that satisfies all the constraints of $S_{\Sigma} \cup \{a : \neg C\}$, and in particular $a : \neg C$. Therefore, $\gamma(a) \not\in C^I_{\mathcal{M}(\Sigma)}$, which implies that $\gamma(a) \not\in \bigcap_{\mathcal{J}:\Sigma} C^I_{\mathcal{M}(\Sigma)}$. It follows that the constraint $a : KC$ cannot be satisfied by any triple $(I, \mathcal{M}(\Sigma), \alpha)$, and therefore $S$ is $\Sigma$-unsolvable.

4. If $S$ contains a $\Sigma$-clash of the form $a : \neg KC$ or of the form $aKPa$, then we can proceed analogously to case 3.

6.1. Suppose $S$ contains a constraint of the form $x : \neg KC$, and for each $a \in C_{\Sigma} \cup \{i\}$, $S[x/a]$ contains a $\Sigma$-clash, where $i$ is an individual in $C$ not appearing in $S$. This means that for each $a \in C_{\Sigma} \cup \{i\}$, either every completion of $S_{\Sigma} \cup \{a : \neg C\}$ contains a $\Sigma$-clash, or every completion of $S[x/a] \setminus \{a : \neg KC\}$ contains a $\Sigma$-clash. By the induction hypothesis $H_1$, this implies that for each $a \in C_{\Sigma} \cup \{i\}$, either $S_{\Sigma} \cup \{a : \neg C\}$ is $\Sigma$-unsolvable, or $S[x/a] \setminus \{a : KC\}$ is $\Sigma$-unsolvable. We show that $S$ cannot be satisfied by any triple $(I, \mathcal{M}(\Sigma), \alpha)$.

Let us assume that the triple $(I, \mathcal{M}(\Sigma), \alpha)$ is a solution of $S$, with $\alpha(x) = p$, and there is $b \in C_S$ such that $\gamma(b) = p$. Obviously, we have that $(I, \mathcal{M}(\Sigma), \alpha)$ is a solution of $S[x/b]$ too, which means both that the constraint $b : \neg KC$ is satisfied by $(I, \mathcal{M}(\Sigma), \alpha)$—i.e., that $S_{\Sigma} \cup \{b : \neg C\}$ is $\Sigma$-solvable—and that every constraint in $S[x/b] \setminus \{b : \neg KC\}$ is satisfied by $(I, \mathcal{M}(\Sigma), \alpha)$—i.e., $S[x/b] \setminus \{b : \neg KC\}$ is $\Sigma$-solvable—which is a contradiction.

Let us assume that the triple $(I, \mathcal{M}(\Sigma), \alpha)$ is a solution of $S$, with $\alpha(x) = p$, and there is no $b \in C_S$ such that $\gamma(b) = p$. It is possible to show that in this case there is a solution $(I', \mathcal{M}(\Sigma), \alpha')$ of $S$ such that $\alpha'(x) = \gamma(i)$. It follows that $(I', \mathcal{M}(\Sigma), \alpha')$ is a solution of $S[x/i]$, too, which is again a contradiction as in the previous case.

6.2, 6.3, 6.4. If $S$ contains a $\Sigma$-clash of the form $x : KC, xKP$, or $xKP$ x, then we can proceed analogously to case 6.1.

The results reported in Chapter 4 show that it is decidable whether a
constraint system that does not include any epistemic constraint is $\Sigma$-solvable. In particular, it is shown that the number of completions obtainable from such a system is finite. Based on this fact, one can easily prove that the number of completions of an $\mathcal{ALCK}$-constraint system is also finite. Observe that, in order to decide whether a complete constraint system $S$ has a $\Sigma$-clash or not, a finite number of $\Sigma$-satisfiability checks suffices, each involving a constraint system whose number of epistemic constraints is less than the number in $S$. Therefore, one can show by induction that the above rules provide us with an algorithm for checking an $\mathcal{ALCK}$-constraint system for $\Sigma$-solvability.

**Proposition 6.2.5** Let $\Sigma$ be a knowledge base in $\mathcal{ALC}$. Then it is decidable whether an $\mathcal{ALCK}$ constraint system $S$ is $\Sigma$-solvable or not.

This in turn implies that we have an effective method both for checking whether $\Sigma \models C(a)$, and for computing the answer set of $C$ w.r.t. $\Sigma$. The next section discusses in more detail the computational complexity of the method.

### 6.2.3 Complexity of Reasoning with the Epistemic Operator

In this section we investigate the complexity of answering epistemic queries. First, we study the complexity of the epistemic entailment of an $\mathcal{ALCK}$-query by an $\mathcal{ALC}$-knowledge base. Subsequently, we focus on two cases of special interest. In particular, we show that a careful use of the $K$ operator helps in reducing the complexity of reasoning. Further, we show a case in which the introduction of the $K$ operator substantially increases the complexity of reasoning.

Regarding the computational complexity of answering $\mathcal{ALCK}$-queries, we prove that this problem can be solved in PSPACE. To this aim, the analysis of the calculus allows us to realize an interesting relation between reasoning with epistemic concepts and reasoning with concepts involving collections of individuals.

In fact, in both cases the crucial point is how to deal with the substitution of variables with individuals in the constraint system. In fact, substituting two variable in different traces with the same individual might result in a clash that is not discovered by an algorithm that works exploring a trace at a time (e.g. the PSPACE algorithm for $\mathcal{ALC}$).

The intuition underlying such relation is that a concept of the form $KC$ can be considered equivalent to the concept $\{a_1, \ldots, a_n\}$, where $a_1, \ldots, a_n$ are exactly the individuals for which $\Sigma \models C(a_i)$ holds. However, reasoning with $\mathcal{ALCK}$ is more complex than reasoning with $\mathcal{ALCO}$ for two reasons: The first is that, while using $\mathcal{ALCO}$ the set $\{a_1, \ldots, a_n\}$ is given, in $\mathcal{ALCK}$ such set must be computed (possibly in a recursive way). The second concerns the presence of the $K$ operator in front of a role name. A role of the form $KP$ is
equivalent to a role of the form \( \{(a_1, b_1), \ldots, (a_n, b_n)\} \) which is not considered
in languages with \( \mathcal{O} \).

Despite the above differences, we can use a technique similar to the one
used for reasoning in \( \mathcal{ALCO} \), for checking \( \Sigma \models C(a) \), as shown below.

**Complexity of \( \mathcal{ALCK} \)**

As for \( \mathcal{ALCO} \), we introduce in the calculus the \( \rightarrow_{\text{guess}} \) rule defined in Section 5.4.1 and we replace the definition of a \( \Sigma \)-clash of type 6 with the following one, so that it checks whether a substitution agrees with the guess:

6'. \( S \) contains \( x : \neg KC, x : KC, x KP y, \) or \( sKP x \), and

(a) for each \( a \in \mathcal{O}_{\Sigma} \), we have that \( S_x \not\subseteq S_a \);

(b) every completion of \( S[x/\iota] \) contains a \( \Sigma \)-clash, where \( \iota \) is an individual in \( \mathcal{O} \setminus \mathcal{O}_S \).

Notice that the individuals in \( \mathcal{O}_S \setminus \mathcal{O}_{\Sigma} \) are not considered. Those are
the individuals previously introduced by other checks for a clash of type 6'.b. They are not taken into account because, since no properties are stated for
such individuals in \( \Sigma \), if \( x \) can be substituted with one of them without leading to a \( \Sigma \)-clash, then it can be substituted with \( \iota \) without giving a \( \Sigma \)-clash, too. Such property is proved in the following lemma.

**Lemma 6.2.6** Given a constraint system \( S \), an individual \( \iota_1 \) in \( \mathcal{O}_S \setminus \mathcal{O}_{\Sigma} \), an individual \( \iota_2 \) non in \( \mathcal{O}_S \), and a variable \( x \), if \( S[\iota_1/x] \) is satisfiable then \( S[\iota_2/x] \) is satisfiable too.

The following Lemmata are analog to Lemmata 5.4.4 and 5.4.5 with the
epistemic operator.

**Lemma 6.2.7** Given a constraint system \( S \), an individual \( b \), and a concept \( C \in \text{Subc}(S) \), such that the \( \rightarrow_{\text{guess}} \) rule is applicable to \( b \) and \( C \) in \( S \), \( S \) is \( \Sigma \)-solvable if and only if there exists a \( S' \) such that \( S' \) is obtained from \( S \) by
the application of the \( \rightarrow_{\text{guess}} \) rule to \( b \) and \( C \), and \( S' \) is \( \Sigma \)-solvable.

**Lemma 6.2.8** Given a constraint system \( S \), let \( S' \) be a constraint system
obtained from \( T_S \) by a sequence of applications of the completion rules. Let \( x \)
be a variable in \( S' \) and \( b \) an individual in \( \mathcal{O}_{\Sigma} \). Then \( S'[x/b] \) is \( \Sigma \)-solvable if and only if \( S' \) is \( \Sigma \)-solvable and \( S'_x \subseteq S'_b \).

The algorithm for instance checking is shown in Figure 6.3.

**Theorem 6.2.9** Algorithm \( \text{Inst}(\Sigma, a, C) \) is correct and terminating.
Algorithm Instance($\Sigma$, $a$, $C$):
Input knowledge base $\Sigma$ in $\mathcal{ALC}$, individual $a$, $\mathcal{ALCX}$-concept $C$;
Output TRUE if $\Sigma \models C(a)$; FALSE otherwise
begin
  $S := S_0 \cup \{a : \neg C\}$;
  loop
    compute a new $T_S$;
    if clash_free($T_S$),
      then return FALSE
  endloop;
  return TRUE
end.
Function clash_free($S$ : constraint system) : boolean;
begin
  if ($s : A, s : \neg A \in S$) or ($s : \bot \in S$) or ($aKPa \in S$ and $aPa \notin S_0$),
    then return FALSE (* clash cases 1, 2, and 5 *)
  elseif $a : KC_1 \in S$
    then return Instance($\Sigma$, $a$, $C_1$) and
      clash_free($S \setminus \{a : KC_1\}$) (* clash case 3 *)
  elseif $a : \neg KC_1 \in S$
    then return (not Instance($\Sigma$, $a$, $C_1$)) and
      clash_free($S \setminus \{a : \neg KC_1\}$) (* clash case 4 *)
  elseif $\sigma \in S$ (where $\sigma = x : KC_1$ or $\sigma = x : \neg KC_1$ or $\sigma = xKPy$ or $\sigma = sKPx$)
    then return ((exists $b \in O_\Sigma : S_x \subseteq S_b$) and clash_free($S \setminus \{\sigma\}$)) or
      clash_free($S[x/x]$) with $i \notin O_\Sigma$ (* clash case 6 *)
  elseif $s : C_1 \cap C_2 \in S$
    then return clash_free($S \setminus \{s : C_1 \cap C_2\} \cup \{s : C_1, s : C_2\}$)
  elseif $s : C_1 \cup C_2 \in S$
    then return (clash_free($S \setminus \{s : C_1 \cup C_2\} \cup \{s : C_1\}$) or
      clash_free($S \setminus \{s : C_1 \cup C_2\} \cup \{s : C_2\}$))
  elseif ($s : \forall R.C_1 \in S$) and (sRt holds in $S$) and ($t : C_1 \notin S$)
    then return clash_free($S \cup \{t : C_1\}$)
  elseif $s : \exists KP.C_1 \in S$
    then return clash_free($S \setminus \{s : \exists KP.C_1\} \cup \{sKPz, x : C_1\}$)
  elseif $s : \exists P.C_1 \in S$
    then return clash_free($S \cup \{sPx, x : C_1\} \setminus \{s \exists Q.D\}$) and
      clash_free($S \setminus \{s : \exists P.C_1\}$)
  else return TRUE
end;

Figure 6.3: The algorithm for instance checking
Proof. Follows from Lemmata 6.2.7 and 6.2.8 and from the results in [SS91, BH91b] on the independence of the traces in $\mathcal{AC}$. \hfill \Box

**Theorem 6.2.10** Algorithm Instance$(\Sigma, a, C)$ works in polynomial space.

Proof. First of all, notice that the number of recursive calls of the algorithm Instance is bounded by the number of nestings of the $\mathbf{K}$ operator in the concept $C$, and it is therefore linear in $|C|$.

We next prove that the size of the constraint system involved in any recursive call of the procedure clash free is polynomial with respect to the size of the starting constraint system $T_S$. First, the number of variables involved in a trace is bounded by the maximum nesting of existential quantifiers in $T_S$, which is linear in $|T_S|$. Second, the number of constraints is polynomially bounded by the number of objects (which is polynomial in $|T_S|$). Since each call adds new constraints and the maximum number of constraints is polynomial, it follows that the number of recursive calls is polynomially bounded, too. The claim then follows from Lemma 5.4.7. \hfill \Box

**Complexity of $\mathcal{AL}/\mathcal{ALEK}$**

The examples of Chapter 8 show that the existential quantification allows one to express queries which require reasoning by case analysis. In Chapter 4, we have shown that this kind of reasoning makes deductions in concept languages computationally hard. In the examples given in Chapter 8 we show that the use of $\mathbf{K}$ may allow us to express queries ruling out case analysis. In particular, this is done by replacing the concepts of the form $\exists R.D$ with concepts of the form $\exists \mathbf{K} R.\mathbf{K} D$. Those examples suggest that a decrease of the complexity of reasoning is made possible by the use of $\mathbf{K}$. In this section, we obtain a general result about the complexity of reasoning in the languages $\mathcal{ALE}$ and $\mathcal{AL}$: The language $\mathcal{ALE}$ is the sublanguage of $\mathcal{AL}$ that consists of all simple concepts which do not contain the union constructor, whereas $\mathcal{AL}$ consists of the $\mathcal{ALE}$-concepts that contain existential quantifications only of the form $\exists R.\top$. In Chapter 4, it is proved that the problem of checking if $\Sigma \models C(a)$, where $\Sigma$ is a knowledge base in $\mathcal{AL}$, and $C$ is an $\mathcal{ALE}$-concept, is coNP-hard w.r.t. the size of $\Sigma$ (knowledge base complexity). The same problem in the case of an $\mathcal{ALE}$-knowledge base $\Sigma$ is proved to be PSPACE-complete w.r.t. the size of $\Sigma$ and $C$ (combined complexity).

We call $\mathcal{ALEK}$ the language obtained by adding the $\mathbf{K}$ operator to $\mathcal{ALE}$ and $\mathcal{ALEK}^-$, the language $\mathcal{ALEK}$ with the restriction that the existential quantifications are only of the form $\exists \mathbf{K} R.\mathbf{K} D$.

We prove that the answer of a query to a knowledge base in $\mathcal{AL}$ can be computed in polynomial time w.r.t. the size of the knowledge base (data complexity) if the query is in $\mathcal{ALEK}^-$. 
In addition, we prove that the same problem is in coNP w.r.t. the combined complexity.

Such results, compared respectively with the coNP-hardness and PSPACE-completeness results mentioned above, confirm the fact that we can lessen the complexity of reasoning using the K operator.

Specifically, we design a polynomial algorithm that checks whether $\Sigma \models C(a)$, where $\Sigma$ is an $\mathcal{AL}$-knowledge base and $C$ is an $\mathcal{ALEK}^-$-concept. The algorithm is an implementation of the calculus of Section 3.1 specialized to deal with an $\mathcal{AL}$-knowledge base and an $\mathcal{ALEK}^-$-query. It is specialized in the sense that there are certain rule applications of the general calculus that are not considered, because they cannot take place in our case.

First, since $\mathcal{ALEK}$ does not have disjunction, no conjunction may occur in the simple form of any negated $\mathcal{ALEK}^-$ concepts. Since the $\rightarrow_\pi$-rule is the only one that can generate two open constraints on the same variable, it follows that each object can be in at most one open constraint involving a subconcept of $\neg C$. Such open constraint is highlighted in the procedure $\text{Clash free}$, being considered outside the constraint system $S$, carried by the 2nd and the 3rd parameter of the procedure.

The $\rightarrow_\varphi^\pi$, $\rightarrow_\pi$, and $\rightarrow_\tau$-rule are applied to the constraints involving the concepts of $\mathcal{AL}$ in the assertions of the knowledge base implicitly in the algorithm, by the operation of computing the completion of a constraint system.

In addition, some clash types cannot appear in our case. First, the constraints of the form $s : \mathcal{K}C$ do not appear because only negated $\mathcal{ALEK}^-$-concepts must be considered, and $\mathcal{ALEK}^-$ does not allow for general negation. Moreover, in our case a constraint of the form $x : \neg C$ is always $\Sigma$-solvable.

In fact, since in $\mathcal{ALEK}$ is not possible to express a concept equivalent to $\top$, the substitution $\{x/i\}$ always solves it.

Notice that, since the whole constraint system has polynomial size w.r.t. the knowledge base (see below), there is no need to use the modified calculus developed in Section 6.2.3. The algorithm, called $\text{Inst}_{\mathcal{AL}/\mathcal{ALEK}^-}$, is shown in Figure 6.4.

The following lemma states the correctness of the algorithm. Moreover, it shows that the time complexity of the algorithm is polynomial.

**Lemma 6.2.11** Let $\Sigma$ be a knowledge base in $\mathcal{AL}$, $a$ be an individual, and $C$ be an $\mathcal{ALEK}^-$-concept. Then $\text{Inst}_{\mathcal{AL}/\mathcal{ALEK}^-}(\Sigma, a, C)$ terminates, returning true if $\Sigma \models C(a)$, and false otherwise. Moreover, it runs in polynomial time with respect to $|\Sigma|$.

**Proof.** The correctness of the algorithm follows from soundness and completeness of the calculus and the above observations. For the termination, it is sufficient to observe that in any recursive call of the algorithm the actual parameter corresponding to $E$ decreases in length.
Algorithm $\text{Inst}_{\mathcal{AL}/\mathcal{AL}E\mathcal{K}^-} (\Sigma, a, C)$

Input knowledge base $\Sigma$ in $\mathcal{AL}$, individual $a$, $\mathcal{AL}E\mathcal{K}^-$-concept $C$;
Output TRUE if $\Sigma \models C(a)$; FALSE otherwise
begin
$S :=$ completion of $S_\Sigma$;
if $S$ contains a clash
then return TRUE
else return not $\text{Clashfree}(S, a, \neg C)$
end.

Function $\text{Clashfree}(S : \text{constraint system}, s : \text{object}, E : \text{concept}) : \text{boolean}$;
begin
 case $E$ of
  $\neg A$: return (s: $A \notin S$);
  A: return (s: $\neg A \notin S$);
  $\neg C_1 \cup \neg C_2$: return $\text{Clashfree}(S, s, \neg C_1)$ or $\text{Clashfree}(S, s, \neg C_2)$;
  $\exists P, \neg C_1$: return $\text{Clashfree}(\text{completion of } S \cup \{sPx \mid x, \neg C_1\})$;
  $\neg KC$: if $s$ is an individual
  then return $\text{Clashfree}(S, s, \neg C)$
  else return TRUE; (* $x$: $\neg KC$ is always satisfiable *)
  $\forall K, \neg C_1$: if $s$ is an individual
  then for all $b \mid (sPb \in S)$
    if not $\text{Clashfree}(S, b, \neg C_1)$
    then return FALSE;
  else for all $a, b \mid (aPb \in S)$
    if not $\text{Clashfree}(S[a/s], b, \neg C_1)$
    then return FALSE;
  return TRUE
  $\exists K, \neg C_1$: if $s$ is an individual
  then if there exists $b \mid (sPb \in S)$
    and $\text{Clashfree}(S, b, \neg C_1)$
    then return TRUE;
  else return FALSE
  else if there exist $a, b \mid (aPb \in S)$
    and $\text{Clashfree}(S[a/s], b, \neg C_1)$
    then return TRUE;
  else return FALSE
  endcase
end

Figure 6.4: The algorithm for Instance Checking in $\mathcal{AL}/\mathcal{AL}E\mathcal{K}^-$
With respect to complexity, first notice that the completion of a constraint system in \( \mathcal{AC} \) can be computed in polynomial size \([LS91a]\). Since all the other operations performed by each call of the procedure \textit{ClashFree} are polynomial, it follows that the single call of the procedure \textit{ClashFree} runs in polynomial time. In addition, we can easily recognize that the number of calls is bounded by \(|\Sigma|^{|C|}\). In fact, each call can fire a number of calls that is at most the cardinality of \( \mathcal{O}_2 \), which is bounded by \(|\Sigma|\). The depth of the recursive calls tree is bounded by the number of nested quantifiers in \( C \), and therefore is bounded by \(|C|\). We can conclude that the whole algorithm works in polynomial time w.r.t. \(|\Sigma|\).

**Theorem 6.2.12** Query answering using \( \mathcal{AC} \) as data language and \( \mathcal{ACLE}\neg \) as query language can be done in polynomial time w.r.t. the knowledge base complexity.

**Proof.** Easily follows from Lemma 6.2.11. \( \square \)

Notice that the same result does not hold if we consider the size of the query, too. In fact, the algorithm \( \text{Inst}_{\mathcal{AC}/\mathcal{ACLE}\neg} \) runs in polynomial time w.r.t. the size of the knowledge base, but in (deterministic) exponential time w.r.t. the size of the query. However, the function \textit{ClashFree} works in non-deterministic polynomial, and allowing us to deduce that the problem is in coNP, as proved in the following theorem.

**Theorem 6.2.13** Query answering using \( \mathcal{AC} \) as data language and \( \mathcal{ACLE}\neg \) as query language is a coNP problem w.r.t. the combined complexity.

**Proof.** A fare

In the next section, we prove that the above problem is coNP-hard w.r.t. the combined size.

**Complexity of \( \mathcal{AC}_0/\mathcal{AC}_0\neg \)**

In this section we prove that, w.r.t. the combined complexity, reasoning with \( \neg \) is coNP-hard. We prove this result for the most simple language for which we found that it holds, \( \mathcal{AC}_0 \) (defined below). The result obviously extends to more expressive languages, such as \( \mathcal{AC} \) and \( \mathcal{ACLE} \).

The language \( \mathcal{AC}_0 \) is obtained from \( \mathcal{AC} \) by eliminating the constructor for the existential quantification, and the language \( \mathcal{AC}_0\neg \), that is \( \mathcal{AC}_0 \) plus the epistemic operator. We now show that deciding whether an \( \mathcal{AC}_0 \)-knowledge base epistemically entails an \( \mathcal{AC}_0\neg \)-query is coNP-hard.

We prove the claim by a reduction from the complement of the uniform-3SAT problem, which is known to be NP-complete. The uniform-3SAT problem amounts to decide the satisfiability of a set of propositional clauses each
one consisting of exactly three literals that are either all positive or all negative.

Let \( \Gamma = \{ \alpha_1, \ldots, \alpha_n \} \) be such a set of propositional clauses. Without loss of generality, we assume that the clauses \( \alpha_1, \ldots, \alpha_k \) are composed of positive literals and the clauses \( \alpha_{k+1}, \ldots, \alpha_n \) are composed of negative ones (with \( 1 \leq k \leq n \)). We will construct an \( \mathcal{ALC}_0 \)-knowledge base \( \Sigma \) and an \( \mathcal{ALC}_0 \mathcal{K} \)-query \( D(c_1) \) such that \( \Gamma \) is unsatisfiable if and only if \( \Sigma \models D(c_1) \).

Suppose the propositional symbols occurring in \( \Gamma \) are \( p_1, \ldots, p_m \). We will consider the \( p_i \) as individuals. In addition, assume that there are individuals \( c_1, \ldots, c_{n+1} \) which are distinct from \( p_1, \ldots, p_m \). Moreover, assume that \( A \) is a primitive concept. Finally, assume that \( P \) and \( Q_1, \ldots, Q_n \) are primitive roles.

Now, let \( \Sigma \) be the knowledge base containing the following assertions

- the assertions \( \forall P. A(c_i) \) for \( i = 1, \ldots, k \);
- the assertions \( \forall P. \neg A(c_i) \) for \( i = k + 1, \ldots, n \);
- the assertions \( Q_i(q_i^1, c_{i+1}), Q_i(q_i^2, c_{i+1}), Q_i(q_i^3, c_{i+1}) \) for \( i = 1, \ldots, n \), where \( q_i^1, q_i^2, q_i^3 \) are the propositional symbols occurring in the clause \( \alpha_i \).

Let \( D \) be the concept \( \forall P. \forall KQ_1, \forall P. \forall KQ_2, \ldots, \forall P. \forall KQ_n, \bot \), that is, \( D \) consists of a chain of universally quantified roles where the roles \( P \) and \( KQ_i \) alternate.

**Lemma 6.2.14**

\[ \Sigma \models D(c_1) \quad \text{if and only if} \quad \Gamma \text{ is unsatisfiable.} \]

Recall that \( \Sigma \models D(c_1) \) holds if and only if the constraint system \( S = S_\Sigma \cup \{ c_1 : \neg D \} \) is \( \Sigma \)-unsolvable.

The constraint system \( S \), rewriting \( \neg D \) in simple form, assumes the following form:

\[
S = \{ c_1 : \forall P.A, \ldots, c_k : \forall P.A, c_{k+1} : \forall P.\neg A, \ldots, c_n : \forall P.\neg A, \\
q_1^1Q_1c_2, q_2^1Q_1c_3, q_3^1Q_1c_2, \ldots, q_n^1Q_n^n c_{n+1}, q_n^2Q_n^n c_{n+1}, q_n^3Q_n^n c_{n+1}, \\
c_1 : \exists P. \exists KQ_1, \exists P. \exists KQ_2, \ldots, \exists P. \exists KQ_n, \bot \}
\]

Applying the completion rules to \( S \), the only completion obtained (up to variable renaming) is the following one:

\[
S' = S \cup \{ c_1Px_1, x_1KQ_1y_2, y_2Px_2, x_2KQ_2y_3, \ldots, y_nPx_n, x_nKQ_ny_{n+1}, \\
y_{n+1} : T, x_1 : A \}
\]
Looking at the clash definition, we realize that the only possible type of clash that may apply to $S'$ is type 6, due to the presence of the constraints of the form $x_i K Q_{i+1}$ (for $i = 1, \ldots, n$). To this respect, it is easy to see that the only substitution for the variables $y_i$ (for $i = 2, \ldots, n$) that does not contain a $\Sigma$-clash of type 5 is $[y_i/c_i]$. In fact, all the constraints in $S$ regarding each role $Q_i$ are $q_i^1 Q_i c_{i+1}, q_i^2 Q_i c_{i+1}, q_i^3 Q_i c_{i+1}$, and all three involve the individual $c_{i+1}$ as second argument.

It follows that $S'$ is $\Sigma$-solvable if and only if there exist a completion of $S'[y_2/c_2, \ldots, y_{n+1}/c_{n+1}]$ that is $\Sigma$-solvable. The only completion of the constraint system $S'[y_2/c_2, \ldots, y_{n+1}/c_{n+1}]$ is the following one:

$$
S'' = S \cup \{ c_1 P x_1, x_1 K Q_1 c_2, c_2 P x_2, x_2 K Q_2 c_3, \ldots, c_n P x_n, x_n K Q_n c_{n+1},\n$$

$$
c_{n+1}: T, x_1: A, \ldots, x_k: A, x_{k+1}: \neg A, \ldots, x_n: \neg A\}
$$

obtained applying the $\neg \phi$-rule to the constraints $c_i: \forall P, A$ (for $i = 1, \ldots, k$) and to the constraints $c_i: \forall P, \neg A$ (for $i = k+1, \ldots, n$).

The constraint system $S''$ is $\Sigma$-solvable if and only if there exists a substitution for the variables $x_i$ that yields to a constraint system without $\Sigma$-clashes. According to the assertions in $\Sigma$, for each $x_i$ (for $i = 1, \ldots, n$) there are three candidate substitutions, namely $[x_i/q_i^1]$, $[x_i/q_i^2]$, and $[x_i/q_i^3]$; all the others yield immediate $\Sigma$-clash of type 5.

"$\Leftarrow$" We now prove that $\Gamma$ is satisfiable there exists a substitution $\delta$ if and only if for $x_i$ (for $i = 1, \ldots, n$) such that $S'' \delta$ is $\Sigma$-solvable. In order to prove this, suppose $T = S''[x_1/q_1^1, \ldots, x_n/q_n^m]$ is $\Sigma$-solvable, where $j_i$ is 1, 2, or 3 for $i = 1, \ldots, n$. The constraint system $T$ contains the constraints $q_i^j: A$ (for $i = 1, \ldots, k$) and the constraints $q_i^k: \neg A$ (for $i = k+1, \ldots, n$). Thus there exists an interpretation $I$ such that $\gamma(q_i^1) \in A^I^W, \ldots, \gamma(q_i^3) \in A^I^W$ and $\gamma(q_k^j) \not\in A^I^W, \ldots, \gamma(q_m^m) \not\in A^I^W$. Consider the propositional assignment $J$ defined in the following way:

$$
J(p) = true \text{ if and only if } \gamma(p) \text{ is in } A^I^W
$$

The propositional assignment $J$ assigns $true$ to at least one literal for each clause $\alpha_i$, proving that $\Gamma$ is satisfiable.

"$\Rightarrow$" Now suppose $\Gamma$ satisfiable, and let $J$ be a propositional assignment that satisfies it. Since $\Gamma$ satisfies $\Gamma$, in each clause $\alpha_i$ (for $i = 1, \ldots, k$) there is a propositional symbol, say $q_i^j$, such that $J(q_i^j) = true$, and in each clause $\alpha_h$ (for $h = k+1, \ldots, n$) there is a propositional symbol, say $q_h^k$, such that $J(q_h^k) = false$. Consider the following interpretation $I$:

$$
A^I = \{ \gamma(p) \mid J \text{ evaluates } p \text{ as } true \}$$
\[ P^T = \{ (\gamma(c_i), \gamma(q^j_i)) \mid i = 1, \ldots, n \text{ and } j = 1, 2, 3 \} \]
\[ Q^T_i = \{ (\gamma(q^j_i), \gamma(c_i)) \mid i = 1, \ldots, n \text{ and } j = 1, 2, 3 \} \]

It is easy to see that \((I, W, \alpha)\) is a \(\Sigma\)-solution of the constraint system \(T\), independently of \(\alpha\), proving the claim. \(\square\)

The following theorem easily follows from Lemma 6.2.14.

**Theorem 6.2.15** Deciding whether an \(\mathcal{AL}_0\)-knowledge base epistemically entails an \(\mathcal{AL}_0K\)-query is \(\text{coNP}\)-hard (combined complexity).

### 6.2.4 Closed World Reasoning with the Epistemic Operator

The reason for the open world semantics of concept languages is that they are generally used in applications where one has to account for incomplete information. For example, even if all the known friends of Bob are female, one does not want to conclude that all possible friends that Bob has are female.

On the other hand, there are situations (see Chapter 8) where it is natural to query a knowledge base under the Closed World Assumption.

We show here that under certain restrictions, our query language allows us to achieve at least the expressive power of the (naive) Closed World Assumption (CWA) (see [Rei78]). The restrictions affect both the content and the language of the knowledge base. In the following we consider simple knowledge bases expressed in the language \(\mathcal{AL}_0\), whose concepts are formed according to the rule:

\[ C, D \rightarrow A \mid \neg A \mid C \cap D \mid \forall R.C. \]

More complex languages and knowledge bases and more powerful forms of closed world reasoning (e.g., Generalized CWA [Min82]) require a more sophisticated treatment, which is outside the scope of this thesis.

We briefly reformulate the CWA in the setting of a simple \(\mathcal{AL}_0\) knowledge base \(\Sigma\). Let \(\Sigma^{\text{CWA}}\) be the knowledge base obtained from \(\Sigma\) by adding \(\neg A(a)\) or \(\neg P(a, b)\), respectively, for every assertion \(A(a)\) or \(P(a, b)\) that is not entailed by \(\Sigma\). Now, for any concept \(C\) the statement \(C(a)\) follows from \(\Sigma\) under the CWA, written \(\Sigma \models_{\text{CWA}} C(a)\), if \(C(a)\) follows from \(\Sigma^{\text{CWA}}\).

Given an \(\mathcal{ALC}\)-concept \(C\), we define the \(\mathcal{ALCK}\)-concept \(\overline{C}\) as follows:

\[ \overline{A} = \text{KA} \]
\[ \overline{\neg A} = \neg\text{KA} \]
\[ \overline{C \cap D} = \overline{C} \cap \overline{D} \]
\[ \overline{C \cup D} = \overline{C} \cup \overline{D} \]
\[ \overline{\exists P.C} = \exists \text{KP,} \overline{C} \]
\[ \overline{\forall P.C} = \forall \text{KP,} \overline{C}. \]
The above transformation puts an epistemic operator in front of every concept name and role name. Now, it is possible to show that, if $\Sigma$ is a simple knowledge base in $\mathcal{AL}_0$, $C$ is an $\mathcal{ALC}$-concept, and $a$ is an individual, then $\Sigma \models_{\text{CWA}} C(a)$ if and only if $\Sigma \models \overline{C(a)}$. Moreover, checking whether $\Sigma \models \overline{C(a)}$ can be done in time polynomial in the size of both the query and the knowledge base. This is in sharp contrast to answering queries that are formulated with arbitrary $\mathcal{ALC}$-concepts, which is a PSPACE-hard problem even for a fixed knowledge base in $\mathcal{AL}_0$.

Intuitively, the reason for the above result is that for an $\mathcal{AL}_0$-concept $C$ the assertion $C(a)$ is logically equivalent to a finite set of Horn clauses and, therefore, simple $\mathcal{AL}_0$ knowledge bases are equivalent to sets of Horn clauses. As a consequence, if such a knowledge base is satisfiable, it always has one minimum model, say $\mathcal{I}_0$. Hence, evaluating a query under the CWA amounts to evaluating it in $\mathcal{I}_0$. Now, putting a $\mathbf{K}$ in front of every concept name $A$ and role $P$ has the effect that $A$ and $P$ are taken as the intersection of their interpretations in all models of $\Sigma$, i.e., they are interpreted in $\mathcal{I}_0$. This explains why closed world reasoning can be enforced through the use of $\mathbf{K}$. That queries can be answered in polynomial time is due to the fact that, on the one hand, the Horn clauses corresponding to a simple knowledge base in $\mathcal{AL}_0$ do not contain function symbols and, on the other hand, that concepts have a hierarchical structure that makes them suitable for efficient bottom up evaluation.

Notice that transforming a query $C$ into $\overline{C}$ implies answering the query under the assumption that the knowledge about every role is complete, like for example in [Neb90a, p. 113]. On the other hand, as noted in [GP86], there are situations where we would like to apply the closed world assumption only to some of the concepts and the roles of the knowledge base.
Chapter 7

Beyond TBox/ABox systems

The strict separation between TBox and ABox is a controversial point in concept-based systems. In fact, although it offers good computation properties, it generally limit the expressive power of the overall system. We have seen in Chapter 5 an example of breaking such separation.

In this chapter, we see some representational mechanisms that we use in the next chapter in our example of a concept-based system. Such mechanisms go beyond the ABox/TBox dichotomy, instead they cannot be obviously placed in any of them.

One mechanism that have been used in actual systems is the trigger rules (e.g. in CLASSIC). Intuitively a trigger rule is a statement of the form "if C then D", with the meaning that if an individual is known to be in the extension of C then it should be placed in the extension of D.

The trigger rules, however, suffer the problem of not having a clear declarative semantics. Nevertheless, they turned out to be very useful in practical cases. With the help of the epistemic operator defined in previous chapter we can provided the trigger rules with a formal semantics (in Section 7.1). Such semantics for trigger rules is now the one used in [PS93].

Another representation mechanism, usually employed in database applications, is the Integrity Constraints. We discuss them in Section 7.2 and we show that they can be easily formalized with the use of the epistemic operator.

In Section 7.3, we discuss the use of the free TBox mechanism. We also prove the decidability of reasoning with free TBoxes in the language $\mathcal{ALCN^\bot}$ and we discuss the relationship between the expressivity of general TBoxes and free TBoxes.

The results of Section 7.1 have been originally proposed in [DLN+92] and they have been expanded and revised in [DLN+93]. The results of Section 7.2 have been presented in [DLN+92]. The results of Section 7.3 have been proposed in [BDS93b], based on previous research done in [DLNS91].
7.1 Rules as Epistemic Statements

In the previous chapter we have considered knowledge bases in $\mathcal{ALC}$. We now consider the case where epistemic sentences of a special kind are introduced into the knowledge base, and show that this extension formalizes the usage of trigger rules (or simply rules), as provided in many practical systems based on concept languages. In fact, systems such as CLASSIC and LOOM, in addition to classical TBox- and ABox-statements provide another mechanism for expressing knowledge, by means of rules. Such rules are sentences the form

$$C \Rightarrow D$$

where $C, D$ are concepts. The meaning of a rule is “if an individual is proved to be an instance of $C$, then derive that it is also an instance of $D$” (see [BBMA89b]), and its behavior is usually described in terms of a forward reasoning process that adds to the knowledge base the assertion $D(a)$ whenever $C(a)$ is proved to hold. We call procedural extension of a knowledge base $\Sigma$ w.r.t. a set of rules the knowledge base resulting from such a forward reasoning process.

Rules in the context of concept-based systems are often defined informally. Attempts to precisely capture the meaning of such rules are based either on viewing them as knowledge base updates (see for example the TELL operation of [Lev84]), or on ad hoc semantics (see [Sch89a]). Our aim in this section is to show that rules can be nicely formalized as particular epistemic sentences.

In the following we consider knowledge bases in $\mathcal{ALCK}$ of the form $\langle T, A, R \rangle$, where $T$ is a free TBox, $A$ is an ABox, and $R$ is a set of epistemic sentences, each one of the form\footnote{In [DLN+92] we used the notation $KC \Rightarrow KD$. The two notations are equivalent in the semantics we give.}

$$KC \preceq D$$

where $C$ and $D$ are $\mathcal{ALC}$-concepts. We call these sentences trigger rules, since they are our formal counterpart of the rules $C \Rightarrow D$. We also call $C$ the antecedent and $D$ the consequent of the trigger rule. As a notational convenience we write the knowledge base $\langle T, A, R \rangle$ as $\langle \Sigma, R \rangle$, where $\Sigma = \langle T, A \rangle$.

From the definition of the semantics of knowledge bases in $\mathcal{ALCK}$ it follows that an epistemic interpretation $(I, W)$ satisfies the trigger rule $KC \preceq D$ if $(KC)^{I,W} \subseteq D^{I,W}$. Intuitively, the set of epistemic sentences $R$ restricts the set of models for $\Sigma$ to the maximal subsets that satisfy every trigger rule in $R$. More precisely, it can be shown that if $(I, W)$ is an epistemic model for $\Phi = \langle \Sigma, R \rangle$, then $W$ is a maximal subset of $M(\Sigma)$ such that for each $J \in W$, $(J, W)$ satisfies every sentence in $R$. Because of the form of such sentences,
it can also be shown that there exists only one maximal subset \( \mathcal{W} \) of \( \mathcal{M}(\Sigma) \) such that for all \( \mathcal{J} \in \mathcal{W}, \{ \mathcal{J}, \mathcal{W} \} \) satisfies every sentence in \( \Phi \).

Observe that when a concept \( C \) is equivalent to \( \top \), i.e. \( C^\mathcal{I} = \Delta^\mathcal{I} \) for every interpretation \( \mathcal{I} \), a trigger rule \( \mathbf{KC} \leq D \) is equivalent to the inclusion \( \top \leq D \). Besides this case, however, trigger rules are not expressible by inclusions. Indeed, the main difference between rules and inclusions is that the formers are intended to provide a reasoning mechanism which applies them in one direction only, namely from the antecedent to the consequent. Our formalization of rules with the epistemic operator correctly captures this property, as shown in the following example.

Consider the knowledge base \( \Phi = \langle \emptyset, \{ \neg B(a) \}, \{ \mathbf{KA} \leq B \} \rangle \), and observe that there exists an epistemic model \( (\mathcal{I}, \mathcal{W}) \) of \( \Phi \) such that \( \gamma(a) \notin \neg A^\mathcal{I} \). Therefore, \( \neg A(a) \) is not a logical consequence of \( \Phi \).

In order to characterize the notion of procedural extension we now introduce the concept of first-order extension of a knowledge base \( \langle \Sigma, \mathcal{R} \rangle \). The first-order extension of \( \Phi = \langle T, A, \mathcal{R} \rangle \) is the knowledge base \( \Sigma_{\mathcal{R}} \) in \( \mathcal{ALC} \), which is the least solution (w.r.t. to set inclusion) of the following equations:

\[
X = \langle T', A' \rangle
\]

where

\[
T' = T \cup \{ \top \leq D \mid \mathbf{KC} \leq D \in \mathcal{R} \text{ and } X \models \top \subseteq C \}
\]

\[
A' = A \cup \{ D(a) \mid \mathbf{KC} \leq D \in \mathcal{R} \text{ and } X \models C(a) \}
\]

We do not delve into the details of the computation of the first-order extension. We simply remark that the solution of the above equations is unique and can be incrementally constructed starting from \( \Sigma \) in a number of steps which is polynomial w.r.t. the size of \( \Phi \).

First-order extensions are linked to the semantics by the following property. Let \( \Phi = \langle \Sigma, \mathcal{R} \rangle \) be a knowledge base in \( \mathcal{ALC}^\mathcal{K} \), let \( (\mathcal{I}, \mathcal{W}) \) be an epistemic model for \( \Phi \), and let \( \Sigma_{\mathcal{R}} \) be the first-order extension of \( \Phi \). Then \( \mathcal{W} \) coincides with the set of models for the knowledge base \( \Sigma_{\mathcal{R}} \) in \( \mathcal{ALC}^\mathcal{C} \). In other words, the result of the forward reasoning process on a knowledge base and set of trigger rules, which is represented by the least solution of the above equations, is correctly captured by the semantics of the knowledge base \( \Phi \), where the trigger rules are expressed as epistemic sentences.

We now show an example of the usage of rules in our framework. Consider the knowledge base \( \Phi = \langle \Sigma, \mathcal{R} \rangle \):

\[
\Sigma = \langle \emptyset, \{ \text{TEACHES(bill, cs248), Grad(bill)} \} \rangle
\]

\[
\mathcal{R} = \{ \mathbf{KGrad} \leq \forall \text{TEACHES.IntCourse} \}.
\]
The first-order extension of $\Phi$ is

$$\Sigma_\mathcal{R} = \langle \emptyset, \{ \text{TEACHES(bill,cs248)} , \text{Grad(bill)}, \forall \text{TEACHES.IntCourse(bill)} \} \rangle.$$ 

Obviously, $\Sigma_\mathcal{R} \models \text{IntCourse(cs248)}$. From the given semantics, one can verify that for every epistemic model $(\mathcal{I}, \mathcal{W})$ for $\Phi$, we have that $\gamma(bill) \in (\forall \text{TEACHES.IntCourse})^\mathcal{I}_\mathcal{W}$ and $\gamma(cs248) \in \text{IntCourse}^\mathcal{I}_\mathcal{W}$, i.e., both the assertions $\forall \text{TEACHES.IntCourse(bill)}$ and $\text{IntCourse(cs248)}$ are logical consequences of $\Phi$, as one would expect.

It is worth noting that the calculus for answering epistemic queries, mentioned in Section 6.2, can be effectively used in the computation of the first-order extension of a knowledge base in $\mathcal{ALCK}$. In fact, the application of a trigger rule $\mathcal{KC} \leq D$ requires to compute the answer set of the query $\mathcal{KC}$, which can be done by means of that calculus.

### 7.2 Integrity Constraints as Epistemic Queries

A further aspect that is usually considered in databases but not in concept-based systems, is that of integrity constraints, which are sentences specifying the set of admissible database states. Differently from rules and TBox-statements, integrity constraints do not play an active role in the query answering process. Conversely, they only specify whether a given knowledge base is admissible or not.

In [Rei90] it is argued that integrity constraints are naturally viewed as epistemic sentences specifying what the knowledge base is supposed to know about a particular aspect of the world, rather than a direct property of the world.

For example, if we want to rule out those knowledge bases which are uncertain about the sex of every person who is known to be student, we cannot simply state that every student has a sex. Rather, we need to specify that for every known student $a$, the knowledge base either knows that $a$ is a male, or knows that $a$ is a female.

In order to put into practice this idea in our setting, we propose to model integrity constraints as $\mathcal{ALCK}$-concepts.

**Definition 7.2.1** An integrity constraint is an $\mathcal{ALCK}$-concept. Given a knowledge base $\Sigma$ in $\mathcal{ALC}$ and an integrity constraint $C$, $\Sigma$ satisfies $C$ if for every individual $a \in \mathcal{O}_\Sigma$, $\Sigma \models C(a)$. If $\mathcal{IC} = \{ C_1, \ldots, C_n \}$ is a set of integrity constraints, then $\Sigma$ is said to be legal with respect to $\mathcal{IC}$ if $\Sigma$ satisfies $C_i$ for each $i \in \{1, \ldots, n\}$.

For example, the previous integrity constraint can be expressed as:

$$(-K\text{Student}) \sqcup (K\text{Male} \sqcup K\neg\text{Male}).$$
Notice that integrity constraints satisfaction (i.e., checking whether a knowledge base is legal w.r.t. a set of integrity constraints) can be easily realized through the calculus presented in Section 6.2.

7.3 Free TBoxes

We have seen in Chapter 2 that free TBoxes are the most expressive type of TBoxes. However, free TBoxes have not been investigated in the literature, and therefore their computational properties are still unknown. In order to make them fully available in concept-based system, a theoretical analysis is needed, to understand at least their decidability.

For this reason, in next section we provide a decidability result for a system employing free TBoxes. In particular, we consider a knowledge base, in the language $\mathcal{ALCN}^R$, composed by a free TBox and an ABox and we show the decidability of reasoning with such a knowledge base.

7.3.1 Decidability of Free TBoxes in $\mathcal{ALCN}^R$

In this section we concentrate on the Consistency problem for a knowledge base in $\mathcal{ALCN}^R$ composed by a free TBox and an ABox. The other reasoning services can be reduced to Consistency (since $\mathcal{ALCN}^R$ includes the constructor $C$ for general negation).

The calculus proposed in Chapter 3 has to be modified, in order to make it work in finite time in our case. In fact, Example 7.3.2 shows that there are cases in which the completion rules can be applied an infinite number of times.

The first modification we make is to assume that the alphabet $\mathcal{V}$ is totally ordered and that variables are introduced in a constraint system according to their ordering. This means that if $y$ is introduced in a constraint system $S$ then $x < y$ for all variables $x$ that are already in $S$.

Given a constraint system $S$ and an object $s$, we define the function $\sigma(\cdot, \cdot)$ as follows: $\sigma(S, s) := \{ C \mid s : C \in S \}$. Moreover, we say that two variables $x$ and $y$ are $S$-equivalent, written $x \equiv_S y$, if $\sigma(S, x) = \sigma(S, y)$. Intuitively, two $S$-equivalent variables can represent the same element in the potential interpretation built by the rules, unless they are separated.

The completion rules are those introduced in Chapter 3, except for the generating ones that are substituted with the following ones:

- $S \rightarrow \{ sRy, y : C \} \cup S$
  
  if $s : \exists R . C$ is in $S$, there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t : C$ is in $S$, and $y$ is a new variable, if $s$ is a variable there is no variable $w$ in $S$ such that $w < s$ and $s \equiv_s w$;
\[ S \rightarrow_2 \{ sRy_1, \ldots, sRy_n \} \cup \{ y_i \neq y_j \mid i, j \in \{1, \ldots, n\}, i \neq j \} \cup S \]

if \( s: (\geq n R) \) is in \( S \), there do not exist \( n \) pairwise separated fillers of \( R \) for \( s \) in \( S \), and \( y_1, \ldots, y_n \) are new variables, if \( s \) is a variable there is no variable \( w \) in \( S \) such that \( w \prec s \) and \( s \equiv w \).

The use of the condition based on the \( S \)-equivalence relation is related to the goal of keeping the constraint system finite even in presence of potentially infinite chains of applications of generating rules. Its role will become clearer in the sequel.

Given a constraint system \( S \), more than one rule might be applicable to it. We define the following strategy for the application of rules:

1. apply a rule to a variable only if no rule is applicable to individuals;
2. apply a rule to a variable \( x \) only if no rule is applicable to a variable \( y \) such that \( y \prec x \);
3. apply generating rules only if no nongenerating rule is applicable.

The above strategy ensures that the variables are processed one at the time according on the ordering \( \prec \).

In the sequel, we assume that rules are always applied according to this strategy and that we always start with a constraint system \( S_\Sigma \) coming from a knowledge base \( \Sigma \) in \( \mathcal{ACN}R \). The following lemma is a direct consequence of these assumptions.

**Lemma 7.3.1 (Stability)** Let \( S \) be a constraint system and \( x \) be a variable in \( S \). Let a generating rule be applicable to \( x \) according to the strategy. Let \( S' \) be any constraint system derivable from \( S \) by any sequence (possibly empty) of applications of rules. Then

1) No rule is applicable in \( S' \) to a variable \( y \) with \( y \prec x \)
2) \( \sigma(S, x) = \sigma(S', x) \)
3) If \( y \) is a variable in \( S \) with \( y \prec x \) then \( y \) is a variable in \( S' \), i.e. the variable \( y \) is not substituted by another variable or by a constant.

**Proof. (Sketch)**

1) By contradiction: Suppose \( S \equiv S_0 \rightarrow_* S_1 \rightarrow_* \cdots \rightarrow_* S_n \equiv S' \), where \( * \in \{ \cup, \cap, \exists, \forall, \geq, \leq, \forall x \} \) and a rule is applicable to a variable \( y \) such that \( y \prec x \) in \( S' \). Then there exists a minimal \( i, i \leq n \), such that this is the case in \( S_i \). Note that \( i \neq 0 \) because of the strategy. So no rule is applicable to any variable \( z \) such that \( z \prec x \) in \( S_0, \ldots, S_{i-1} \). By an exhaustive analysis of all rules we see that—whichever is the rule applied from \( S_{i-1} \) to \( S_i \)—no rule is applicable to \( y \) in \( S_i \), contradicting the assumption.
2) By contradiction: Suppose $\sigma(S, x) \neq \sigma(S', x)$. Call $y$ the direct predecessor of $x$, then a rule must have been applied either to $y$ or to $x$ itself. Obviously we have $y < x$, therefore the former cannot be because of 1). A case analysis shows that the only rules which can have been applied to $x$ are generating ones and the $\rightarrow_\forall$ and the $\rightarrow_\exists$ rules. But these rules add new constraints only to the direct successors of $x$ and not to $x$ itself and therefore do not change $\sigma(\cdot, x)$ 

3) This follows from 1) and the strategy.

Lemma 7.3.1 proves that for a variable $x$ which has a direct successor, $\sigma(\cdot, x)$ is stable, i.e. it will not change because of subsequent applications of rules.

A clash is defined in the usual way. If a completion $S$ contains no clash, we prove that it is always possible to generate a model for $\Sigma$ on the basis of $S$. Before looking at the technical details of the proof, let us consider an example of application of the calculus for checking Consistency.

**Example 7.3.2** Consider the following knowledge base $\Sigma = \langle T, A \rangle$:

$$T = \{\text{Italian} \leq \exists \text{FRIEND}. \text{Italian}\}$$

$$A = \{\text{FRIEND}(\text{peter, susan}),$$
$$\forall \text{FRIEND} . \neg \text{Italian}(\text{peter}),$$
$$\exists \text{FRIEND} . \text{Italian}(\text{susan})\}$$

The corresponding constraint system $S_\Sigma$ is:

$$S_\Sigma = \{\forall x . x : \neg \text{Italian} \cup \exists \text{FRIEND}. \text{Italian},$$
$$\text{peter} \exists \text{FRIEND} \text{susan},$$
$$\text{peter} : \forall \text{FRIEND}. \neg \text{Italian},$$
$$\text{susan} : \exists \text{FRIEND}. \text{Italian}$$
$$\text{peter} \neq \text{susan}\}$$

A sequence of applications of the completion rules to $S_\Sigma$ is as follows:

$$S_1 = S_\Sigma \cup \{\text{susan} : \neg \text{Italian}\} (\rightarrow_\forall \text{-rule})$$

$$S_2 = S_1 \cup \{\text{peter} : \neg \text{Italian} \cup \exists \text{FRIEND}. \text{Italian}\} (\rightarrow_\forall \text{-rule})$$

$$S_3 = S_2 \cup \{\text{susan} : \neg \text{Italian} \cup \exists \text{FRIEND}. \text{Italian}\} (\rightarrow_\forall \text{-rule})$$

$$S_4 = S_3 \cup \{\text{peter} : \neg \text{Italian}\} (\rightarrow_\forall \text{-rule})$$

$$S_5 = S_4 \cup \{\text{peter} \exists \text{FRIEND} . x : \text{Italian}\} (\rightarrow_\exists \text{-rule})$$

$$S_6 = S_5 \cup \{x : \neg \text{Italian} \cup \exists \text{FRIEND}. \text{Italian}\} (\rightarrow_\forall \text{-rule})$$

$$S_7 = S_6 \cup \{x : \exists \text{FRIEND}. \text{Italian}\} (\rightarrow_\exists \text{-rule})$$

$$S_8 = S_7 \cup \{\text{xFRIEND} . y : \text{Italian}\} (\rightarrow_\exists \text{-rule})$$

$$S_9 = S_8 \cup \{y : \neg \text{Italian} \cup \exists \text{FRIEND}. \text{Italian}\} (\rightarrow_\forall \text{-rule})$$
\[ S_{10} = S_9 \cup \{ y \triangleright \text{FRIEND.Italian} \} \ (\rightarrow U\text{-rule}) \]

One can verify that \( S_{10} \) is a complete clash-free constraint system. In particular, \( \rightarrow U \)-rule is not applicable to \( y \). In fact, since \( x \equiv_{S_{10}} y \), its condition is not satisfied. From \( S_{10} \) one can build an interpretation \( \mathcal{I} \), as follows:

\[
\Delta^I = \{ \text{peter, susan, x, y} \}
\]

\[
\begin{align*}
\text{peter}^I &= \text{peter}, \\
\text{italian}^I &= \{ x, y \}
\end{align*}
\]

It is easy to see that \( \mathcal{I} \) is indeed a model for \( \Sigma \). \( \square \)

In order to prove that it is always possible to obtain an interpretation from a complete constraint system we need some additional notions. Let \( S \) be a constraint system and \( x, w \) be variables in \( S \). We call \( w \) a witness of \( x \) in \( S \) if the following conditions hold:

1. \( x \equiv_s w \)
2. \( w \prec x \)
3. there is no variable \( z \) such that \( z \prec w \) and \( z \) satisfies conditions 1. and 2., i.e., \( w \) is the least variable w.r.t. \( \prec \) satisfying conditions 1. and 2.

We say \( x \) is blocked (by \( w \)) if \( x \) has a witness (\( w \)) in \( S \).

The following lemma states a property of witnesses.

**Lemma 7.3.3** Let \( S \) be a constraint system, \( x \) a variable in \( S \). If \( x \) has a witness then (i) \( x \) has no direct successor and (ii) \( x \) has exactly one witness.

**Proof.** (i) By contradiction: Suppose that \( x \) is blocked in \( S \) and \( xPy \) is in \( S \). During the completion process leading to \( S \) a generating rule must have been applied to \( x \) in a system \( S' \). It follows from the definition of the rules, that in \( S' \) for every variable \( w \prec x \) we had \( x \not\equiv_{S'} w \). Now from Lemma 7.3.1 we know, that for the constraint system \( S \) derivable from \( S' \) and for every \( w \prec x \) in \( S \) we also have \( x \not\equiv_s w \). Hence there is no witness for \( x \) in \( S \), contradicting the hypothesis that \( x \) is blocked. (ii) follows directly from condition 3. for a witness. \( \square \)

As a consequence of Lemma 7.3.3, in a constraint system \( S \), if \( w_1 \) is a witness of \( x \) then \( w_1 \) cannot have a witness itself, since both the relations ‘\( \prec \)’ and \( S \)-equivalence are transitive. The uniqueness of witness for a blocked variable is important for defining the following particular interpretation out of \( S \).

Let \( S \) be a constraint system. We define the canonical interpretation \( \mathcal{I}_S \) and the canonical \( \mathcal{I}_S \)-assignment \( \alpha_S \) as follows:

---

**CHAPTER 7**
1. $\Delta^I_S := \{ s \mid s \text{ is an object in } S \}$

2. $\alpha_S(s) := s$

3. $s \in A^I_S$ if and only if $s: A$ is in $S$

4. $(s, t) \in P^I_S$ iff

   (a) $sPt$ is in $S$ or

   (b) $s$ is a blocked variable, $w$ is the witness of $s$ in $S$ and $wPt$ is in $S$.

We call $(s, t)$ a $P$-role-pair of $s$ in $I_S$ if $(s, t) \in P^I_S$, we call $(s, t)$ a role-pair of $s$ in $I_S$ if $(s, t)$ is a $P$-role-pair for some role $P$. We call a role-pair explicit if it comes up from case 4.(a) of the definition of the canonical interpretation and we call it implicit if it comes up from case 4.(b).

From Lemma 7.3.3 it is obvious that a role-pair cannot be both explicit and implicit. Moreover, if a variable has an implicit role-pair then all its role-pairs are implicit and they all come from exactly one witness, as stated by the following lemma.

**Lemma 7.3.4** Let $S$ be a completion and $x$ a variable in $S$. Let $I_S$ be the canonical interpretation for $S$. If $x$ has an implicit role-pair $(x, y)$, then

1. all role-pairs of $x$ in $I_S$ are implicit

2. there is exactly one witness $w$ of $x$ in $S$ such that for all roles $P$ in $S$ and all $P$-role-pairs $(x, y)$ of $x$ the constraint $wPy$ is in $S$.

**Proof.** Point 1 follows from (i) of Lemma 7.3.3 and point 2 follows from (ii) of Lemma 7.3.3 together with the definition of $I_S$. $\square$

We have now all the machinery needed to prove the main theorem of this subsection.

**Theorem 7.3.5** Let $S$ be a complete constraint system. If $S$ contains no clash then it is satisfiable.

**Proof.** Let $I_S$ and $\alpha_S$ be the canonical interpretation and canonical $I$-assignment for $S$. We prove that the pair $(I_S, \alpha_S)$ satisfies every constraint $c$ in $S$. If $c$ has the form $sPt$ or $s \not\in t$, then $(I_S, \alpha_S)$ satisfies them by definition of $I_S$ and $\alpha_S$. Considering the $\rightarrow_{\geq}$ -rule and the $\rightarrow_{\leq}$ -rule we see that a constraint of the form $s \not\in s$ can not be in $S$. If $c$ has the form $s: C$, we show by induction on the structure of $C$ that $s \in C^I_S$.

We first consider the base cases. If $C$ is a concept name, then $s \in C^I_S$ by definition of $I_S$. If $C = \top$, then obviously $s \in T^I_S$. The case that $C = \bot$ cannot occur since $S$ is clash-free.
Next we analyze in turn each possible complex concept $C$. If $C$ is of the form $\neg C_1$ then $C_1$ is a concept name since all concepts are simple. Then the constraint $s : C_1$ is not in $S$ since $S$ is clash-free. Then $s \notin C_1^{T_S}$, that is, $s \in \Delta^{T_S} \setminus C_1^{T_S}$. Hence $s \in (\neg C_1)^{T_S}$.

If $C$ is of the form $C_1 \cap C_2$ then (since $S$ is complete) $s : C_1$ is in $S$ and $s : C_2$ is in $S$. By induction hypothesis, $s \in C_1^{T_S}$ and $s \in C_2^{T_S}$. Hence $s \in (C_1 \cap C_2)^{T_S}$.

If $C$ is of the form $C_1 \cup C_2$ then (since $S$ is complete) either $s : C_1$ is in $S$ or $s : C_2$ is in $S$. By induction hypothesis, either $s \in C_1^{T_S}$ or $s \in C_2^{T_S}$. Hence $s \in (C_1 \cup C_2)^{T_S}$.

If $C$ is of the form $\forall R.D$, we have to show that for all $t$ with $(s, t) \in R^{T_S}$ it holds that $t \in D^{T_S}$. If $(s, t) \in R^{T_S}$, then according to Lemma 7.3.4 two cases can occur. Either $t$ is an $R$-successor of $s$ in $S$ or $s$ is blocked by a witness $w$ in $S$ and $t$ is an $R$-successor of $w$ in $S$. In the first case $t : D$ must also be in $S$ since $S$ is complete. Then by induction hypothesis we have $t \in D^{T_S}$. In the second case by definition of witness, $w : \forall R.D$ is in $S$ and then because of completeness of $S$, $t : D$ must be in $S$. By induction hypothesis we have again $t \in D^{T_S}$.

If $C$ is of the form $\exists R.D$ we have to show that there exists a $t \in \Delta^{T_S}$ with $(s, t) \in R^{T_S}$ and $t \in D^{T_S}$. Since $S$ is complete either there is a $t$ that is an $R$-successor of $s$ in $S$ and $t : D$ is in $S$ or $s$ is a variable blocked by a witness $w$ in $S$. In the first case by induction hypothesis and the definition of $T_S$ we have $t \in D^{T_S}$ and $(s, t) \in R^{T_S}$. In the second case $w : \exists R.D$ is in $S$. Since $w$ cannot be blocked and $S$ is complete, we have that there is a $t$ that is an $R$-successor of $w$ in $S$ and $t : D$ is in $S$. So by induction hypothesis we have $t \in D^{T_S}$ and by the definition of $T_S$ we have $(s, t) \in R^{T_S}$.

If $C$ is of the form $(\leq n R)$ we show the goal by contradiction. Assume that $s \notin (\leq n R)^{T_S}$. Then there exist at least $n + 1$ distinct objects $t_1, \ldots , t_{n+1}$ with $(s, t_i) \in R^{T_S}$, $i \in 1..n + 1$. This means that, since $R = P_1 \cap \ldots \cap P_k$ there are pairs $(s, t_i) \in P_j^{T_S}$, where $i \in 1..n + 1$, $j \in 1..k$. Then according to Lemma 7.3.4 one of the two following cases must occur. Either all $s P_j t_i$ for $j \in 1..k$, $i \in 1..n + 1$ are in $S$ or there exists a witness $w$ of $s$ in $S$ with all $w P_j t_i$ for $j \in 1..k$, $i \in 1..n + 1$ are in $S$. In the first case the $\rightarrow \leq \cdot$-rule can not be applicable because of completeness. This means that all the $t_i$'s are pairwise separated, i.e., that $S$ contains the constraints $t_i \neq t_j$, $i, j \in 1..n + 1, i \neq j$. This contradicts the fact that $S$ is clash-free. And the second case leads to an analogous contradiction.

If $C$ is of the form $(\geq n R)$ we show the goal by contradiction. Assume that $s \notin (\geq n R)^{T_S}$. Then there exist at least $m \leq n$ ($m$ possibly 0) distinct objects $t_1, \ldots , t_m$ with $(s, t_i) \in R^{T_S}$, $i \in 1..m$. We have to consider two cases. First case: $s$ is not blocked in $S$. Since there are only $m$ $R$-successors of $s$ in $S$, the $\rightarrow \geq \cdot$-rule is applicable to $s$. This contradicts the fact that $S$ is complete. Second case: $s$ is blocked by a witness $w$ in $S$. Since there are $m$
7.3 Free TBoxes

R-successors of \( w \) in \( S \), the \( \rightarrow \) -rule is applicable to \( w \). But this leads to the same contradiction.

If \( e \) has the form \( \forall x.x:D \) then, since \( S \) is complete, for each object \( t \) in \( S \), \( t:D \) is in \( S \)—and, by the previous cases, \( t \in D^I \). Therefore, the pair \((I_S, \alpha_S)\) satisfies \( \forall x.x:D \). Finally, since \((I_S, \alpha_S)\) satisfies all constraints in \( S \), \((I_S, \alpha_S)\) satisfies \( S \).

\[ \square \]

**Theorem 7.3.6 (Correctness)** A constraint system \( S \) is satisfiable if and only if there exists at least one clash-free completion of \( S \).

**Proof.** \( \Leftarrow \) Follows immediately from Theorem 7.3.5. \( \Rightarrow \) Clearly, a system containing a clash is unsatisfiable. If every completion of \( S \) is unsatisfiable, then from Proposition 3.1.2 \( S \) is unsatisfiable. \[ \square \]

**Lemma 7.3.7** Let \( S \) be a constraint system, let \( n \) be the number of concepts appearing in \( S \) (including subconcepts), and let \( S' \) be derived from \( S \) by means of the completion rules. In any set of variables in \( S' \) including more than \( 2^n \) variables there are at least two variables \( x, y \) such that \( x \equiv y \).

**Proof.** Each constraint \( x:C \in S' \) may contain only concepts of the constraint system \( S \). Since there are \( n \) such concepts, given a variable \( x \) there cannot be more than \( 2^n \) different sets of constraints \( x:C \) in \( S' \).

**Lemma 7.3.8** Let \( S \) be a constraint system, let \( n \) be the number of concepts in \( S \), and let \( S' \) be any constraint system derived from \( S \) by applying the completion rules with the given strategy. Then, in \( S' \) there are at most \( 2^n \) non-blocked variables.

**Proof.** Suppose there are \( 2^n + 1 \) non-blocked variables. From Lemma 7.3.7, we know that in \( S' \) there are at least two variables \( y_1, y_2 \) such that \( y_1 \equiv y_2 \). Obviously either \( y_1 < y_2 \) or \( y_2 < y_1 \) holds; suppose (without loss of generality) that \( y_1 < y_2 \). From the definitions of witness and blocked either \( y_1 \) is a witness of \( y_2 \) or there exists a variable \( y_3 \) such that \( y_3 < y_1 \) and \( y_3 \) is a witness of \( y_2 \). In both cases \( y_2 \) is blocked, contradicting the hypothesis. \[ \square \]

**Theorem 7.3.9 (Termination and space complexity)** Let \( \Sigma \) be a knowledge base in \( ALCNR \) and let \( n \) be its size. Every completion of \( S_\Sigma \) is finite and its size is \( O(2^n) \).

**Proof.** Let \( S \) be a completion of \( S_\Sigma \). From Lemma 7.3.8 it follows that there are at most \( 2^n \) non-blocked variables in \( S \). Therefore there are at most \( m \times 2^n \)
total variables in \( S \), where \( m \) is the maximum number of direct successors for a variable in \( S \).

Observe that \( m \) is bounded by the number of \( \exists R. C \) (sub)concepts (at most \( n \)) plus the sum of all numbers appearing in number restrictions. Since these numbers are expressed in binary, their sum is bounded by \( 2^n \). Hence, \( m \leq 2^n + n \). Since the number of individuals is also bounded by \( n \), the total number of objects in \( S \) is at most \( m \times (2^n + n) \leq (2^n + n) \times (2^n + n) \), that is, \( O(2^{2n}) \).

The number of different constraints of the form \( s : C, \forall x.x: C \) in which each object \( s \) can be involved is bounded by \( n \), and each constraint has size linear in \( n \). Hence, the total size of these constraints is bounded by \( n \times n \times 2^n \), that is \( O(2^{3n}) \).

The number of constraints of the form \( sPt, s \neq t \) is bounded by \((2^n)^2 = 2^{4n}\), and each constraint has constant size.

In conclusion, we have that the size of \( S \) is \( O(2^{4n}) \). \( \square \)

Notice that the above one is just a coarse upper bound, obtained for theoretical purposes. In practical cases we expect the actual size to be much smaller than that. For example, if the numbers involved in number restrictions were either expressed in unary notation, or limited by a constant (the latter being a reasonable restriction in practical systems) then an argumentation analogous to the above one would lead to a bound of \( 2^{3n} \).

**Theorem 7.3.10 (Decidability)** Given a knowledge base \( \Sigma \) in \( \mathcal{ACLNR} \), checking whether \( \Sigma \) is consistent is a decidable problem.

**Proof.** This follows from Theorems 7.3.6 and 7.3.9 and the fact that \( \Sigma \) is consistent if and only if \( S_\Sigma \) is satisfiable. \( \square \)

We can refine the above theorem, by giving tighter bounds on the time required to decide satisfiability.

**Theorem 7.3.11 (Time complexity)** Given a knowledge base \( \Sigma \) in \( \mathcal{ACLNR} \), checking whether \( \Sigma \) is consistent can be done in nondeterministic exponential time.

**Proof.** In order to prove the claim it is sufficient to show that each completion is obtained with an exponential number of applications of rules. Since the number of constraints of each completion is exponential (Theorem 7.3.9) and each rule, but the \( \rightarrow_{\leq} \)-rule, adds new constraints to the constraint system, it follows that all such rules are applied at most an exponential number of times. Regarding the \( \rightarrow_{\leq} \)-rule, it is applied for each object at most as many times as the number of its direct successors. Since such number is at most exponential
(if numbers are coded in binary) w.r.t. the size of the knowledge base, the claim follows.

A lower bound of the complexity of Consistency is obtained exploiting previous results about the language $\mathcal{ALC}$, which is a sublanguage of $\mathcal{ALCN}^R$ that does not include number restrictions and role conjunction. We know from McAllester [McA91], and (independently) from an observation of Werner Nutt [Nut92] that Consistency in knowledge bases in $\mathcal{ALC}$ is EXPTIME-hard and hence it is hard for knowledge bases in $\mathcal{ALCN}^R$, too. Hence, we do not expect to find any algorithm solving the problem in polynomial space, unless PSPACE=EXPTIME.

The calculus proposed above improves the one we proposed in [BDS93b, Sec. 4], which works in exponential space but may require double exponential time. Nevertheless, our calculus still requires nondeterministic exponential time. We are currently working to devise an algorithm working in deterministic exponential time.

Notice that, since the domain of the canonical interpretation $\Delta^I_S$ is always finite, we have also implicitly proved that knowledge bases in $\mathcal{ALCN}^R$ have the finite model property, i.e., any consistent knowledge base has a finite model. This property has been extensively studied in modal logics [HC84] and dynamic logics [Har84]. In particular, a technique called filtration, has been developed both to prove the finite model property and to build a finite model for a satisfiable formula. This technique allows one to build a finite model from an infinite one by grouping the worlds of a structure in equivalence classes based on the set of formulae that are satisfied in each world. It is interesting to observe that our calculus, based on witnesses, can be considered as a variant of the filtration technique where the equivalence classes are determined on the basis of our $S$-equivalence relation. However, because of number restrictions, variables that are $S$-equivalent cannot be grouped, since they might be separated (e.g., they might have been introduced by the same application of the $\rightarrow_2$-rule). Nevertheless, they can have the same direct successors, as stated in point 4.(b) of the definition of canonical interpretation on page 141. This would correspond to grouping variables of an infinite model in such a way that separations are preserved.

### 7.3.2 Free TBoxes versus General TBoxes

Now we compare the expressive power of TBoxes defined as a set of inclusions and equations and TBoxes defined as a set of (possibly cyclic) concept definitions and specifications (i.e. statements of the form $A \subseteq D$ and $A \equiv D$).

Unlike [Baa90a] and [Sch91], we consider reasoning problems dealing with TBox and ABox together. The result we have obtained is that free TBoxes
and general TBoxes in \( \mathcal{ALCN}\mathcal{R} \) actually have the same expressive power (provided that we use the descriptive semantics). In detail, we show that the consistency of a knowledge base \( \Sigma = \langle A, T \rangle \), where \( T \) is a set of inclusion and equation statements, can be reduced to the consistency of a knowledge base \( \Sigma' = \langle A', T' \rangle \) such that \( T' \) is a set of concept definitions and specifications.

We use the sake of simplicity, we suppose equations of the form \( C \doteq D \) to be expressed as the following pair of inclusions \( C \doteq D \) and \( D \doteq C \).

As a notation, given a TBox \( T = \{ C_1 \doteq D_1, \ldots, C_n \doteq D_n \} \), we define the concept \( C_T \) as \( C_T = (\neg C_1 \cup D_1) \cap \cdots \cap (\neg C_n \cup D_n) \). As pointed out in [Baa90a] for \( \mathcal{ALC} \), an interpretation satisfies a TBox \( T \) if and only if it satisfies the equation \( C_T = \top \). This result easily extends to \( \mathcal{ALCN}\mathcal{R} \), as stated in the following proposition.

**Proposition 7.3.12** Given a TBox \( T = \{ C_1 \doteq D_1, \ldots, C_n \doteq D_n \} \), an interpretation \( I \) satisfies \( T \) if and only if it satisfies the equation \( C_T = \top \).

**Proof.** An interpretation \( I \) satisfies an inclusion \( C \doteq D \) if and only if it satisfies the equation \( \neg C \cup D \doteq \top \); \( I \) satisfies the set of equations \( \neg C_1 \cup D_1 = \top \), \ldots, \( \neg C_n \cup D_n = \top \) if and only if \( I \) satisfies \( (\neg C_1 \cup D_1) \cap \cdots \cap (\neg C_n \cup D_n) = \top \). The claim follows.

Given a knowledge base \( \Sigma = \langle A, T \rangle \) and a concept \( A \) not appearing in \( \Sigma \), we define the knowledge base \( \Sigma' = \langle A', T' \rangle \) as follows:

\[
\begin{align*}
A' &= A \cup \{ A(b) \mid b \text{ is an individual in } \Sigma \} \\
T' &= \{ A \doteq C_T \cap \forall P_1 A \cap \cdots \cap \forall P_n A \}
\end{align*}
\]

where \( P_1, P_2, \ldots, P_n \) are all the role names appearing in \( \Sigma \). Note that \( T' \) has a single inclusion, which could be also thought of as one primitive concept specification.

**Theorem 7.3.13** \( \Sigma = \langle A, T \rangle \) is satisfiable iff \( \Sigma' = \langle A', T' \rangle \) is satisfiable.

**Proof.** In order to simplify the machinery of the proof, we will use for \( T' \) the following (logically equivalent) form:

\[
T' = \{ A \doteq C_T, A \doteq \forall P_1 A, \ldots, A \doteq \forall P_n A \}
\]

"\doteq" Suppose \( \Sigma = \langle A, T \rangle \) satisfiable. From Theorem 7.3.6, there exists a complete constraint system \( S \) without clash, which defines a canonical interpretation \( I_S \) which is a model of \( \Sigma \). Define the constraint system \( S' \) as follows:

\[ S' = S \cup \{ w \colon A \mid w \text{ is an object in } S \} \]
and call $\mathcal{I}_{S'}$ the canonical interpretation associated to $S'$. We prove that $\mathcal{I}_{S'}$ is a model of $\Sigma'$.

First observe that every assertion in $\mathcal{A}$ is satisfied by $\mathcal{I}_{S'}$ since $\mathcal{I}_{S'}$ is equal to $\mathcal{I}_S$ except for the interpretation of $A$, and $A$ does not appear in $\mathcal{A}$. Therefore, every assertion in $\mathcal{A}'$ is also satisfied by $\mathcal{I}_{S'}$, either because it is an assertion of $\mathcal{A}$, or (if it is an assertion of the form $A(b_i)$) by definition of $S'$.

Regarding $\mathcal{T}'$, note that by definition of $S'$, we have $A^{T_{S'}} = \Delta^{T_{S'}} = \Delta^{T_S}$; therefore both sides of the inclusions of the form $A \preceq \forall P_i A$ ($i = 1, \ldots, n$) are interpreted as $\Delta^{T_{S'}}$, hence they are satisfied by $\mathcal{I}_{S'}$. Since $A$ does not appear in $C_T$, we have that $(C_T)^{T_{S'}} = (C_T)^{T_S}$. Moreover, since $\mathcal{I}_S$ satisfies $\mathcal{T}$, we also have, by Proposition 7.3.12, that $(C_T)^{T_{S'}} = \Delta^{T_{S'}}$, therefore $(C_T)^{T_{S'}} = (C_T)^{T_S} = \Delta^{T_S} = \Delta^{T_{S'}}$. It follows that also both sides of the definition $A \preceq C_T$ are interpreted as $\Delta^{T_{S'}}$. In conclusion, $\mathcal{I}_{S'}$ satisfies $\mathcal{T}'$.

"⇒" Suppose $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ satisfiable. Again, because of Theorem 7.3.6, there exists a complete constraint system $S'$ without clash, which defines a canonical interpretation $\mathcal{I}_{S'}$ which is a model of $\Sigma'$. We show that $\mathcal{I}_{S'}$ is also a model of $\Sigma$.

First of all, the assertions in $\mathcal{A}$ are satisfied because $\mathcal{A} \subseteq \mathcal{A}'$, and $\mathcal{I}_{S'}$ satisfies every assertion in $\mathcal{A}'$. To prove that $\mathcal{I}_{S'}$ satisfies $\mathcal{T}$, we first prove the following equation:

$$A^{T_{S'}} = \Delta^{T_{S'}}$$

Equation 7.1 is proved by showing that, for every object $s \in \Delta^{T_{S'}}$, $s$ is in $A^{T_{S'}}$.

In order to do that, observe a general property of constraint systems: every variable in $S'$ is a successor of an individual. This comes by the definition of the generating rules, which add variables to the constraint system only as direct successors of existing objects, and at the beginning $S_{\Sigma'}$ contains only individuals.

Then, the above equation is proved by observing the following three facts:

1. for every individual $b$ in $\Delta^{T_{S'}}$, $b \in A^{T_{S'}}$;

2. if an object $s$ is in $A^{T_{S'}}$, then because $\mathcal{I}_{S'}$ satisfies the inclusions $A^{T_{S'}} \subseteq (\forall P_1 A)^{T_{S'}}$,
   $$\ldots, A^{T_{S'}} \subseteq (\forall P_n A)^{T_{S'}}$$, every direct successor of $s$ is in $A^{T_{S'}}$;

3. the successor relation is closed under the direct successor relation

From the Fundamental Theorem on Induction (e.g. cfr. [Wan80, page 41]) we conclude that every object $s$ of $\Delta^{T_{S'}}$ is in $A^{T_{S'}}$. This proves that Equation 7.1 holds.

From Equation 7.1, and the fact that $\mathcal{I}_{S'}$ satisfies the inclusion $A^{T_{S'}} \subseteq (C_T)^{T_{S'}}$, we derive that $(C_T)^{T_{S'}} = \Delta^{T_{S'}}$, that is $\mathcal{I}_{S'}$ satisfies the equation
$C_T \equiv T$. Hence, from Proposition 7.3.12, $I_{S^T}$ satisfies $T$, and this completes the proof of the theorem.

The machinery present in this proof is not new. In fact, realizing that the inclusions $A \subseteq \forall P_1.A, \ldots, A \subseteq \forall P_n.A$ simulate a transitive closure on the roles $P_1, \ldots, P_n$, one can recognize similarities with the proofs given by Schild [Sch91] and Baader [Baa90a]. The difference is that their proofs rely on the notion of connected model (Baader uses the equivalent notion of rooted model). In contrast, the models we obtain are not connected, when the individuals in the knowledge base are not. What we exploit is the weaker property that every variable in the model is a successor of an individual.

Note that the above reduction strongly relies on the fact that disjunction $\cup$ and negation $\neg$ are within the language. In fact, disjunction and negation are necessary in order to express all the inclusions of a TBox $T$ inside the concept $C_T$. Therefore, the proof holds for knowledge bases in $\mathcal{ALC}$, but does not hold for languages not allowing for these constructors of concepts (e.g., $\text{BACK}$).

Furthermore, for the language $\mathcal{FL}_0$ introduced in Section 3.2.2, the opposite result holds. In fact, in [McA91], it is proved that computing Subsumption w.r.t. a set of inclusions is $\text{EXPTIME}$-hard, even in the small language $\mathcal{FL}_0$. Conversely, Nebel in [Neb91], proves that Subsumption w.r.t. a set of cyclic definitions in $\mathcal{FL}_0$ can be done in $\text{PSPACE}$. Combining the two results, we can conclude that for $\mathcal{FL}_0$ Subsumption w.r.t. a set of inclusions and Subsumption w.r.t. a set of definitions are in different complexity classes (assuming $\text{EXPTIME} \neq \text{PSPACE}$), hence inclusion statements are strictly more expressive than concept definitions in $\mathcal{FL}_0$.

It is still open whether inclusions and definitions are equivalent in languages whose expressivity is between $\mathcal{FL}_0$ and $\mathcal{ALC}$.

### 7.4 Weak Inclusions as Rules

Due to its complexity, the treatment of inclusion is one of the critical aspects of the implementation of knowledge representation systems based on concept languages.

This problem is addressed for example in $\text{LOOM}$ [Mac88] by adopting a weak form of inclusion, which applies only to known individuals and disregards many inferences based on the use of contrapositives.

In this section we argue that the class of epistemic sentences used in the formalization of trigger rules can be regarded as a form of weak inclusion which may lead to significant computational advantages in comparison to inclusion statements.

To this purpose we introduce the notion of weakening of an knowledge
base in $\mathcal{ALCK}$, which is the knowledge base in $\mathcal{ALCK}$ obtained by replacing every inclusion statement $C \preceq D$ by the epistemic statement $\text{KC} \preceq D$. More formally, let $\Phi = \langle \Sigma, \text{R} \rangle$ be an knowledge base as defined in the previous section. The weakening of $\Phi$ is the knowledge base

$$\Phi^- = \langle \Sigma', \text{R}' \rangle$$

where

$$\Sigma' = \langle \emptyset, \text{A} \rangle$$

and

$$\text{R}' = \text{R} \cup \{ \text{KC} \preceq D | (C \preceq D) \in T \}.$$  

Intuitively, every inference we can make in $\Phi^-$ can be done in $\Phi$ as well, while the converse of course is not true. Hence, $\Phi^-$ can be regarded as a sound approximation of $\Phi$, where the lost inferences are traded with a gain in the efficiency of reasoning. In Chapter 8, we present an example of the weakening transformation.

Let us now consider the computational advantages of weakening an knowledge base in $\mathcal{ALCK}$. In order to show such advantages, consider the knowledge base $\Phi = \langle \Sigma, \text{R} \rangle$, where $\Sigma = \langle T, \text{A} \rangle$, and let $\Phi^- = \langle \Sigma', \text{R}' \rangle$, where $\Sigma' = \langle \emptyset, \text{A} \rangle$, be its weakening. Furthermore, assume that no rule in $\text{R}'$ has an antecedent which is equivalent to $T$.

Based on the result in [McA91] query answering in $\Phi$ is EXPTIME-hard. On the other hand, query answering in $\Phi^-$ amounts to solving the same problems in $\Sigma'_{\text{R}'}$, which is the first-order extension of $\Sigma' = \langle \emptyset, \text{A} \rangle$ w.r.t. $\text{R}'$. Observing that $\Sigma'_{\text{R}'}$ is a knowledge base constituted by an ABox only, we know from Section 6.2 that this problem can be solved in polynomial space. Since the size of $\Sigma'_{\text{R}'}$ is polynomially related to the size of $\Phi^-$, and therefore of $\Phi$ too, the above observation shows that weakening the inclusions of a knowledge base in $\mathcal{ALCK}$ leads to an exponential decrease of the space required for query answering.

We can conclude that the notion of weakening proposed here provides a form of incomplete reasoning that is both computationally advantageous and semantically well-founded.
Chapter 8

Design of a Powerful Concept-Based KR System

In Chapters 2 and 7 we have discussed the various representational mechanisms. We have noticed that the mechanisms, offering different expressivity, require different computational resources according with the expressivity/tractability trade-off.

In this chapter, we propose a concept-based system, called AL-K equipped with a highly expressive language for the definition of the knowledge base, and we provide the user with a query language enhanced with the epistemic operator defined in Chapter 6. Moreover, the system allows the user to express integrity constraints, trigger rules, and concept inclusions and equations.

The philosophy underlying AL-K is to give the user the ability to choice between tractable and intractable mechanisms, based on her/his needs. The challenge of AL-K is that of being able to work in polynomial time in the tractable cases and to work efficiently in the average intractable cases.

The example in Section 8.2 is mostly taken from [DLN+93].

8.1 Definition of the System AL-K

The concept language we use for the knowledge base definition is $\mathcal{ALCNRIOT}$, that we call $\mathcal{DDL}$ for simplicity, while our query language is $\mathcal{DDL}$ augmented with the $\mathcal{K}$ operator, that we call $\mathcal{EQL}$.

A concept-based knowledge base $\Sigma$ is a quadruple $\langle T, A, R, IC \rangle$ where $T$ is a free TBox in $\mathcal{DDL}$, $A$ is an ABox in $\mathcal{DDL}$, $R$ is a set of trigger rules in $\mathcal{EQL}$, and $IC$ is a set of concepts in $\mathcal{EQL}$, the integrity constraints.

The semantics of $\Sigma$ is obtained composing the definitions given in previous chapters. Since the semantics associated with concept languages is an open world semantics, the answer to a query $Q$ will be $\text{YES}$ if $Q$ is true in every
model for \( \Sigma \), **NO** if \( Q \) is false in every model, and **UNKNOWN** otherwise.

### 8.2 Example

In Figure 8.1 we show a knowledge base \( \Sigma = (T, \mathcal{A}, \emptyset, IC) \) in DDLC describing information about a university. The TBox \( T \) contains information about the various classes of persons working in the university and the courses supplied by the university. The ABox \( \mathcal{A} \) keeps track of the actual persons and courses involved in the university, together with the relations between them. The ABox \( \mathcal{A} \) is also shown in graph form in Figure 8.2. The integrity constraints specify the legal states of the knowledge base. For example, the first one imposes the condition that every professor must teach at least one course.

It can be easily shown that \( \Sigma \) is satisfiable and that it has several different models. In fact, it does not have complete knowledge about the represented world. For example, since EE282 is an intermediary course, \( \Sigma \) knows that at least one graduate student is enrolled in EE282, but it doesn’t know who she/he actually is. Similarly, \( \Sigma \) knows that Susan is either a graduate or an undergraduate, without knowing which one.

Notice that the information in \( T \) plays a role in the deduction of properties of individuals in \( \mathcal{A} \). For example, \( \Sigma \) knows that Mary is a graduate student, because she has a bachelor’s degree and thus, according to \( T \), she falls under the description of graduate student.

#### 8.2.1 Epistemic Operator in the Query Language

Our goal here is to show that the use of epistemic operators in queries allows for a more sophisticated interaction with the a knowledge representation system. For this purpose we discuss various kinds of queries that can be posed to it using the language \( \mathcal{EQL} \). In particular, in order to understand the role of the epistemic operator \( K \), we consider both DDLC queries and modified versions of them including \( K \). The comparison between their respective semantics highlights the role of \( K \) in the query language.

We start with a pair of queries involving one single existential quantifier:

- **Query 1a**: \( \Sigma \models \exists \text{ENROLLED}.\text{Grad}(\text{ee282}) \)
  
  **Answer**: **YES**.

- **Query 1b**: \( \Sigma \models \exists \text{KENROLLED}.\text{KGrad}(\text{ee282}) \)
  
  **Answer**: **NO**.

Query 1a asks whether there is a graduate student enrolled in EE282. The answer is **YES** because EE282 is an intermediary course and therefore, according to \( T \), there is at least one graduate student enrolled in it. However, as we already mentioned, the name of the enrolled student is unknown. It might either be one of the individuals named in \( \Sigma \) or a different one about
8.2 Example

\[
\begin{align*}
\text{AdvCourse} & \triangleq \text{Course} \sqcap \forall \text{ENROLLED.Grad}, \\
\text{IntCourse} & \triangleq \text{Course} \sqcap \forall \text{ENROLLED.Undergrad}, \\
\text{MedCourse} & \triangleq \text{Course} \sqcap \exists \text{ENROLLED.Grad} \sqcap \exists \text{ENROLLED.Undergrad}, \\
\exists \text{TEACHES.Course} & \subseteq \text{Grad} \sqcup \text{Professor}, \\
\text{Grad} & \triangleq \text{Student} \sqcap \exists \text{DEGREE.Bachelor}, \\
\text{Undergrad} & \triangleq \text{Student} \sqcap \neg \text{Grad}
\end{align*}
\]

The TBox \( \mathcal{T} \)

\[
\begin{align*}
\text{Professor(bob), TEACHES(bob, ee282), TEACHES(john, cs324),} \\
\text{TEACHES(john, cs221), Course(cs221), Course(cs324),} \\
\text{MedCourse(ee282), ENROLLED(ee282, peter), ENROLLED(cs221, mary),} \\
\text{ENROLLED(cs221, susan), ENROLLED(cs324, susan),} \\
\text{ENROLLED(cs324, peter), Undergrad(peter), Student(susan),} \\
\text{Student(mary), DEGREE(mary, bs), Bachelor(bs)}
\end{align*}
\]

The ABox \( \mathcal{A} \)

\[
\begin{align*}
\neg K\text{Professor} & \sqcup K\exists \text{TEACHES.Course}, \\
\neg K\text{Course} & \sqcup K\exists \text{ENROLLED.Student}
\end{align*}
\]

The set of integrity constraints \( \mathcal{IC} \)

Figure 8.1: The knowledge base example
Figure 8.2: A pictorial representation of the ABox of the knowledge base example

whom no information is given. Moreover, it is not even ensured that it is the same one in all models.

On the other hand, Query 1b asks whether there exists an individual who is known both to be enrolled in EE282 and to be a graduate student. In other words, it asks for an individual, say fred, such that both the assertions ENROLLED(ee282, fred) and Grad(fred) hold in every model for Σ. Such an individual does not exist, thus the answer to the query is NO.

The next pair of queries shows the interaction of the epistemic operator with the disjunction constructor:

- Query 2a: Σ ⊨ Grad ∨ Professor(john) ?  
  Answer: YES.

- Query 2b: Σ ⊨ KGrad ∨ KProfessor(john) ?  
  Answer: NO.

Query 2a asks whether John is either a graduate student or a professor. The answer is YES, and it can be derived by the fact that it is stated in the ABox that he teaches two courses, and, according to the TBox, everybody who teaches at least one course is either a graduate student or a professor.

Query 2b, instead, asks whether he is either known to be a graduate student or known to be a professor. It is easy to verify that none of them is true and therefore the answer to this query is NO.

We consider now queries that involve also universal quantifiers:
• Query 3a: $\Sigma \models \forall T E A C H E S . (M e d C o u r s e \sqcup \neg C o u r s e)(b o b)$ ?
  Answer: UNKOWN.

• Query 3b $\Sigma \models \exists T E A C H E S . K (M e d C o u r s e \sqcup \neg C o u r s e)(b o b)$ ?
  Answer: YES.

Query 3a asks whether every course taught by Bob is an intermediary one. The answer is UNKOWN because there are models for $\Sigma$ in which Bob teaches only intermediary courses as well as models in which he teaches also courses that are not intermediary.

Query 3b, instead, asks whether everything that is known to be taught by Bob is also known to be either an intermediary course or not to be a course. Since the only thing taught by Bob is EE282, and it is indeed an intermediary course, the answer to Query 3b is YES.

In the above example the addition of $K$ has changed the answer from UNKOWN to YES. Notice that it is also possible that Query 3a could be answered NO and Query 3b still be answered YES: Suppose that the assertion $\exists T E A C H E S . A d v C o u r s e (b o b)$ is added to $\Sigma$ and then the same queries are asked. Query 3a now gets the answer NO, because $A d v C o u r s e$ and $M e d C o u r s e$ are disjoint concepts. However, the set of known courses taught by Bob is not changed, and therefore the answer to Query 3b is still YES.

We now consider some queries involving nested quantifiers: Queries 4a and 4b involve two levels of existential quantification. The innermost quantifier is carried by the concept $M e d C o u r s e$, which has existential quantifiers in its definition in $T$.

• Query 4a: $\Sigma \models \exists T E A C H E S . M e d C o u r s e (j o h n)$ ?
  Answer: YES.

• Query 4b: $\Sigma \models \exists K T E A C H E S . K M e d C o u r s e (j o h n)$ ?
  Answer: NO.

Query 4a asks whether John teaches an intermediary course. At a superficial reading of the query, it might seem that the answer should be NO. The answer NO is supported by the fact that none of the courses taught by John is known to be an intermediary course, i.e. neither $M e d C o u r s e (c s 2 2 1)$ nor $M e d C o u r s e (c s 3 2 4)$ is a logical consequence of $\Sigma$. Nevertheless, the correct answer is YES, and in order to get it, one must reason by case analysis: As we have already remarked, the knowledge base does not provide the information as to whether Susan is a graduate or an undergraduate; however, since she is a student, according to $T$, she must either be one or the other. This fact ensures that in every model for $\Sigma$ either $G r a d (s u s a n)$ or $U n d e r g r a d (s u s a n)$ holds. Consider now the set of models for $\Sigma$ in which $G r a d (s u s a n)$ holds. In each of these models, the course CS324 is taken by both a graduate (Susan) and an undergraduate (Peter), thus it is an intermediary course. Similarly, consider
the set of the remaining models for $\Sigma$, i.e. the ones in which $\text{Undergrad}(\text{susan})$ holds. It is easy to see that in every model for this set the course CS221, this time, is taken by both a graduate (Mary) and an undergraduate (Susan), and therefore it is an intermediary course.

In conclusion, in every model for $\Sigma$ either CS324 or CS221 is an intermediary course. It follows that in every model for $\Sigma$ John teaches an intermediary course, proving that the correct answer to Query 4a is **YES**.

On the other hand, Query 4b asks whether John is known to teach a course that is known to be an intermediary course. The courses known to be taught by John are CS221 and CS324 and the only known intermediate course is EE282, therefore none of them is within the conditions required by the query.

Query 4a shows how, in some cases, the first order semantics might not agree with the intuitive reading of a query. In fact, most people tend to read Query 4a as requiring the reasoning pattern that is actually associated with the semantics of Query 4b. In other words, they tend to rule out the case analysis from the computation. One good reason to do so is that case analysis generally makes reasoning harder. In fact, as proved in Chapter 4, the problem of answering queries with existential quantification under the first order semantics, is in general coNP-hard. Whereas, as shown in Chapter 6, queries involving existential quantification only of the form $\exists K.P.KC$ can be answered in polynomial time w.r.t. the size of the knowledge base. However, there are also cases in which the intuition agrees with the first order interpretation. For this reason, in our opinion, it is important to have the operator $K$, which gives the possibility to choose between the two alternative readings of the query.

Regarding the interaction between the epistemic operator and the quantifiers, notice that we have always considered queries of the form $\exists K.P.KC$ and $\forall K.P.KC$, i.e. queries in which the $K$ operator is placed in front of both the concept and the role. Such queries usually have an easy intuitive interpretation and therefore are the most interesting. Nevertheless, it might be worthwhile to consider even other possible variations of them, for example queries like $\exists K.P.C$ or $\forall P.KC$. Such queries are perfectly legal in $\mathcal{EQL}$, however, in some cases, they may lack an intuitive meaning. The reason is that they amalgamate $\mathcal{DDC}$-concepts with epistemic ones, resulting in something to which it is usually hard to give an intuitive meaning.

In other cases, though, they can play a useful role. As an example consider the following queries:

- **Query 4c**: $\Sigma \models \exists K \text{TEACHES}\_\text{MedCourse}(\text{john})$?  
  **Answer**: **YES**.

- **Query 4d**: $\Sigma \models \exists \text{TEACHES} \_K \text{MedCourse}(\text{john})$?  
  **Answer**: **UNKNOWN**.

Query 4c gets the same answer (**YES**) as Query 4a. In fact, since the knowledge base “knows” that $\text{TEACHES}(\text{john, CS221})$ and $\text{TEACHES}(\text{john, CS324})$
hold, the addition of $K$ in front of $\text{TEACHES}$ does not change the answer to the query. Query 4d, instead, is answered $\text{UNKNOWN}$ because the only known intermediate course is EE282 and we can neither prove nor exclude that John teaches it.

The fact that Query 4c gets the answer YES and Query 4d the answer UNKNOWN may help us understand the answers to Query 4a and 4b. In particular, it clarifies which is the actual reason that makes Query 4a and 4b different: It tells us that the incompleteness of the knowledge base is related to the concept $\text{MedCourse}$ and not to the role $\text{TEACHES}$. In fact, $\text{TEACHES}(\text{john}, \text{cs324})$ and $\text{TEACHES}(\text{john}, \text{cs221})$ are both true in $\Sigma$, while $\text{MedCourse}(\text{cs324})$ and $\text{MedCourse}(\text{cs221})$ are not—only their disjunction is true.

Let us see now the case of an existential quantifier:

- Query 5a: $\Sigma \vDash \forall \text{TEACHES.}\exists \text{ENROLLED}(\text{john})$? Answer: $\text{UNKNOWN}$
- Query 5b: $\Sigma \vDash \exists K \text{TEACHES.}\exists \text{ENROLLED}(\text{john})$? Answer: YES

Query 5a gets the answer $\text{UNKNOWN}$ because there is a model for $\Sigma$ where John teaches a course, say ee148, but there are no students enrolled in ee148, i.e. ee148 is not an instance of the concept $\exists \text{ENROLLED}$. On the other hand, the correct reading of Query 5b is as follows: Is it true that for every course that John is known to teach, there is at least one student enrolled in it? It is easy to see the answer to the query is YES.

The above example shows that the use of $K$ allows one to pose queries to a knowledge base $\Sigma$ asking the system to assume complete knowledge on a certain individual $a$ and a certain role $P$ in $\Sigma$ ($\text{john}$ and $\text{TEACHES}$ in the example). In particular, assuming complete knowledge on $a$ and $P$ here means assuming that for every pair $(a, b)$ such that $\Sigma \not\vDash P(a, b)$, the assertion $P(a, b)$ is false in $\Sigma$. It is clear that this kind of reasoning is a form of closed world reasoning.

We argue that the use of epistemic operators is a natural way to achieve such a flexible way of interacting with the knowledge base. Indeed, the careful introduction of the epistemic operator into the query induces the system to answer queries under the assumption that part of the knowledge base is complete, in contrast to assigning a closed world semantics to the knowledge base itself.

Consider the following query to the knowledge base $\Sigma$:

- Query 4e: $\Sigma \vDash \exists K \text{TEACHES.} K(\text{Course} \cap \exists \text{ENROLLED, Grad} \cap \exists \text{ENROLLED, } (\text{Student} \cap \neg K \text{Grad}))(\text{john})$? Answer: YES.

Query 4e is syntactically equal to Query 4b, except that the concept
MedCourse is replaced by the $\mathcal{EQC}$-concept
\[
\text{Course} \sqcap \exists \text{ENROLLED. Grad} \sqcap \exists \text{ENROLLED.} (\text{Student} \sqcap \neg \text{KGrad}).
\]

(8.1)

Concept (8.1) differs from the definition of MedCourse in the fact that Undergrad is replaced by (Student $\sqcap \neg \text{KGrad}$). Concept (8.1) should be interpreted as the concept describing the courses that are intermediary under the assumption that every student is an undergraduate, unless the contrary is known. In fact, a course belongs to such a concept if both a graduate and a student not known to be a graduate are enrolled in it.

It is easy to see that the course CS221 is an instance of Concept (8.1), and therefore the answer to Query 4e is YES.

Notice that asking queries like Query 4e is completely different from giving some kind of closed world semantics to the knowledge base. In fact, in our framework the knowledge base is perfectly monotonic, whereas using the epistemic operator the queries can be formulated in such a way that the reasoning which is required to compute the answers is nonmonotonic.

### 8.2.2 Weakening the Inclusions

In case the complexity of reasoning would turn out to be to high, it is possible to consider the weakening of some or all inclusions. In this case, the choice of which inclusion should be weakened, strongly depend on the way the inclusions are used in the reasoning process. We now suppose to weaken all of those of the knowledge base $\Sigma$ of our example. The weakening $\Sigma^-$ will be $\langle \emptyset, A, R, I_C \rangle$, where $R$ is shown in Figure 8.3. Recall that all definitions of the form $C \sqsubseteq D$ are equivalent to $C \leq D$ and $D \leq C$.

It can be verified that all queries asked to $\Sigma$ in Section 4 have the same answer in $\Sigma^-$, except for queries 4a and 4c, reported here for the sake of clarity.

- Query 4a: $\Sigma \models \exists \text{TEACHES.MedCourse} (\text{john})$ ? Answer: YES.

- Query 4c: $\Sigma \models \exists \text{KTEACHES.MedCourse} (\text{john})$ ? Answer: YES.

These queries receive the answer YES in $\Sigma$ because of a case analysis on Susan. Recall that, according to $T$, the TBox of $\Sigma$, the two concepts Grad and Undergrad partition the concept Student. Being a student, Susan can be either a graduate or an undergraduate. In the first case, the course CS221 is an intermediary course, while in the second case CS324 is an intermediary course. Hence, in both cases John teaches an intermediary course.

On the contrary, it is easy to see that this does not happen in $\Sigma^-$, as shown by the following queries.

- Query 4f: $\Sigma^- \models \exists \text{TEACHES.MedCourse} (\text{john})$ ? Answer: UNKNOWN.
8.2 Example

\[
\begin{align*}
\text{KAdvCourse} & \preceq (\text{Course} \cap \forall \text{ENROLLED.Grad}), \\
\text{K(Course} \cap \forall \text{ENROLLED.Grad}) & \preceq \text{AdvCourse}, \\
\text{KIntCourse} & \preceq (\text{Course} \cap \forall \text{ENROLLED.Undergrad}), \\
\text{K(Course} \cap \forall \text{ENROLLED.Undergrad}) & \preceq \text{IntCourse}, \\
\text{KMedCourse} & \preceq (\text{Course} \cap \exists \text{ENROLLED.Grad} \cap \exists \text{ENROLLED.Undergrad}), \\
\text{K(Course} \cap \exists \text{ENROLLED.Grad} \cap \exists \text{ENROLLED.Undergrad}) & \preceq \text{MedCourse}, \\
\text{K\exists \text{TEACHES.Course}} & \preceq \text{Grad} \cup \text{Professor}, \\
\text{KGrad} & \preceq (\text{Student} \cap \exists \text{DEGREE.Bachelor}), \\
\text{K(Student} \cap \exists \text{DEGREE.Bachelor}) & \preceq \text{Grad}, \\
\text{KUndergrad} & \preceq (\text{Student} \cap \lnot \text{Grad}), \\
\text{K(Student} \cap \lnot \text{Grad}) & \preceq \text{Undergrad}
\end{align*}
\]

Figure 8.3: Weakening of the knowledge base example

- Query 4g: $\Sigma^\sim \models \exists \text{KTEACHES.MedCourse(john)}$? Answer: UNKNOWN.

This is because in $\Sigma^\sim$ the two concepts Grad and Undergrad do not partition the concept Student. What we just know is that individuals known to be undergraduates are inferred to be students and nongraduates, and vice versa, that individuals known to be students and nongraduates are inferred to be undergraduates. Since Susan is in neither of the two conditions, we cannot infer anything about her. In fact, there are now epistemic models for $\Sigma^\sim$ where Susan is neither a graduate nor an undergraduate. Therefore, the two queries 4f and 4g receive the answer UNKNOWN.

One can also verify that contrapositives are not applicable in $\Sigma^\sim$. Compare the answer to $\lnot \exists \text{DEGREE.Bachelor(peter)}$ in the two knowledge bases:

- Query 6a: $\Sigma \models \lnot \exists \text{DEGREE.Bachelor(peter)}$? Answer: YES.
- Query 6b: $\Sigma^\sim \models \lnot \exists \text{DEGREE.Bachelor(peter)}$? Answer: UNKNOWN.

In fact, in $\Sigma$ Peter is known to be an undergraduate, hence a student who is a nongraduate. Since graduates are defined as students with a bachelor’s degree, we can infer that Peter has none by using the contrapositive of the inclusion $(\text{Student} \cap \exists \text{DEGREE.Bachelor}) \preceq \text{Grad}$. Instead, in $\Sigma^\sim$ we only can infer that Peter is a student and a nongraduate. This does not activate the contrapositive of the trigger rule $\text{K(Student} \cap \exists \text{DEGREE.Bachelor}) \preceq \text{Grad}$.
Chapter 9

Discussion and Conclusions

In this chapter we would like to give some general comments about the work done, rather than commenting about specific results, which has been already done at the end of the various chapters.

The chapter is split in two parts, the first one is devoted to the comments to the results and the second one regards the open problems.

The contents of this chapter are original observations of the author, except for the discussion pursued in Section 9.1.3, which is mostly taken from [BDS93a].

9.1 Comments to the Results

In this section, we discuss three topics that underlie the whole thesis, namely the study of the computational complexity, the use of expressive representational languages, and the complete reasoning procedures. Specifically, in Section 9.1.1 we discuss the importance of making a complexity analysis of the reasoning tasks of an AI system. In Section 9.1.2 we argue for the necessity of expressive representation mechanisms for general purpose KR systems. In Section 9.1.3, we defend the usefulness of the complete (but computationally intractable) reasoning procedures, such as those used throughout the thesis, from both theoretical and practical points of view.

9.1.1 Complexity Analysis

The importance of the complexity analyses has always been a debated point in the AI community. To this respect, there is now a large consensus on the importance of mapping out the borders of the tractability, see e.g. [LB87, Byd91]. In fact, in almost all areas of AI, there have been a big effort in identifying which problems are polynomial and which, instead, are NP-hard or coNP-hard.
On the other hand, it is more controversial whether it does really matter to completely classify a problem that has already been discovered to be NP-hard (or coNP-hard). However, as argued by Gottlob in [Got93] for the complexity of logic programming and nonmonotonic reasoning, we believe that determining the complexity of a problem does not merely consist in attaching to this problem a quantitative result, namely the degree of its intractability in the worst case. Conversely, we think that the worst case complexity analysis gives us usually far more, namely, a very good qualitative understanding of the problem. This qualitative knowledge can be fruitfully applied for designing algorithms or for understanding the relationship between the various reasoning services and the various languages.

For instance, recognizing that a given problem is in $\Pi_p^2$, suggests us to design an algorithm based on two levels of non-deterministic choice. In addition, techniques used to solve other $\Pi_p^2$ problems, also in different areas can be profitable exploited for the solution of our problem.

In addition, the comparison of the complexity of different services highlights the possible sources of complexity of reasoning. For example, the fact that Instance Checking in $\mathcal{ALC}$ is PSPACE-complete, whereas Subsumption is NP-complete highlights a new source of complexity in concept-based reasoning that have a big practical impact in the design of Instance Checking algorithms in any language that is a superlanguage of $\mathcal{ALC}$.

### 9.1.2 Expressive Representation Mechanisms

Several researchers have argued for the use of restricted representation formalisms (e.g. [Pat84]), whereas several others claim the necessity of having an expressive representation language (see e.g. [DP91, BN92]). Our opinion is that the user should be given all the expressive power she/he needs, provided that she/he is aware of the computational demand of her/his application. Therefore, we shouldn’t give up the expressiveness of the overall system, on condition that the efficiency of simple cases is not penalized. Therefore, we think that we should work for expressive systems that are able to perform comparably with small systems in the cases managed by the small system (as done in [BHN+92]). In this way the user is able to choose whether to have a fast simple-structured application or a slow expressive system. The system $\mathcal{AL-K}$ goes in this direction. For example, it provides both rules and inclusions which are similar mechanisms with different computational expense. The user, based on her/his knowledge on the domain, may choose whether to employ one mechanism or the other for each specific piece of knowledge to be modeled.
9.1.3 Complete (Intractable) Reasoning Procedures

Throughout this thesis, we only focused on complete reasoning procedures. Several researchers have argued for the use of only polynomial (even sub-quadratic) procedures, claiming that the speed of reasoning is the primary quality of a practical KR system.

However, we identify several practical reasons to have a complete procedure. First of all, a complete procedure can perform well in practical cases despite its worst-case intractability. Moreover, it may work in polynomial time for small sublanguages where reasoning is tractable, while being still complete when solving more complex tasks.

Secondly, a complete procedure offers a benchmark for comparing incomplete procedures, not only in terms of performance, but also in terms of missed inferences. Let us illustrate this point in detail, by providing a blatant paradox: consider the mostly incomplete constant-time procedure, answering always “No” to any check. Obviously this useless procedure outperforms any other one, if missed inferences are not taken into account. This paradox shows that incomplete procedures can be meaningfully compared only if missed inferences are considered. But to recognize missed inferences over large examples, one needs exactly a complete procedure—even if not an efficient one—like ours. We believe that a fair detection of missed inferences would be of great help even when the satisfaction of end users is the primary criterion for judging incomplete procedures.

Thirdly, a complete procedure can be used for “anytime classification”, as proposed in [Mac92]. The idea is to use a fast, but incomplete algorithm as a first step, and then do more reasoning in background. In the cited paper, resolution-based theorem provers are proposed for performing this background reasoning. We argue that any specialized complete procedure will perform better than a general theorem prover. For instance, theorem provers are usually not specifically designed to deal with filtration techniques.

Moreover, our calculus deals with rules. Our result in Section 7.3 about inclusions implies that rules, being a limited form of inclusion, can be applied also to unknown individuals (our variables in a constraint system) without endangering decidability (at least in $\mathcal{ALCN}^\mathcal{R}$). This result is to be compared with the negative result in [BH92], where it is shown that Subsumption becomes undecidable if rules are applied to unknown individuals in $\text{CLASSIC}$.

Finally, the calculus provides a new way of building incomplete procedures, by modifying some of the completion rules. Since the rules build up a model, modifications to them have a semantical counterpart which gives a precise account of the incomplete procedures obtained. E.g., one could limit the size of the canonical model by a polynomial in the size of the knowledge base. Semantically, this would mean to consider only “small” models, which
is reasonable when the intended models for the knowledge base are not much bigger than the size of the knowledge base itself. We believe that this way of designing incomplete procedures “from above”, i.e. starting with the complete set of inferences and weakening it, is dual to the way incomplete procedures have been realized so far “from below”, i.e. starting with already incomplete inferences and adding inference power by need.

In addition, as argued in [BN92], there are several arguments against the use of incomplete algorithms. Ideally, there should be no gap between the declarative semantics, i.e. what the system description promises a user, and the operational semantics, i.e. what the system actually does. A possible way of invalidating this argument is to develop a nonstandard semantics for algorithms which are incomplete with respect to standard semantics. However, this alternative is acceptable only if this semantics not just simulates the algorithm on the semantic level, and therefore is not better than a description of the algorithm. Even if this constraint is satisfied, such nonstandard semantics is often more difficult to understand for a user; thus some effort is required to make it simple and understandable.

In [BN92] it is also argue that when adding nonmonotonic features (like defaults or epistemic operators) to a concept-based system, the distinction between completeness and soundness no longer makes sense: Incompleteness of the basic reasoner causes the nonmonotonic reasoner to be unsound as well. Since, caused by user demand, there has recently been a lot of activity in this direction, this argument should not be neglected.

9.2 Open Problems

Speaking about open problems a distinction is necessary: There are general open problems, i.e. problems of the whole area of concept-based reasoning, and specific one, more closely related to the thesis.

Regarding the former ones, they can be recognized in the following issues.

1. Non-monotonic reasoning (e.g. [BH93])

2. Embedding in programming environments (e.g. [YNM91])

3. Interface with databases (e.g. [DBMP90])

4. Integration with other reasoning paradigms (e.g. [WL92])

5. Implementation issues and scalability (e.g. [McG92])

6. Revision and update (e.g. [Neb90a])

7. Standardization [PS93]
A treatment of the above items is out of the scope of this thesis, therefore we refer to the cited paper (and many others) for a discussion about them. Most of them have been broadly discussed also in [MMMR92, BLNP94]. Conversely, in the following sections, we discuss the open problems closely related to the thesis.

### 9.2.1 Missing Complexity Results

Various results are still missing from our analysis. The complexity of Instance Checking in \( \mathcal{AL} \) w.r.t. the data complexity has been conjectured to be \( \Pi^2_2 \)-complete, but not formally proved. In addition, the complexity of the reasoning tasks in the system \( \mathcal{AL-K} \) has not been identified yet. It is not even yet known whether it is at least decidable.

Moreover, the set of languages considered in this thesis is far from being exhaustive. Many other languages can be considered and classified in the complexity classes hierarchy. However, producing a complete picture is an enormous task and it might be useless. In our opinion, the important task is to identify the constructors (or combinations of constructors) that are responsible for a source of complexity.

Another related task still to be accomplished is the identification of languages that are maximally polynomial w.r.t. the Instance Checking, as done in [DLNN91b, DLNN94] for Subsumption.

### 9.2.2 Implementation

The algorithms proposed in this thesis are generally very naive. In fact, they have been designed mostly for theoretical purposes (proof of upper bounds) than for actual implementation. In order to make them effective, several optimizations are necessary, such as those proposed in [BHN+92].

Furthermore, all the technical details have been neglected. In fact, several issues should be addressed to build an actual reasoning procedure, e.g. suitable data structures, parsing, and hashing mechanisms. In addition, all the software engineering issues [Bra92, McG92] have been ignored.

Finally, it is necessary that reasoning algorithms are able to perform well in the subcases where reasoning is tractable. One naive way to achieve such capability is to design a family algorithms able to manage the various sublanguages of a given language, combined with a syntax analyzer which decides which algorithm must be employed in each specific case.

### 9.2.3 Restrictions

We have seen that only limited languages show tractable reasoning tasks, whereas actual users usually demand for more expressive mechanisms. One
way to “solve” this problem is to abandon the tractability, requiring the much less restrictive condition of the decidability.

Another possible way to face this situation might be that of giving the user only a limited form of those constructors that lead to intractability. For example, it is shown in [DLNN94] that the inverse role combined with other constructors is intractable; however, the intractability of inverse role is due to the possibility of nesting arbitrary number of quantification using the same role in direct and inverse form. It might be the case that some syntactic constraint on the use the inverse role rules out the intractable cases without ruling out the cases useful for practical purposes.

As another example, we have seen that the use of terminological cycles leads to high intractability (EXPTIME or higher). However, as it is shown in [BJNS94, BDNS94], a limited form of cycles can be introduced without endangering the tractability of reasoning. In particular, in [BDNS94] we argue for the use of cycles only in a part of the TBox, called schema, expressed only in terms of concept specifications with no definition.

### 9.2.4 Extensions

Several extension of our system might be considered. Various constructors of practical interest have not been mentioned in this thesis, for example the use of functional roles (*features*) and agreements of paths of features. Moreover, the addition of concrete domains, such as integers and strings [BH91a], is undoubtly necessary in practical systems, and it is still open.
Bibliography


[BBH+91] Franz Baader, Hans-Jürgen Bürkert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernard Nebel, Werner Nutt,


[BH91b] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In Proc. of the Workshop on Processing Declarative Knowledge,


[DLN+93] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaefer. Queries, rules and definitions as epis-


[Hol93] Bernhard Hollunder. How to reduce reasoning to satisfiability checking of concepts in the terminological system KRIS. Submitted for publication, 1993.


[NS89] Bernhard Nebel and Gert Smolka. Representation and reasoning with attributive descriptions. Technical Report 81, IWBS, IBM Deutschland, P.O. Box 80 08 80 D-7000 Stuttgart 80, Germany, 1989.


[Sch93] Andrea Scherf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal*


