WARSAW UNIVERSITY
OF TECHNOLOGY

**Faculty of Electronics
and Information Technology**

Ph.D. THESIS

Przemysław Więch, M.Sc.

Distributed Default Reasoning in the Semantic Web

Supervisor
Professor Henryk Rybiński, Ph.D., D.Sc.

Warsaw, 2011

Streszczenie

Rozprawa przedstawia nowatorskie podejście do rozproszonego wnioskowania w systemie wieloagentowym, w którym bazy wiedzy agentów wyrażone są przy pomocy logik opisowych z domniemaniami. W Semantycznym Internecie (ang. Semantic Web) wiedza jest rozproszona pomiędzy wiele niezależnych jednostek. Agent w takim systemie wieloagentowym może potrzebować zebrać informacje z różnych zdalnych źródeł, aby odpowiedzieć na zadane pytanie. Logika domniemań może zostać włączona do logiki opisowej w celu właściwego traktowania niepełnej wiedzy i umożliwienia wycofywania się z wniosków, które stały się sprzeczne z aktualnym stanem bazy wiedzy. Praca przedstawia formalizm transformacji domniemań, który może zostać wykorzystany do wyprowadzania odpowiedzi na zapytania w postaci domniemań. Tak powstałe domniemania mogą być traktowane jako pośrednie rezultaty w procesie wnioskowania. W pracy pokazano, że przekazywanie komunikatów zawierających domniemania powstałe w wyniku transformacji niesie więcej informacji niż przekazywanie faktów, a także pozwala na uniknięcie zbyt szybkiego wyciągania wniosków na podstawie domniemań. Praca przedstawia również model systemu wieloagentowego wykorzystującego zaproponowane algorytmy. Zaprezentowana jest również implementacja prototypowego systemu, który łączy w sobie wszystkie zaproponowane algorytmy.

Słowa kluczowe: *Semantic Web, system wieloagentowy, rozproszone wnioskowanie, logika domniemań, logika opisowa.*

Abstract

The dissertation presents a novel approach to distributed reasoning in a multi-agent system, in which the agents' knowledge bases are expressed in the language of description logic with defaults. In the Semantic Web environment, knowledge is distributed among many independent entities. An agent in such a multi-agent system may need to gather information from several remote sources in order to provide an answer to a query. Default logic can be embedded into description logic in order to handle incomplete knowledge, where it should be possible to retract inferences, which are proved to be incorrect. The thesis proposes a formalism of default transformations, which can be used to derive answers to queries in the form of defaults. Such new defaults can then be treated as intermediate results in the reasoning process. It is shown that passing messages containing transformed defaults is more informative than strict statements and avoids reaching conclusions too early. Furthermore, the model of a multi-agent system, which utilizes the proposed algorithms is proposed. Finally, the work describes the implementation and evaluation of a prototype system, which combines all of the introduced algorithms.

Keywords: *Semantic Web, multi-agent system, distributed reasoning, default logic, description logic.*

Acknowledgements

I would like to thank my supervisor, Henryk Rybiński, for his support and guidance over the years. Without his continuous belief in me, this work would not have been finished.

I am grateful to my family for supporting me in every moment. I would especially like to thank my wife, Magda, for her endless understanding and patience.

I would like to express my appreciation to my fellow Ph.D. students and employees of the Division of Information Systems for the many insightful discussions.

Przemysław Więch

Warsaw, 2011

The research was supported by the Polish National Budget Funds 2009–2011 for science under the grant N N516 378136.

Contents

Chapter 1

Introduction

The current World Wide Web can only be viewed as a syntactic structure of web pages in which search engines analyse only sequences of words detached from their meaning. The idea for transforming the existing Web, named Semantic Web [16], envisions the knowledge contained in the Web to be expressed in a formalised way to make it possible to be processed on the level of semantics. The semantic information is intended to be integrated into web pages and other web resources by means of annotating data and supplying concrete meanings to terms. This process is expected to lead to a network of linked data, which is seen as a concept similar to joining tables in a relational database. In this case, however, it should be possible to link data from different remote sources.

The research concerning the elements of the Semantic Web is very active. There is work on knowledge representation in designing expressive ontology languages to describe information. Tools are being built for creating and maintaining ontologies and adding semantics to web pages. Moreover, Semantic Web Services are considered as a way to remotely access semantic information. The Internet is a decentralized environment and it is impossible to store information in one location. This is why distributed algorithms are needed to make use of all accessible information. The Semantic Web vision is also connected to the notion of agents and multi-agent systems working in a peer-to-peer network. Agents are envisioned to perform tasks on behalf of their users by collecting information from remote sources and by communicating with other agents. In this case, the role of ontologies is to provide the agents with a common vocabulary for knowledge exchange.

The Semantic Web requires a knowledge representation (KR) formalism to express the knowledge in the Web. KR has a long history in the domain of artificial intelligence, where it is intended to represent world objects, their properties and relationships between objects. Techniques such as frames, logic programs and semantic networks form the foundation for today's knowledge representation methods.

Ontologies fulfill the necessity to formally convey information and act as a conceptualisation of a specialisation [28]. Their purpose is to systematically express knowledge, enable different systems to communicate using a vocabulary with common semantics and to simplify the reuse of gathered knowledge. Ontologies define concepts, which are used to describe world objects, and relationships between concepts or objects. It is required that ontology languages have a well defined syntax and semantics, an adequate degree of expressivity and support for reasoning mechanisms.

The logical formalism behind ontologies is provided by description logics (DLs) [10], which are a family of knowledge representation languages. DLs are constrained subsets of first-order logic that provide efficient reasoning algorithms. A description logic knowledge base is constructed in terms of concepts, roles and individuals. The primary inference tasks in description logics are subsumption checking and instance checking. The former tests whether all instances of one concept are instances of another concept, whereas the latter checks whether a concrete individual is an instance of a given concept. Today, the most often used ontology language is OWL (Web Ontology Language) [52] which is based on an expressive description logic.

In the domain of databases, the closed-world assumption (CWA) is commonly used in representing data. It says that if a fact cannot be proved to be true, then it is presumed to be false. Thus, negative information does not have to be explicitly included. In contrast, the knowledge accessible on the Web by its nature cannot be treated as a closed and complete knowledge base. Because of this, the Semantic Web knowledge representation formalisms are based on the open-world assumption (OWA), which states that if something is not known, it cannot be treated as being false. This way, the knowledge base behaves monotonically – newly introduced facts do not falsify previous conclusions.

The closed-world assumption, can also be beneficial in some cases in such an open environment as the Semantic Web. Making assumptions is a commonsense way of reasoning about incomplete information. For example, it is justified to assume that most birds fly, although there are some known exceptions. When presented with an unknown bird, which does not belong to the known exceptions, it is a rational conclusion that it does fly. Expressing such assumptions is possible when dealing with nonmonotonic logic formalisms [4].

One of the most studied examples of nonmonotonic logic is default logic proposed by Raymond Reiter [59]. This formalism allows to express defaults as rules, which are defeasible. By providing an exception, the conclusion made by a default rule can be invalidated. Introducing the notion of defaults into the Semantic Web knowledge representation formalisms can improve the expressivity of online knowledge bases and allow dealing with incomplete knowledge.

The Semantic Web is an inherently distributed environment. A peer in the Web can represent an online service (Web Service), software agent or information source. Peers have local knowledge bases and reasoning engines, however it is desirable for an agent to query the peers' combined knowledge. Apart from dealing with knowledge bases of different entities, distributed reasoning is used to increase efficiency by splitting the knowledge base into partitions with local reasoning engines. Moreover, the Semantic Web is already made of separate but interconnected ontologies. The aim of distributed reasoning is to combine the knowledge that is located within the peers in the Web and make conclusions based on the joined knowledge. An important property of the Semantic Web is that distributed reasoning has to take into account that the knowledge is already partitioned and one has no influence on the way the partitions are formed.

In the environment of a multi-agent system, the agents communicate using a formalised protocol. Among many protocols, which can be used in a multi-agent system, the simplest and most commonly used is the query-answer method, where an agent issues a query to another agent, which in turn returns an answer according to its knowledge base. This simple interaction is presented by the Foundation for Intelligent Physical Agents as the *FIPA Query Interaction Protocol*[1]. One of the primary reasoning tasks in description logics is the subsumption problem, where we are interested whether a concept C is a subconcept of D, which means that all instances of C also belong to D. This reasoning task can be translated into a query, which is sent between agents. Using description logic without defaults, the answer to such a query can only be positive or negative. However, when the agent, which is queried can reach a positive answer only by applying defaults, it makes some assumptions, which could later be proved to be incorrect. If this agent answers with a simple positive statement, the information about these assumption is lost. In this work, we propose a method that enables agents to exchange defaults in reply to subsumption queries. This method preserves the assumptions made during reasoning and reduces the loss of information when dealing with defaults.

In order to achieve this goal, the formalism of default transformations is introduced. We show a set of basic default transformations, which have the property that adding a transformed default to the original knowledge base results in obtaining the same conclusions as if the reasoning was made with the original default theory. By applying a series of transformations, we can obtain a default, which can be used as an answer to a subsumption query. We introduce the algorithm for finding transformed defaults, which match a sumbsumption query, if this query cannot be resolved with a simple positive or negative answer. This algorithm is based on several other algorithms. Firstly, a tableau reasoning procedure is used for the underlying description logic [10]. Secondly, we use the method of Risch and Schwind [61] for calculating the extensions of a default theory. Lastly, a method described by Kalyanpur [43] is used for determining maximal consistent subsets of description logic statements.

For the default reasoning formalism we have built a model of a multi-agent system for exchanging knowledge in the Semantic Web. An agent in this model consists of domain and environmental knowledge. Moreover, the agent can use global knowledge publicly available online. We describe a distributed reasoning algorithm, in which

[1]http://www.fipa.org/specs/fipa00027/

the agents exchange queries and answers to reach a specified goal. The algorithm introduces a tableau rule that triggers sending queries to remote agents. Opposite to the existing proposals for using default reasoning, we presume that not only the logical values or induced facts can be subject of exchange between agents, but also the "deduced" default rules, so that the agent can properly continue reasoning. To this end our method utilizes the previously introduced default transformations to provide the possibility of exchanging defaults between agents.

The thesis of this dissertation is formulated as follows:

1. Default logic embedded in description logic makes it easier to deal with incomplete knowledge.
2. Description logic combined with default logic is an appropriate method for knowledge representation in the Semantic Web environment.
3. This method appropriately treats the incompleteness of descriptions and is suitable for distributed reasoning in a multi-agent system.
4. Distributing the reasoning processes leads to the improvement of efficiency.

The dissertation is composed of seven chapters.

Chapter 2 contains an overview of the state of the art in the domain of the Semantic Web, multi-agent systems and default logic. The concept of ontologies for knowledge representation and exchange is introduced together with ontology languages used to express information. The problem of distributed reasoning in the Web is presented in the context of multi-agent systems. Finally, the nonmonotonic default logic is described as a way to express assumptions about incomplete knowledge and introducing the closed-world assumption to open-world knowledge bases.

Chapter 3 introduces the definitions and algorithms connected with description logics, default logic and the combination of these two formalisms.

Chapter 4 focuses on the logical apparatus of description logic with defaults (DL$\mathfrak{D}$) with the emphasis on preparing the local knowledge base for answering queries in a distributed system. The notion of default transformations is introduced and a theorem regarding the application of default transformations is presented. Finally, the algorithm for reasoning in a DL$\mathfrak{D}$ knowledge base is described that makes use of default transformations.

In Chapter 5 the model of a multi-agent system for distributed reasoning is presented. The model makes use of the inference method described in the previous chapter in which the components of the agent knowledge base and the algorithm for distributed reasoning through exchanging knowledge between agents are presented.

Chapter 6 provides a description of a prototype implementation of the multi-agent system proposed in the previous chapters. Key components of the system architecture are introduced and implementations of specific problems are discussed. Finally, the proposed algorithms are evaluated in terms of correctness and scalability.

Chapter 7 concludes the thesis outlining the main claims and results. Moreover, perspectives for future research interests are presented.

Chapter 2

Related work

2.1 Semantic Web

The World Wide Web currently consists mostly of linked hypertext documents, which are readable for people but are not prepared for computers to acquire information from them because they are written in the natural language. Instead, computer programs can only treat texts as streams of words, which are not connected with their meaning. Even information which comes from databases is often stripped from its original structural information. Moreover, the HTML markup language, which is primarily used to create web pages, contains only presentational information without any semantic annotations. The Semantic Web [16, 6, 36] is a vision for transforming the World Wide Web into a semantically rich web of information. Such information could be collected and processed by computer programs not only as sentences in natural language, but as knowledge which is associated with its actual meaning. In consequence, computers could interpret the contents of the web in the Way the authors of the information had meant it.

The Semantic Web, just as the Internet and the World Wide Web, is decentralized and open for everyone to add content. Moreover, people publish content in various forms such as text, images, sound and video. The Web also spans across languages and cultures. This makes it very hard to capture the meaning of each piece of information made public on the Web by individuals around the world.

In order to be able to express the meaning of information stored in the Web, a language for knowledge representation is needed, which would allow computers to process it, link related information and infer new facts from the available knowledge. The current trend in the Semantic Web community is to utilize ontologies for expressing knowledge in the Web.

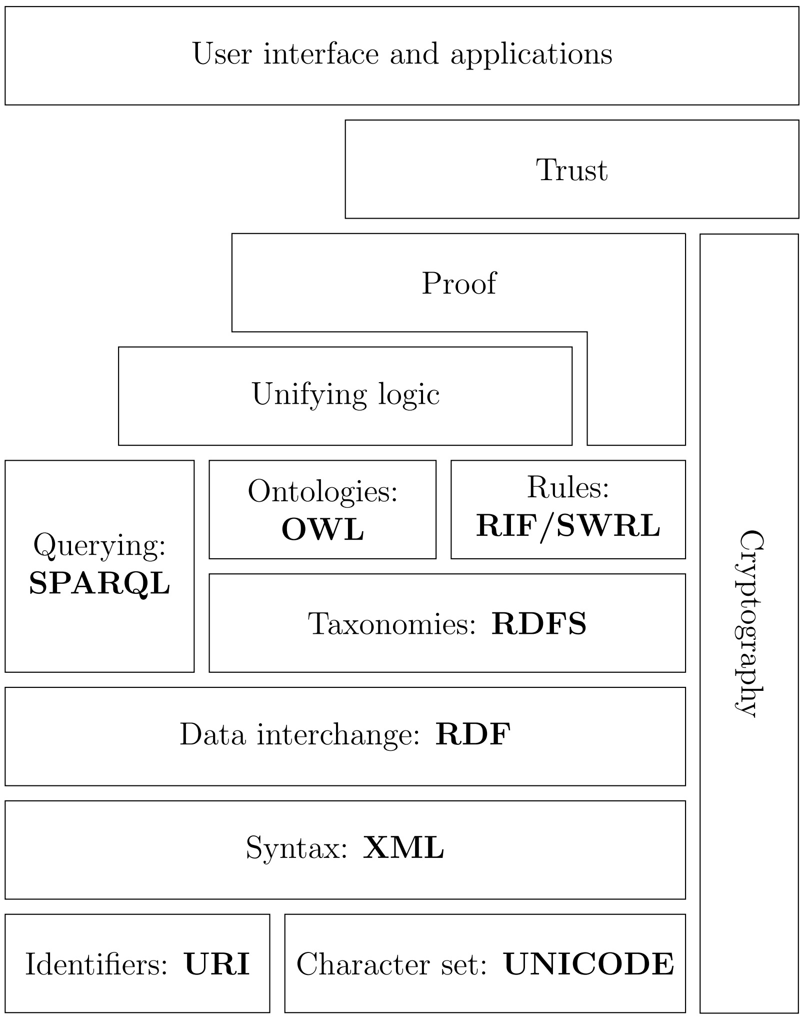

Figure 2.1: The Semantic Web stack

Figure 2.1 presents the Semantic Web Stack, which was first proposed by Tim Berners-Lee in one of his presentations [1]. The diagram shows the main layers of the Semantic Web vision. The Semantic Web utilizes the same underlying technologies as the current World Wide Web. Uniform Resource Identifiers (URIs) to identify web

[1]`http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`

resources, Unicode as the standard character encoding and XML as a universal data format are already the dominant standards. The Resource Description Framework (RDF) is a basic data model and together with RDF Schema is also becoming the standard as the base for knowledge representation on the Web. The next layer specifies three components, which represent the need for a query language for the knowledge in the Semantic Web, a rich ontology language for world descriptions and a rule language to complement the knowledge representation. The unifying logic layer's role is to merge the lower layers into a single logical view of the knowledge represented by ontologies and rules. The proof layer involves obtaining, presenting and validating proofs. The trust layer uses digital signatures and other information to verify information, which is exchanged on the Web.

Horrocks et al. [37] provide an alternative layering of the Semantic Web stack, where it should be split into "two towers", where the *rules* component should be placed directly on top of the XML layer beside the RDF, RDFS and OWL stack. This model is motivated by the fact that current rules languages for the Semantic Web are based on Datalog, which follows the closed world assumption, and as such is incompatible with RDF, which is based on first-order logic and the open world assumption. Rules language extensions such as *negation as failure* (NAF) can be then placed on top of the *rules* layer.

This work's contribution extends the ontology component by adding support for distributed reasoning with the addition of defaults to an ontology language based on description logic.

The ability to search for information is crucial for the Web. There are currently great amounts of documents which can only be indexed by keywords or the words contained in these documents. There is a significant need for searching for information based on its semantics. Moreover, once relevant documents are found, the required information has to be extracted from its content. It is a very complex task, requiring advanced natural language processing, to create software which would give precise answers to questions based on natural language texts. Moreover, the sought facts often cannot be obtained from a single source. This implies that an agent has to be run to collect information from different sources and combine it to form the answer to a query. Finally, once a software agent collects some information, it has to present it in a readable way to the user. It can also be required to be able to display an explanation of the information, for example in the form of information sources and reasoning steps taken to achieve the goal.

The works [16] and [6] provide real-world example scenarios, where a Semantic Web agent searches for a specialist to establish an appointment, checks the user's schedule, negotiates the price, looks up the specialists' ratings in an independent source and after presenting several choices, takes necessary actions to finalize the task. The authors of *The Semantic Web Primer* [6] claim that such scenarios are already feasible from the scientific point of view. Progress now has to be made in the integration, standardization, development and adoption of semantic tools, which are already being created.

2.1.1 Ontologies

The term *ontology* originally refers to the subfield of philosophy, which deals with the nature of existence. This branch of metaphysics tries to identify and describe the things that actually exist. The studies date back to Plato and Aristotle, where the well-known "Tree of Porphyry" (Figure 2.2) tries to present a categorisation of world entities and their distinguishing features. The word *ontology* has recently been adopted by computer science with a different meaning. The definition given by Studer et al. [73], which extends Gruber's definition [28], states that "an ontology is an explicit and formal specification of a conceptualization".

The role of ontologies in computer science is to formally describe a domain of discourse. Usually an ontology will contain a set of terms together with relationships between them. The terms typically denote classes of objects and relationships include the most often used *is-a* hierarchy and other domain-specific associations.

In the Semantic Web setting, ontologies provide a shared vocabulary for expressing statements in a domain. They are needed to alleviate the problem of differences in terminology.

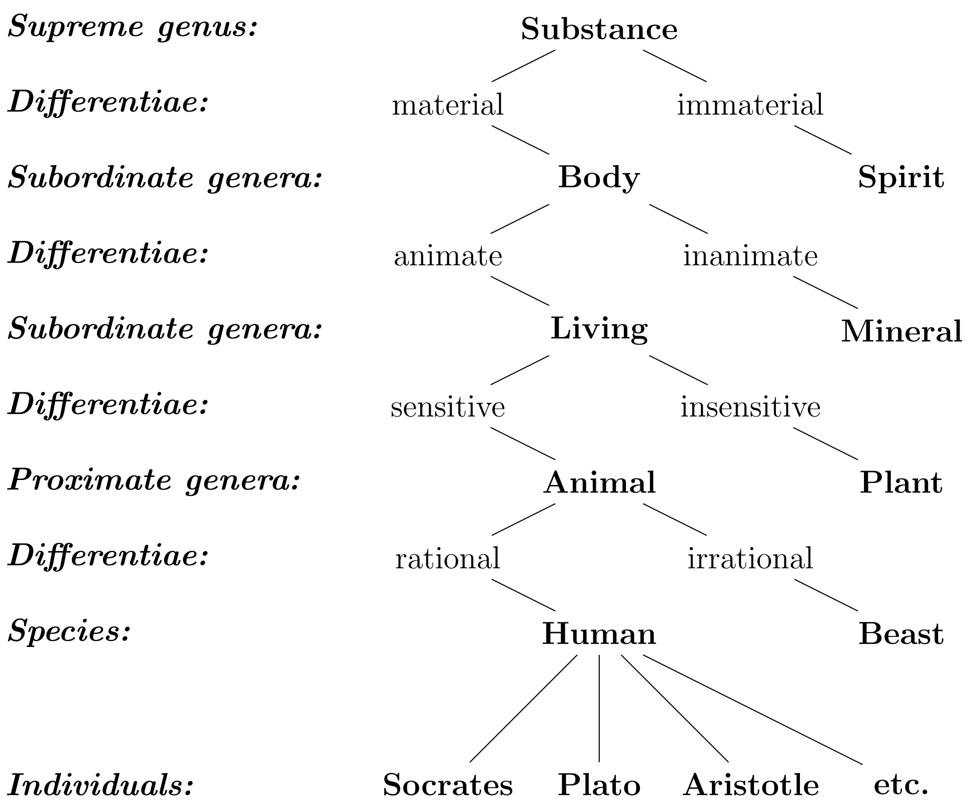

Figure 2.2: Tree of Porphyry as drawn by Peter of Spain (1239)

2.1.2 Ontology languages

Ontology languages allow us to describe domain models There are several key requirements of an ontology language [6]. A well-defined syntax is necessary for parsing and automated processing of information. Contemporary ontology languages are built upon the XML and RDF syntax. A formal semantics is needed to precisely define the meaning of knowledge. The semantics of ontologies is usually based on a logic formalism, such as first-order logic. Formal semantics is needed to make each statement unambiguous. An ontology language has to support efficient reasoning, which is enabled by the formal semantics of the language. Reasoning support is required among others in order to check the consistency of a knowledge base and to answer queries. For example, the OWL ontology language is based on description logic and thus existing DL reasoners can be employed. A sufficient expressive power is also required to describe as many aspects of a domain as possible. Every ontology language will have its limitations as to the knowledge it can express. However, expressivity is always increased at the cost of computational complexity of reasoning.

Ontology languages had been created before the Semantic Web concept was introduced. CycL [47] is a formal language developed for the Cyc project. It's aim is expressing a large knowledge base of commonsense knowledge. The language is based on first-order logic predicate calculus with features allowing to use nonmonotonic reasoning and the information in CycL is organized using frames. KIF (Knowledge Interchange Format) [26] is a language for the interchange of knowledge between heterogeneous software systems, whereas it was not originally intended to be an internal representation of knowledge. Ontolingua [28] is a representation language based on KIF with the addition of a frame ontology. An ontology created in Ontolingua is defined by relations, classes, functions, individuals and axioms.

The contemporary ontology languages are primarily using RDF (Resource Description Framework) [49] as the underlying data model. RDF is a very general data model for describing resources and relationships between them. The RDF data model is build with statements in the form of object-attribute-value triples. The primary syntax of RDF is based on XML [14]. However, other syntaxes could be used to express the RDF data model.

RDF schema [19] specializes RDF for describing properties and classes of RDF resources. It provides the means to express hierarchies of these classes and properties. RDF Schema explicitly defines the "is subclass of" relationship, which does not exist in RDF.

OWL (Web Ontology Language) [52] is semantically and syntactically a further specialization of RDF. It provides an expressive language for describing relationships between classes and individuals. OWL is a proposal of the W3C Web Ontology Working Group and is based on DAML+OIL [22]. DAML+OIL in turn comes from a joint initiative combining two earlier ontology languages, which are: DAML-ONT [51] and OIL [23].

Apart from the knowledge representation languages, the Semantic Web also needs a query language. Similarly to SQL for databases, SPARQL is a currently proposed by

W3C query language for RDF [58]. For a survey of different RDF query languages, see [30]. SPARQL is a graph-matching query language. A query forms a pattern, which is matched against the knowledge base. Since SPARQL deals with the general RDF data model, it is not very well suited for OWL, and especially OWL DL, which is strictly based on description logics. Thus, a subset of SPARQL, named SPARQL-DL, has been proposed [69]. SPARQL-DL can be implemented to make use of existing DL reasoners.

Despite the fact that ontology languages are based on logic, they are usually equivalent to a less expressive subset of first-order logic (FOL). This is caused by the necessity to choose between the expressive power and computational efficiency of knowledge representation languages. Brachman and Levesque [18] argue that the more expressive a representation language is, the more it is difficult to perform inferences. An important property of logic-based ontology languages is that a trace of the inference steps can be provided as an explanation of the result of reasoning.

2.1.3 Description Logic

Description logics (DLs) [10] are a family of knowledge representation formalisms. They are descendants of semantic networks [76] and the KL-ONE system [67]. Knowledge in DLs is represented by defining concepts from a selected domain, which comprise a terminology, and using these concepts for classifying objects and describing their properties. Description logics differ from earlier knowledge representation methods, such as semantic networks because they have a formalised semantics based on first-order logic. This property of DLs enables reasoning using logic-based methods.

A knowledge base expressed in description logic consists of two components. The *terminology* (Tbox) is a set of axioms defining and describing *concepts* and *roles*. Complex concepts and roles are defined using atomic ones. The *assertional component* (Abox) contains facts about objects expressed using concepts and roles. For example, using description logic the concepts $\mathsf{Penguin}$ and Bird can be created and it can be stated that all penguins are birds.

One of the simplest and much studied description logic is $\mathcal{ALC}$ [66], which provides the basic constructors for creating concept descriptions – conjunction, disjunction, complement, universal and existential quantifiers. More expressive description logic languages can be obtained by extending $\mathcal{ALC}$ and are named using letters that indicate the extensions included in the description logic. $\mathcal{ALC}$ with transitional roles is often denoted by $\mathcal{S}$. Other extensions include role hierarchy – $\mathcal{H}$ (e.g. $\mathsf{hasDaughter} \sqsubseteq \mathsf{hasChild}$), nominals (singleton classes) – $\mathcal{O}$ (e.g. $\{\mathsf{POLAND}\}$), inverse roles – $\mathcal{I}$ (e.g. hasChild is the inverse of isChildOf), quantified number restrictions – $\mathcal{Q}$ (e.g. $\geq_2$ $\mathsf{hasChild.Woman}$) and number restrictions – $\mathcal{N}$ (e.g. $\leq_1$ $\mathsf{hasChild}$) . The DL language $\mathcal{SHOIN}$ is the basis of the Web Ontology Language (OWL).

Description logics have a well understood base of reasoning algorithms. The most commonly used algorithms are based on semantic tableaux. These algorithms are implemented in reasoners such as Pellet [70] and FaCT++ [74]. A different approach to DL reasoning is taken in the KAON2 inference engine [40], where the knowledge base is first transformed into a disjunctive logic program.

The statements in description logic are interpreted as universal statements, which do not allow exceptions. This allows to automatically classify objects by deducing all concepts they belong to. However, this formalism does not allow to state assumptions. For instance, we would like to assume that typically birds fly, unless we have contrary information about a specific individual to conclude otherwise. Thus, a statement that Bird is a subconcept of Flies would lead to an incorrect conclusion that penguins (which are birds) can fly. Explicitly stating that penguins cannot fly would in this case be inconsistent with the information that birds can fly. To deal with such situations, a nonmonotonic reasoning method, such as default logic, should be taken into account.

2.1.4 OWL

The main focus of this work is on the Web Ontology Language. OWL is based on RDF Schema and uses its XML syntax. However, OWL adds several features that increase the expressive power of OWL over RDF Schema. Examples of such features include local scopes of properties (*Cows eat only plants*), disjointness of classes (*Male disjoint-with Female*), boolean combinations of classes (*A person is male of female*), cardinality restrictions (*A person has exactly two parents*) and special characteristics of properties (eg. inverse or transitive properties).

The W3C Web Ontology Working Group defined three sublanguages of OWL, each having a different proportion between the expressive power and computational complexity.

OWL Full uses all possible OWL primitives. It is entirely compatible with RDF, meaning that any valid RDF document is also a valid OWL Full document. The disadvantage of such an expressive language is that it is actually undecidable, which means that it is infeasible to build a complete reasoner for OWL Full.

OWL DL is a sublanguage of OWL Full which is restricted to correspond to a well-defined description logic. The advantage of OWL DL is that efficient reasoning is possible, as complete DL reasoners already exist. However, these restrictions cause the loss of full compatibility with RDF. Any valid OWL DL document is nonetheless a valid RDF document.

OWL Lite is in turn a sublanguage of OWL DL. The language has further restrictions on the constructs that can be used. The restricted expressivity makes the language easier to understand and easier to implement.

Since OWL is based on description logics, it uses the open world assumption and the non-unique-name assumption. The open world assumption is addressed by adding defaults to description logics and allowing to create rules which simulate the closed world assumption for selected concepts.

2.1.5 Open and closed world assumptions

The open and closed world assuptions show how to deal with incomplete knowledge. The problem is whether to permit statements, which have an undefined truth value, or to always presume that what is not known is false.

The *closed world assumption* is used in databases and in the Prolog programming language. It states that a fact is assumed to be false if it cannot be proved from the existing data. This form of reasoning is useful as it allows to draw inferences in the absence of explicit information.

Example 2.1. Consider a knowledge base with only two statements:

$$\mathsf{Teacher}(\mathsf{JOHN}); \mathsf{Teacher}(\mathsf{MARY})$$

If the closed world assumption is used, we conclude that there are only two teachers, and an example query $\mathsf{Teacher}(\mathsf{FRANK})$ gives the answer FALSE.

The closed world assumption is, however, inappropriate in a setting, where it is known that information can be incomplete. The Semantic Web is such an environment with factors such as the large size and the fact that information can change dynamically. Thus, very often it cannot be stated whether complete information is already gathered and the assumption that unknown facts must be false cannot be used.

The *open world assumption* allows only to draw conclusions from the information that is explicitly given. In Example 2.1 the query $\mathsf{Teacher}(\mathsf{FRANK})$ will give an undefined answer, as it is not explicitly stated whether Frank is a teacher or not.

Although the Semantic Web needs to rely on the open world assumption, sometimes it would be beneficial to be able to state that the possessed information is actually complete. Let us consider the knowledge base from Example 2.1. In description logic one can define the completeness of information by using nominals, i.e. listing all members of a concept.

$$\mathsf{Teacher} \equiv \{\mathsf{JOHN}, \mathsf{MARY}\}$$

However, this only works for simple cases and this method is very verbose.

Heflin and Muñoz-Avila [31] propose to include the notion of a local closed world (LCW) in a description logic language. LCW allows to state that the closed world assumption should be used for a specific concept. For example, a statement $LCW(\mathsf{Teacher}(x))$ could be used to say that if an individual x cannot be proved to be a teacher, then it should be inferred that x is not a teacher.

Default logic (see Section 2.3 and Section 3.3) provides a method of introducing a form of the closed world assumption into knowledge bases, which natively work with open worlds. The default for the given example would state that "Unless proved otherwise, x is not a teacher". The notion of local closed world can be expressed using default logic by substituting each $LCW(C)$ statement with a prerequisite-free normal default $\frac{:\neg C}{\neg C}$, which expresses the closed world assumption.

2.2 Distributed Environment

The Semantic Web is a distributed environment with many entities playing different roles. Knowledge in this environment changes dynamically and cannot be centrally controlled. The Internet provides good grounds for running autonomous agents, which perform tasks on behalf of their users.

2.2.1 Multi-Agent Systems

Agents always exist in the context of multi-agent systems, in which they can communicate with each other. Many definitions of an agent can be found in literature [25]. In this work we will use the following definition: "Agents are pieces of software that work autonomously and proactively" [6]. The agent's autonomy can be understood as the lack of external control over the agent. It can be given tasks by the user but the agent makes decisions and performs actions based on its own internal state. The proactiveness of agents manifests itself in the fact that agents do not wait for requests but act when they believe it is necessary. Personal agents act on behalf of the user and perform various tasks such as seeking and comparing information. Agents can communicate with other agents while performing their tasks.

Multi-agent systems emerged from the field of distributed artificial intelligence (DAI), which is concerned with decomposing and distributing complex artificial intelligence tasks [72]. In MAS however, each agent can have its own goal it pursues. Agents can even have contradictory goals such as in a multi-agent marketplace [42], where each agent negotiates with other agents to achieve the best price.

Agents in the Semantic Web use metadata of Web resources combined with ontologies, which provide the context and meaning of descriptions. Ontologies enable the agent to interpret obtained information and communicate with other agents. For the agents to work in the Semantic Web environments, agent communication languages and distributed reasoning algorithms have to be developed.

Taking into account the language for interaction between agents, two layers can be distinguished. Firstly, the *communication language* is responsible for formulating messages and allows to include information such as the sender, recipient or the type of message. Secondly, the actual body of the message is expressed in a *content language*. The content can carry a query, an answer or any information, which can be expressed in the content language. The two most commonly used communication languages are FIPA-ACL (specified by the Foundation for Intelligent Physical Agents) [1] and KQML (Knowledge Query and Manipulation Language) [24]. However, there is no standard content language for agent communication. In the Semantic Web context, an ontology language such as OWL can be used to express information and SPARQL could be used to form queries. These two languages have been chosen in the AgentOWL implementation of Semantic Web agents [46]. A special case of the content language can be the natural language, nevertheless it is not suitable for automatic processing.

In the Semantic Web environment there is an important issue that arises when trying to build a Web-wide agent system. Namely, since the knowledge in the Web is distributed, each agent can use a different ontology for its own purpose and finding a common language with another agent may not be a trivial task.

2.2.2 Knowledge Distribution

Knowledge bases provide the functionality of reasoning in order to derive new facts from existing information and to allow complex queries to be answered. In distributed environments, such as the Internet, it can often be found that relevant knowledge needed for reasoning has to be obtained from several distributed sources. It is the task of distributed reasoning to combine the knowledge that is scattered among the peers in a distributed system in order to reach appropriate conclusions.

Knowledge can become distributed in two main ways. Firstly, large knowledge bases can be split into smaller modules, which is done in order to increase reasoning performance by enabling to run parallel reasoning procedures on each smaller part of the whole knowledge base. Works such as [3] show how to divide an initial knowledge base into smaller partitions in order to run local reasoners on each of the partitions. This approach has the advantage of the ability to divide the knowledge in a predictable way such that the performance gain is maximized. For instance, building a stratified knowledge base is known to increase reasoning performance [20]. Secondly, knowledge can already exist in a distributed environment, for example created and shared by different entities. In this case, to acquire sought information it may be required to contact remote entities which can answer queries. The approach to reasoning in this case is different, as one does not control the remote knowledge bases and one cannot specify how the knowledge is divided. In this scenario, performance gain cannot be guaranteed. The work [2] deals with an already existing network of peers, each containing its local knowledge base. The global knowledge, understood as the sum of all local knowledge bases, is unknown. Due to the possibly large size, the dynamic nature of the network and also privacy constraints, it would be impractical or even impossible to gather all accessible knowledge in one place.

Both of the above scenarios exist in the context of the Semantic Web. Works such as [27] show methods of dividing large ontologies, such as the eClassOWL product and services classification [32] or the SNOMED medical ontology [71]. The task of splitting a large ontology into a set of smaller ones, which are not necessarily disjoint. The aim of this task is to improve the computational complexity of reasoning and to enable running parallel reasoning procedures on each module separately.

On the other hand, in the Web environment the knowledge is inherently distributed and there is no central repository of information. The current World Wide Web already contains hundreds of ontologies, each used for a specific purpose. However, success has yet to be achieved to actually use ontologies to enable communication between systems, which use slightly different ontologies. This task requires a standardized method for mapping ontology vocabularies.

Multiple distributed ontologies require a method for interoperability. Different entities create their own ontologies which suit their needs, often reusing parts of other ontologies. However, ontologies created by different people can have overlapping domains. This does not mean that these ontologies contain concepts which have identical meanings. Even if two ontologies include concepts with the same names, these concepts can express slightly different meanings.

Standardization can be viewed as an attempt to solve the problem of refering to the same concepts by different ontologies. If two ontologies refer to animals, they should link to an appropriate standard ontology of the animal domain. Standardization is undoubtedly necessary in the process of building the Semantic Web. Since the goal of using ontologies is to provide a common vocabulary for different systems, standardized ontologies have to be created and adopted by the peers, who want to communicate with each other. Nevertheless, it is inevitable that new ontologies will be created by different entities that will form their own vocabularies. For example each camera manufacturer has a different name for its image stabilization system (Canon: *Image Stabilizer*, Nikon: *Vibration Reduction*, Pentax: *Shake Reduction*).

The basic method for joining ontologies included in the OWL language is the `owl:import` statement. However, this method of combining ontologies, though simple, has serious drawbacks. The `owl:import` statement causes all imported ontologies to be downloaded as a whole and loaded into the local reasoner. What is more, if the imported ontologies also have dependencies, then every one of the dependent ontologies is recursively added to the local ontology in order to perform a reasoning task on a combined *global* knowledge base. In simple cases this might be an acceptable solution but it does not scale very well. Apart from that, it is often required to actually reuse only several terms from the imported ontology, so it makes it even more inefficient to load the remote ontology entirely, especially when there are many unnecessary axioms. Another drawback of ontology importing is the lack of methods for ensuring privacy of local knowledge. The only option is to share an entire ontology without having an option to selectively publish only its parts and requiring authorization for restricted knowledge.

One solution to this problem is to import only those parts of a remote ontology that are actually going to be used. Grau et al. [27] propose an algorithm for determining and retrieving the fragment of an ontology that is relevant for a given concept.

In a distributed environment such as a multi-agent system, each peer can have its own local knowledge. By means of exchanging information the peers can perform inferences involving additional knowledge from other sources. An agent will often have its own domain of expertise and it will contact other agents in order to gather knowledge from other domains. In fact, different knowledge bases can have overlapping content. The common concepts can be used as the communication language between two peers.

Distributed Description Logic

Distributed Description Logic (DDL) [17] is an extension to the Description Logic formalism, which provides semantic mappings between ontologies via bridge rules. A bridge rule connects a concept from one ontology with a concept from another ontology, stating that one is subsumed by the other. This means that one ontology refers only to selected concepts from other ontologies instead of including all axioms from remote sources. Such semantic mapping makes it possible to construct algorithms for distributed reasoning, where local reasoning processes are combined to answer a query. Moreover, since a peer does not expose its ontologies publicly but only offers a reasoning service, it is possible to implement privacy ensuring methods on the concept level.

Package-based Description Logic

Bao et al. [13] propose a formalism named Package-based description logic (P-DL), which supports contextual reuse of knowledge from multiple DL knowledge bases. This formalism is based on the $\mathcal{SHOIQ}$ DL language. Each ontology is viewed as a package connected with a local domain. Every concept, role and individual name have their *home package*, which is the source of this name. The authors restrict negation ($\neg$) and the top concept ($\top$) to one module, denoting them as $\neg_i$ and $\top_i$.

2.3 Default Logic

Many knowledge representation formalisms, including description logic and ontologies, suffer the lack of support for reasoning with incomplete or inconsistent information.

Very often it is actually impossible to obtain complete information in a domain being described. First-order logic and in consequence description logics reasoning follow the open world assumption (OWA). This means that if some information is not explicitly expressed or derivable from the facts, there is no other mechanism to use this information. In contrast, the closed world assumption (CWA) says that if something cannot be proved, it is taken to be false.

In order to be able to reason with incomplete data, the logic used to represent knowledge has to be augmented by a nonmonotonic reasoning mechanism. Such mechanisms include negation-as-failure [21] used for example in Prolog [54], circumscription [50], Nute's defeasible logic [55] and Reiter's default logic [59, 57]. All these formalisms allow nonmonotonic reasoning, which provides the possibility to remove inferences by adding new information to the knowledge base. A *default* in Reiter's default logic can express the notion that "Typically, if α, then β", which suggests that there can exist exceptions to the rule that usually will be applicable.

The formalism of default logic asusmes that there can be many worlds, called *extensions*, containing contradictory results of reasoning. There has been significant effort to identify one preferred extension in order to give only one specific answer to a query. Ryżko [62] proposes prioritizing defaults based on a confidence measure, which is learnt by a multi-agent system. Katz and Golbeck [44] calculate priority values of defaults based on trust in social networks. In this work, however, we address finding all extensions of a default theory. Nonetheless, it can be extended to utilize one of the formalisms to prefer extensions.

2.3.1 Distributed Default Logic

Distributed reasoning with defaults is introduced in [62] and further developed in [64]. This approach utilizes environmental knowledge, which is added to the defaults in the knowledge base and forming default templates. This work does not take into account the problems occurring in the Semantic Web, in which the distribution of knowledge cannot be controlled and particularly stratification of the modules is not possible. Moreover, agents can use different vocabularies in their local knowledge bases.

2.3.2 Embedding Defaults in Description Logic

For default logic to be applicable to the Semantic Web environment, it has to be tied to a knowledge representation language used in this context. The OWL language and description logics have dominated the Semantic Web arena and there have been attempts to introduce nonmonotonic reasoning to the DL formalism. Antoniou [5, 7] proposed to integrate defeasible logic into description logic. These works introduce *defeasible rules*, which can be overridden by strict rules.

A different example of introducing defeasible reasoning in description logics is presented by Heymans and Vermeir [33]. This work proposes to include a preference order on the axioms of a knowledge base. With such a strict partial order, some axioms can override other less preferred ones.

Baader and Hollunder [12] propose a formalism, which is used in this work, to embed default logic into description logic. A more detailed characterization of their results is presented in Section 3.4. Kolovski [45] presents an implementation of the integration of defaults into the OWL language. This work, however, does not take into account distributed reasoning.

Chapter 3

Basic Concepts and Definitions

3.1 Description Logics

Description logics (DLs) are a family of knowledge representation formalisms, which are based on first-order logic. A DL knowledge base consists of two components. The *terminology* (Tbox) is a set of axioms defining and describing *concepts* and *roles*. Complex concepts and roles are defined using atomic ones. The *assertional component* (Abox) contains facts about objects expressed using concepts and roles.

One of the simplest and much studied description logic is $\mathcal{ALC}$ [66]. This language provides a set of constructors for defining new concepts shown in Table 3.1. The table also shows the analogy to first-order logic as concepts can be interpreted as unary predicates, roles as binary predicates and individuals as constants.

3.1.1 Definitions

A formal semantics is defined by considering interpretations $\mathcal{I}$ consisting of a domain $\Delta^{\mathcal{I}}$ and an interpretation function assigning to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation of the DL constructs are defined as follows:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall_b (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists_b (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}
\end{aligned}
$$

The terminology is defined using the concept subsumption relation ($\sqsubseteq$) and the equivalence relation ($\equiv$). The semantics of these relations is as follows. Two concepts C, D are equivalent ($C \equiv D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretations. A concept C is subsumed by a concept D ($C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations.

The TBox is a set of general concept inclusions (GCI) in the form $C \sqsubseteq D$ and concept definitions ($A \equiv C$). Concepts can be atomic or complex. Complex concepts are built using DL constructors (see Table 3.1). It is assumed that only one definition of a concept is allowed and definitions are *acyclic*, which means that a concept is never directly or indirectly used in its own definition.

constructor	first-order logic analogy	description
A	$A(x)$	concept name
R	$R(x,y)$	role name
$\top$	TRUE	top
$\bot$	FALSE	bottom
$C \sqcap D$	$C(x) \wedge D(x)$	conjunction
$C \sqcup D$	$C(x) \vee D(x)$	disjunction
$\neg C$	$\neg C(x)$	complement*
$\forall R.C$	$\forall y\ R(x,y) \Rightarrow C(y)$	universal quantifier
$\exists R.C$	$\exists y\ R(x,y) \wedge C(y)$	existential quantifier

Table 3.1: $\mathcal{ALC}$ constructors. C, D – concepts.

Example description logic constructs (TBox) are as follows:

$$\mathsf{Parent} \equiv \mathsf{Father} \sqcup \mathsf{Mother}$$
$$\mathsf{Student} \sqsubseteq \mathsf{Person} \sqcap \exists \mathsf{attends}.\mathsf{University}$$

Example ABox assertions:

$$\mathsf{Father}(\mathsf{JOHN})$$
$$\mathsf{attends}(\mathsf{JOHN}, \mathsf{WUT})$$

The semantics of a DL knowledge base in the form of a TBox and an ABox makes it equivalent to a set of first-order logic axioms. Thus, it contains implicit knowledge that can be obtained through inferences. The primary inference problems include consistency checking. This is an important inference task, as all other reasoning tasks can be reduced to consistency checking. The main inference problems concerning concepts, all of which can be reduced to subsumption, are:

- subsumption ($C \sqsubseteq D$)
- satisfiability ($C \not\sqsubseteq \bot$)
- equivalence ($C \sqsubseteq D$ and $D \sqsubseteq C$)
- disjointness ($C \sqcap D \sqsubseteq \bot$)

The subsumption problem ($C \sqsubseteq D$) can in turn be reduced to concept satisfiability by testing whether the concept $C \sqcap \neg D$ is unsatisfiable.

Apart from TBox inferences, there is also an important inference problem for the ABox, which is *instance checking* – checking whether a given individual is an instance of a given concept. It can be shown that an individual a is an instance of concept C, which is denoted as $C(a)$, if adding the assertion $\neg C(a)$ makes the knowledge base inconsistent.

A concept is in negation normal form (NNF) if negation ($\neg$) only appears directly in front of atomic concepts. It is shown in [10] that for each complex concept description an equivalent NNF exists.

3.1.2 Tableau reasoning algorithm

A tableau algorithm for description logics was first presented by Schmidt-Schauß and Smolka [66] for the $\mathcal{ALC}$ DL. This algorithm has been further extended to more expressive DL languages [34, 8, 11, 9, 65, 38, 29, 39].

The tableau algorithm for TBox inferences reduces the presented query to concept satisfiability. The algorithm works by trying to construct a finite interpretation $\mathcal{I}$ such that for the tested concept C, $C^{\mathcal{I}} \neq \emptyset$. An individual a is created, which belongs to the concept C, i.e. $a \in C^{\mathcal{I}}$. If C is a complex description, rules are applied to break it down into assertions on atomic concepts. If no more rules can be applied and contradictory statements $A(x)$ and $\neg A(x)$ for some x are encountered, then concept C is found unsatisfiable. If a disjunction of two concepts $A \sqcup B(x)$ is analyzed, the algorithm has to check both possibilities $A(x)$ and $B(x)$ independently. If at least one path does not lead to a contradiction, then the queried concept is said to be satisfiable.

The tableau algorithm works on a data structure, called a tableau, which is a set of ABox assertions in the form $C(x)$ and $R(x, y)$, where C is a concept description, R is a role name and x and y are individual names. To test the satisfiability of the concept C, the algorithm starts with the ABox $\mathcal{A}_{\prime} = \{C(a)\}$, where a is a new individual name. The procedure applies transformation rules to the ABox until no more rules can be applied or the ABox contains an obvious contradiction, which is called a *clash*. In the first case, $\mathcal{A}_0$ is consistent and C is satisfiable, whereas in the second case $\mathcal{A}_0$ is inconsistent and C is unsatisfiable. The transformation rule, which applies to the disjunction of concepts is non-deterministic, meaning that the input ABox is transformed into a set of ABoxes with the property that the input ABox is consistent if and only if at least one of the output ABoxes is consistent. This leads to maintaining a set of ABoxes $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ instead of only one. $\mathcal{S}$ is said to be consistent if and only if there exists some i such that $\mathcal{A}_i$ is consistent.

The $\rightarrow_{\sqcap}$-rule:
If $\mathcal{A}$ contains $(C \sqcap D)(x)$, but it does not contain both $C(x)$ and $D(x)$,
then $\mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$.

The $\rightarrow_{\sqcup}$-rule:
If $\mathcal{A}$ contains $(C \sqcup D)(x)$, but neither $C(x)$ nor $D(x)$,
then $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{D(x)\}$.

The $\rightarrow_{\exists}$-rule:
If $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual name z such that $R(x, z)$ and $C(z)$ are in $\mathcal{A}$,
then $\mathcal{A}' = \mathcal{A} \cup \{R(x, y), C(y)\}$, where y is an individual name not occurring in $\mathcal{A}$.

The $\rightarrow_{\forall}$-rule:
If $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but it does not contain $C(y)$,
then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

Figure 3.1: Tableau transformation rules

The algorithm starts with $\mathcal{S}$ containing the starting ABox $\mathcal{A}_0$. In each step, one element $\mathcal{A} \in \mathcal{S}$ is taken and a rule of Figure 3.1 is applied obtaining one ($\mathcal{A}'$) or two ($\mathcal{A}', \mathcal{A}''$) ABoxes. $\mathcal{A}$ is then replaced by $\mathcal{A}'$ or $\mathcal{A}'$ and $\mathcal{A}''$. The algorithm finishes if there is some $\mathcal{A} \in \mathcal{S}$ for which no transformation rules can be applied or all ABoxes in $\mathcal{S}$ contain clashes.

3.2 Distributed Description Logic

Let I be a set of indexes and we have a set of description logic knowledge bases KB_i for $i \in I$. The set $\mathcal{B}_{ij}$ contains bridge rules B_{ij}. Bridge rules in $\mathcal{B}_{ij}$ represent the point of view of KB_j and are only used by knowledge base KB_j. Note that bridge rules $\mathcal{B}_{ji}$ may be different from $\mathcal{B}_{ij}$. Let us denote concepts from knowledge base KB_i as $i : C$.

Definition 3.1. *A* **bridge rule** B_{ij} *is in one of the forms:*

$$i : A \xrightarrow{\sqsubseteq} j : G$$

$$i : A \xrightarrow{\sqsupseteq} j : G$$

$$i : A \xrightarrow{\equiv} j : G$$

where $i, j \in I$ and A and G are concepts. The last bridge rule is defined as the conjunction of the first two rules. □

To fully define distributed description logic the individual correspondence relation needs to be introduced.

Definition 3.2. Individual correspondence *is an expression $i : x \mapsto j : y$, where x is an individual in KB_i and y is an individual in KB_j.*

Definition 3.3. A DDL knowledge *base $\mathcal{KB} = \{\mathrm{KB}_i\}$ for $i \in I$ is a set of local knowledge bases $\mathrm{KB}_i = \langle T_i, A_i, B_i \rangle$ where $\langle T_i, A_i \rangle$ is a DL knowledge base and B_i is a set of bridge rules B_{ji} for different values of $j \in I$ $(i \neq j)$.* □

The formal semantics of DDL is provided using local interpretations of individual DL knowledge bases, as defined in Section 3.1.

A distributed interpretation $\mathfrak{I} = \langle \{\mathcal{I}_i\}_{i \in I}, \mathbf{r} \rangle$ of $\mathcal{KB}$ consists of interpretations $\mathcal{I}_i$ for KB_i over domain $\Delta^{\mathcal{I}_i}$ and a function $\mathbf{r}$ associating a binary relation $\mathbf{r}_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$ to each $i, j \in I$.

3.3 Default Logic

Default logic has been introduced by Reiter [59] as an approach to commonsense reasoning. It can be used to deal with the inability to fully describe the world and to provide more concise representations of knowledge due to the form of specifying exceptions to defaults. The notion of a *default* is introduced, which acts as an inference rule on the current set of beliefs.

3.3.1 Definitions

The following definitions present the formalism of default logic as it was defined by Reiter.

Definition 3.4. *A* **default** *is in the form:*

$$\frac{\alpha : \beta}{\gamma}$$

where α, β and γ are well-formed formulae. α is the **prerequisite***, β is the* **justification** *and γ is the* **consequent***.* □

The default can be applied and the consequent inferred if the prerequisite can be proved and the justification is consistent with the current knowledge.

The original default logic proposed by Reiter allowed multiple justifications written as $\frac{\alpha \,:\, \beta_1, \ldots, \beta_n}{\gamma}$. Here, we limit the number of defaults to one, following most further works on default logic. In this case, multiple justifications can be obtained by joining them with conjunction, i.e. $\frac{\alpha \,:\, \beta_1 \wedge \ldots \wedge \beta_n}{\gamma}$.

For a default d, let $Pre(d)$, $Jus(d)$ and $Con(d)$ denote the formulae occurring in the prerequisite, justification and consequent, respectively, of the default d.

Similarly, for a set of defaults D, let:

$$PRE(D) = \{Pre(d) : d \in D\}$$

$$JUS(D) = \{Jus(d) : d \in D\}$$

$$CON(D) = \{Con(d) : d \in D\}$$

The following definition defines defaults of special forms, that will be of interest in our considerations.

Definition 3.5. *We define two forms of defaults:*

A **normal** *default is in the form*

$$\frac{\alpha : \gamma}{\gamma}$$

A **semi-normal** *default is in the form*

$$\frac{\alpha : \beta \wedge \gamma}{\gamma}$$

□

A normal default is usually read as "Typically, if α then β".

We say that a default is **closed** if it contains no free variables.

Definition 3.6. *A* **default theory** *is a pair $\Delta = \langle D, W \rangle$, where W is a set of first-order logic formulae creating a world description and D is a set of defaults. The default theory is* **closed** *if it contains only closed defaults.* □

The consequences of a default theory are defined using the notion of an *extension*, which is a set of deductively closed formulae. In general, a default theory can have more than one extension or even no extensions. However, a default theory consisting only of normal defaults (normal default theory) guarantees having at least one extension. The following definition shows a non-deterministic iterative process of obtaining extensions of a default theory. In each step a default is used to add the consequent to the resulting set of formulae. An extension is defined by the fixed point of this process. By $Th(E)$ we denote the deductive closure of a set of formulae E.

Definition 3.7. *Let E be a set of closed formulae, and $\langle D, W \rangle$ be a closed default theory. Let*

$$E_0 = W$$

$$E_{i+1} = E_i \cup \left\{ \gamma \mid \frac{\alpha : \beta}{\gamma} \in D, \alpha \in Th(E_i) \text{ and } \neg\beta \notin Th(E) \right\}$$

$Th(E)$ is an **extension** *of $\langle D, W \rangle$ iff*

$$Th(E) = \bigcup_{i=0}^{\infty} Th(E_i)$$

□

Different extensions can be generated depending on the order the defaults are applied. By $ext(\Delta)$ we denote the set of all extensions of the default theory Δ.

Example 3.1. Consider a default theory $\Delta = \langle D, W \rangle$, where

$$D = \left\{ \frac{q \ : \ p}{p}, \frac{r \ : \ \neg p}{\neg p} \right\}$$
$$W = \{n \rightarrow q, \ n \rightarrow r, \ n\}$$

This represents the classic Nixon diamond problem, where n – Nixon, q – Quaker, r – Republican and p – Pacifist. This theory has two extensions $E_1 = \{n, q, r, p\}$ and $E_2 = \{n, q, r, \neg p\}$. The extensions provide two views of the world, where Nixon is a pacifist in one and not in the other.

The consequences of a default theory can be defined by employing *skeptical* or *credulous* reasoning.

Definition 3.8. *Given the default theory Δ, a formula ϕ is a consequence of* **skeptical reasoning** *in Δ if it is in at least one extension, i.e. $\exists_{E \in ext(\Delta)} \phi \in E$. A formula ϕ is a consequence of* **credulous reasoning** *in Δ if it is in all extensions, i.e. $\forall_{E \in ext(\Delta)} \phi \in E$.*

In Example 3.1, both p and $\neg p$ are consequences of credulous reasoning, however neither p nor $\neg p$ is a consequence of skeptical reasoning.

Definition 3.9. *The set of* **generating defaults** *for an extension E of theory $\Delta = \langle D, W \rangle$ is the set*

$$GD(E, \Delta) = \{\alpha : \beta / \gamma \in D | \alpha \in E \text{ and } \neg \beta \notin E\}$$

□

Each extension has its unique set of generating defaults. The extensions of the theory from Example 3.1 have the following generating defaults:

$$GD(E_1, \Delta) = \left\{ \frac{q \ : \ p}{p} \right\}$$
$$GD(E_2, \Delta) = \left\{ \frac{r \ : \ \neg p}{\neg p} \right\}$$

During the iteration process from Definition 3.7 only consequences of defaults are added to the extensions allowing to express any extension in the form

$$E = Th(W \cup CON(GD(E, \Delta))) \tag{3.1}$$

This relation is used in the reasoning algorithm. Having found the sets of generating defaults for all extensions, the consequences of the generating defaults combined with W constitute the resulting extension.

3.3.2 Default Reasoning

The primary task of default reasoning is computing the extensions of a default theory. Once the extensions are obtained, it is easy to make credulous or skeptical inferencs. Only closed defaults will be considered, as explained in the next section about embedding defaults into description logic. In this section, a method for computing the extensions proposed by Risch and Schwind [61] will be described. Other methods of finding default extensions include directly applying the definition of an extension, which results in a very inefficient exponential complexity. A method proposed by Junker and Konolige [41] should also be mentioned, which translates a default theory into a Truth Maintanance Network.

The algorithm by Risch and Schwind is based on the definition of a grounded set of defaults and a theorem describing the set of generating defaults of an extension. The result of the algorithm is the set of extensions expressed as their sets of generating defaults.

Definition 3.10. *Let W be a set of closed formulae, and D be a set of closed defaults. Let*

$$D_0 = \emptyset$$

$$D_{i+1} = D_i \cup \left\{ d = \frac{\alpha : \beta}{\gamma} \mid d \in D \text{ and } W \cup CON(D_i) \models \alpha \right\}$$

*D is **grounded** in W iff*

$$D = \bigcup_{i=0}^{\infty} D_i$$

This definition can be directly used as a procedure for finding grounded sets of defaults. Moreover, if D is not grounded, then $\bigcup_{i=0}^{\infty} D_i$ is the largest subset of D that is grounded in W.

Theorem 3.1. (Risch and Schwind [61]) *Let $\Delta = \langle D, W \rangle$ be a closed default theory. A subset $\hat{D}$ of D is a set of generating defaults of an extension of Δ iff the following two conditions hold:*

1. *$\hat{D}$ is grounded in W*
2. *For all $d = \frac{\alpha:\beta}{\gamma} \in D$ we have:*
   (a) *$d \in \hat{D}$ iff $W \cup CON(\hat{D}) \models \alpha$, and*
   (b) *$W \cup CON(\hat{D}) \not\models \neg\beta$*

The theorem constitutes a test of whether a subset $\hat{D}$ of D is a set of generating defaults of an extension of the default theory $\Delta = \langle D, W \rangle$. Without loss of generality, we assume that W is consistent. Algorithm 3.1 computes the generating sets of defaults of all extensions of Δ. Let D_0 be the largest subset of D that is grounded in W and $D_1, \ldots, D_n$ be all maximal subsets of D_0 such that $W \cup CON(D_i)$ is consistent. The sets D_i have the property that each set of generating defaults is a subset of one of the D_i.

Algorithm 3.1: allExtensions

```
Input: Theory $\Delta = \langle D, W \rangle$
begin
    result $\leftarrow \emptyset$;
    foreach maximal subset $D'$ of $D$ such that $W \cup CON(D')$ is consistent do
        result $\leftarrow$ result $\cup$ removeDefaults($\Delta$, $D'$);
    return result;
```

3.4 Description Logic with Defaults

A description logics knowledge base consists of only universal statements, which do not allow exceptions. This allows the reasoning system to unambiguously assign individuals

Algorithm 3.2: removeDefaults

```
Input: Theory $\Delta = \langle D, W \rangle$
Input: Potential superset of a generating set of defaults $D'$
begin
    $D_0 \leftarrow$ largest subset of $D'$ that is grounded in $W$;
    if there exists $d = \frac{\alpha:\beta}{\gamma} \in D_0$ such that $W \cup CON(D_0) \models \neg Jus(d)$ then
        result $\leftarrow$ removeDefaults($\Delta$, $D_0 \setminus d$);
        foreach maximal subset $D'$ of $D_0$ such that
                $d \in D'$ and $W \cup CON(D') \not\models Jus(d)$ do
            result $\leftarrow$ result $\cup$ removeDefaults($\Delta$, $D'$) ;
        return result;
    else if for each $\frac{\alpha:\beta}{\gamma} \in D \setminus D_0$ either
            $W \cup CON(D_0) \not\models \alpha$ or
            $W \cup CON(D_0) \models \neg\beta$ then
        return $D_0$;
    return $\emptyset$;
```

to concepts. However, this method does not provide means for commonsense reasoning, where some assumptions can eventually be shown to be false. The application of the results achieved in default logic can provide a method for commonsense reasoning without losing important features of description logics. Reiter's default logic uses first-order logic as the base language and since description logics are decidable subclasses of FOL, they can be extended with the notion of defaults using the original semantics. Baader and Hollunder [12] show how defaults can be embedded into description logics.

Defaults for description logics are formed analogously to the original defaults syntax. Here, the formulas for the prerequisite, justification and conclusion are replaced by DL concepts. A default is in the form

$$\frac{A : B}{C}$$

where A, B and C are concept expressions. This default is equivalent to the default in which concepts are expressed as unary predicates:

$$\frac{A(x) : B(x)}{C(x)}$$

The default expresses that it can be inferred that x is an instance of the concept C if x is an instance of A and it is consistent to assume that x is an instance of B.

Embedding defaults in description logics is not as straightforward as it may seem. The problem is with treatment of open defaults by Skolemization. An $\mathcal{ALC}$ knowledge base is undecidable, unless we consider only closed defaults. This means that defaults can only be applied to named individuals which already exist in the knowledge base.

The following example shows the application of defaults only to individuals that explicitly appear in the ABox.

Example 3.2. Consider a knowledge base consisting of a fact that *Tom has a child that is a doctor* and that *typically doctors are rich.*

$$\frac{\mathsf{Doctor} : \mathsf{Rich}}{\mathsf{Rich}} \quad (\exists \mathsf{hasChild.Doctor})(\mathsf{TOM})$$

One might expect that it could be inferred that Tom has a child that is rich. However, the default cannot be applied to Tom's child, which is an implicit individual and does not exist int the ABox. Thus, the conclusion $(\exists\mathsf{hasChild.Rich})(\mathsf{TOM})$ cannot be reached.

A normal default in the form $\frac{A:B}{B}$ can be seen as a weaker form of subsumption, such that permits exceptions. The default $\frac{A:\top}{B}$ is also weaker than the axiom $A \sqsubseteq B$ because although there is no possibility of specifying exceptions, it does not imply the contrapositive $\neg B \sqsubseteq \neg A$.

Definition 3.11. *A DL$\mathfrak{D}$ knowledge base is a triple* KB $= \langle D, T, A\rangle$ *where T is a TBox, A is an ABox and D is a set of defaults.* □

Because of problems caused by Skolemization, which are described in [12], restricted semantics has to be used. Defaults are only applied to individuals that exist in the ABox forming a finite set of closed defaults.

Chapter 4

Distributed Description Logic with Defaults

One of the main reasoning tasks in description logics is subsumption checking. The query of whether a concept B subsumes a concept A is stated in description logic language as $A \sqsubseteq B$, which corresponds to the statement $A(x) \rightarrow B(x)$ in first-order logic. Such queries are useful when reasoning in a distributed environment such as a multi-agent system. For instance, the distributed reasoning algorithm presented in [68] exchanges such queries between the peers in a Distributed Description Logic system. In this setting the answer to a query can only be TRUE or FALSE.

Taking into account the integration of defaults into distributed reasoning, giving such a definitive answer to a subsumption query does not always fully express the knowledge contained in a remote knowledge base. Moreover, it can cause loss of valuable information about the assumptions made during reasoning.

Exchanging knowledge between agents in an efficient way requires an agent to answer a question as precisely as possible. When agent A asks agent B whether it believes that formula ϕ is true, it expects a short answer whether ϕ is true or not. However, when using defaults in the reasoning process, the answering agent might use defaults while finding the answer to a question, which leads to making possibly wrong assumptions. Giving a strict answer of TRUE or FALSE would make the asking agent interpret the answer as "Agent B **believes** that ϕ is true (false)" while agent B only **assumes** that ϕ is true (false).

In order to deal with such situations, we propose to include the assumptions made during reasoning in the answers to queries. As these assumptions are contained in defaults, we introduce the possibility to generate defaults as reasoning results, which are ready to be communicated between agents. This is an extension of a preliminary solution, which was introduced in [75].

4.1 Motivating example

Let us start with an example to illustrate the problems arising when default reasoning is used in a distributed system.

Example 4.1. Let us consider an example DL$\mathfrak{D}$ knowledge base which can be a part of a distributed system.

$$d_1 : \frac{\mathsf{Bird} : \mathsf{Flies}}{\mathsf{Flies}} \qquad \begin{array}{ll} \mathsf{Stork} \sqsubseteq \mathsf{Bird} & \mathsf{Stork(SAM)} \\ \mathsf{Goose} \sqsubseteq \mathsf{Bird} & \mathsf{Stork(TIM)} \\ \mathsf{Penguin} \sqsubseteq \mathsf{Bird} & \neg\mathsf{Flies(TIM)} \\ \mathsf{Penguin} \sqsubseteq \neg\mathsf{Flies} & \mathsf{Penguin(PAT)} \end{array}$$

Following queries to this knowledge base:

Case 1

$$Q1 : \mathsf{Flies(TIM)}$$
$$A1 : \textsc{False}$$

The answer to this query is False because there is a straightforward fact in the knowledge base stating that TIM cannot fly.

Case 2

$$Q2 : \mathsf{Flies(SAM)}$$
$$A2 : \textsc{True}$$

A positive answer is returned by applying the default d_1.

Case 3

$$Q4 : \mathsf{Goose} \sqsubseteq \mathsf{Flies}$$
$$A4 : \textsc{Undefined}$$

Unlike the previous case, the default cannot be applied without a concrete individual taken into account, because only closed defaults can be treated by the reasoning algorithm. The lack of explicit knowledge stating that the concept Goose subsumes the concept Flies prevents from giving a definitive answer.

Case 4

$$Q3 : \mathsf{Stork} \sqsubseteq \mathsf{Flies}$$
$$A3 : \textsc{Undefined}$$

Similarly to the previous case, the answer cannot be unambiguously stated. If this query is interpreted as “Do all storks fly?”, the answer could be negative, as TIM is an example of a stork that does not fly. However, since the reasoner only operates on explicit subsumption statements and does not analyze individuals, such a result cannot be reached. □

From this example we can see that the answers to queries $Q2 - Q4$ lose information which exists in the form of the default. In a distributed system, where many knowledge bases may contain the sought information, making early assumptions can lead to hasty conclusions. In these cases, answers with the following meanings could be expected:

A2′. It is assumed that SAM *flies unless it is proved otherwise.*

A3′. Typically, geese fly unless it is proved otherwise.

A4′. Typically, storks fly unless it is proved otherwise.

These statements can be expressed as the following defaults:

$$A2' : \frac{:\mathsf{Flies(SAM)}}{\mathsf{Flies(SAM)}}$$
$$A3' : \frac{\mathsf{Goose} : \mathsf{Flies}}{\mathsf{Flies}}$$
$$A4' : \frac{\mathsf{Stork} : \mathsf{Flies}}{\mathsf{Flies}}$$

These defaults, if provided as answers, give more information from the original knowledge base than usual answers from the default theory. The rules form a concise intermediate result and can be applied to achieve the final answer. In a distributed environment, the triggering of these rules will occur on the side of the asking agent. Its own knowledge gathered from other sources can be useful for providing justifications for defaults or rejecting defaults based on provided exceptions to defaults.

As shown in [60], combining defaults and implication, although semantically correct, can lead to conclusions that are not intended. For example, the statements *Typically adults are married* and *18-year-olds are adults* leads to a default *Typically 18-year-olds are married.* However, such situations can be solved by adding additional defaults or adding justifications to existing ones.

In a distributed environment, the fact that only closed defaults have to be used in order to provide inferences is very limiting. This would imply that two agents sharing knowledge have to have a common set of individuals and one agent would not be able to ask another agent about a general relationship $A(x) \rightarrow B(x)$ without instantiating the variable x.

In the next section we will discuss what transformations can be applied to defaults while an agent prepares answers in the form of defaults.

4.2 Transforming defaults

When default logic is used in a distributed reasoning system, strict True or False answers to queries may cause loss of information, as an agent may only *assume* that the answer it gives is correct. Thus, an answer to an agent's query should also carry the information about assumptions made during the reasoning process. In default logic, assumptions are expressed through the use of justifications in defaults. By tracing the justifications of defaults that would be triggered when trying to prove a formula, additional information can be collected and further used in answers to queries.

For a question in the form $a \rightarrow b$ (or $A \sqsubseteq B$ in the language of description logics) the answer can be given from the set $\{$True, False, True *with justifications* j $\}$, where the last answer is interpreted as a default $\frac{a:b\wedge j}{b}$ and j is the set of justifications that would be checked in defaults that would be applied to prove the queried formula $a \rightarrow b$.

Defaults in Reiter's default logic are treated as inference rules on the same level of reasoning as *modus ponens* or *modus tollens*. In the basic form the inference methods do not permit creating new inference rules as the result of reasoning. Example 4.1 shows that returning defaults as the result of reasoning can be beneficial by making answers to queries more informative. Only semi-normal defaults are considered because using general defaults would lead to the possibility of generating defaults that would make the theory inconsistent.

In order to be able to generate answers in the form of defaults, a mechanism is needed to create new defaults based on the current knowledge base. Such rules must have the property that when they are added to the set of defaults of a default theory, the theory does not change with respect to the results of reasoning. In other words, the set of extensions of the default theory $ext(\Delta)$ must remain unchanged.

Definition 4.1. *A* **default transformation** $t : \Delta \rightarrow \mathcal{D}$ *produces a new default* δ *from a default theory* $\Delta = \langle D, W \rangle$ *and is denoted by* $\Delta \mid\!\sim \delta$. □

We define a set of transformations which have very useful features and will be used in the process of default reasoning. A general form of a transformation is $\langle D_t, f_t \rangle \mid\!\sim \delta$, where $D_t \subseteq D$, $W \models f_t$, and δ is a new concluded default.

Definition 4.2. *Given well-formed formulae a, b, c, d, e, we define the following transformations:*

a). Prerequisite substitution
$\left\langle\{\frac{a:b\wedge c}{b}\}, d \to a\right\rangle \quad \mid\!\sim \frac{d:b\wedge c}{b}$

b). Consequent substitution
$\left\langle\{\frac{a:b\wedge c}{b}\}, b \to e\right\rangle \quad \mid\!\sim \frac{a:b\wedge c\wedge e}{e}$

c). Justification reduction
$\left\langle\{\frac{a:b\wedge c\wedge d}{b}\}, a \to d\right\rangle \mid\!\sim \frac{a:b\wedge c}{b}$

d). Default transitivity
$\left\langle\{\frac{a:b\wedge c}{b}, \frac{b:e\wedge f}{e}\}, \top\right\rangle \mid\!\sim \frac{a:b\wedge c\wedge e\wedge f}{b\wedge e}$

The set of transformations (a)–(d) will be called *basic transformations.* These transformations can be further used in the communication process. Let us start with the following lemma.

Lemma 4.1. *Given default theories $\Delta = \langle D, W\rangle$ and $\Delta' = \langle D \cup \delta, W\rangle$, where δ is obtained using basic transformation $t = \langle D_t, f_t\rangle \mid\!\sim \delta$, and E is an extension of Δ', if the set of generating defaults $GD(E, \Delta')$ includes δ, it also contains D_t, i.e.*

$$\delta \in GD(E, \Delta') \to D_t \subseteq GD(E, \Delta')$$

Proof. By Definition 3.9, $\delta \in GD(E, \Delta')$ iff

$$Pre(\delta) \in E, \text{ and}$$
$$\neg Jus(\delta) \notin E$$

and $D_t \subseteq GD(E, \Delta')$ iff

$$PRE(D_t) \subseteq E, \text{ and}$$
$$\forall_{\beta \in JUS(D_t)} \neg\beta \notin E$$

To prove the lemma, we will prove that for each basic transformation the following two conditions hold:

$$Pre(\delta) \in E \to PRE(D_t) \subseteq E \tag{4.1}$$

$$\neg Jus(\delta) \notin E \to \forall_{\beta \in JUS(D_t)} \neg\beta \notin E \tag{4.2}$$

Firstly, let us show that each basic transformation fulfills condition 4.1. Since $W \models f_t$ and $Th(W) \subseteq E$, then $f_t \in E$. For the basic transformation (a) this shows $d \to a$. For the basic transformations (b)–(d), $Pre(\delta) \in PRE(D_t)$.

Condition 4.2 can be generalized as:

$$\neg Jus(\delta) \notin E \to \neg \bigwedge_{\beta \in JUS(D_t)} \beta \notin E$$

and rewritten as:

$$\neg \bigwedge_{\beta \in JUS(D_t)} \beta \in E \to \neg Jus(\delta) \in E$$

This is true if

$$\neg \bigwedge_{\beta \in JUS(D_t)} \beta \to \neg Jus(\delta)$$

and further

$$Jus(\delta) \to \bigwedge_{\beta \in JUS(D_t)} \beta \tag{4.3}$$

Condition 4.3 trivially holds for basic transformations (a), (b) and (d). It also holds for basic transformation (c) knowing that $a \in E$ and $W \models a \to d$.

Having proven conditions 4.1 and 4.2 this end the proof of the lemma.

$\square$

Example 4.2. Let us illustrate the effect of Lemma 4.1 on the first basic transformation. Let us take the initial default theory $\Delta = \langle D, W \rangle$, where $D = \{\frac{a:b\wedge c}{b}\}$ and $W = \{d \rightarrow a\}$. Using the basic transformation (a) we construct the new default theory $\Delta' = \langle D', W \rangle$, where $D' = D \cup \{\frac{d:b\wedge c}{b}\}$.

Let us now consider two modifications to the default theory Δ'.

$$\Delta_1 = \langle D', W \cup \{d\} \rangle$$

$$\Delta_2 = \langle D', W \cup \{a\} \rangle$$

The default Δ_1 theory has exactly one extension $E_1 = \{a, b, d\}$ with the generating set of defaults $GD(E_1, \Delta_1) = \{\frac{a:b\wedge c}{b}, \frac{d:b\wedge c}{b}\}$.

The default Δ_2 theory has exactly one extension $E_2 = \{a, b\}$ with the generating set of defaults $GD(E_2, \Delta_2) = \{\frac{a:b\wedge c}{b}\}$.

This shows that the new default only appears in a generating set where the source default also exists. However, the inverse is not true as the set of generating defaults of E_2 contains only the source default.

Lemma 4.2. *Given the default theories $\Delta = \langle D, W \rangle$ and $\Delta' = \langle D \cup \delta, W \rangle$, where δ is obtained using basic transformation $t = \langle D_t, f_t \rangle \mid\!\sim \delta$, and E is an extension of Δ', the following holds:*

$$Con(\delta) \in Th(CON(D_t) \cup W)$$

Proof. For the basic transformations (a), (c), (d) $Con(\delta) \in CON(D_t)$. For (b) the lemma holds, knowing that $W \models b \rightarrow e$ □

Let us introduce the order relation between extension sets. We say that $ext(\Delta) \preceq ext(\Delta')$ for

$$ext(\Delta) = \{E_1, \ldots, E_n\}$$

$$ext(\Delta') = \{F_1, \ldots, F_k\}$$

iff

$$\forall_{E \in ext(\Delta)} \exists_{E \in ext(\Delta')} E \subseteq F$$

We also define the equality relation $ext(\Delta) = ext(\Delta')$ iff

$$ext(\Delta) \preceq ext(\Delta') \text{ and } ext(\Delta') \preceq ext(\Delta)$$

Lemma 4.3. *Given default theories $\Delta = \langle D, W\rangle$ and $\Delta' = \langle D \cup \delta, W\rangle$, where $\Delta \mid\!\sim \delta$ using any one basic transformation, the theory Δ' gives the same extensions as Δ, i.e. $ext(\Delta') = ext(\Delta)$.*

Proof. Assume that for a default theory $\Delta = \langle D, W\rangle$ transformation $t = \langle D_t, f_t\rangle \mid\!\sim \delta$ was applied. This entails that $D_t \subseteq D$ and $W \models f_t$ hold. Let $\Delta' = \langle D \cup \delta, W\rangle$.

For the theories Δ and Δ' we have finite sets of extensions $ext(\Delta) = \{E_1, \ldots, E_k\}$ and $ext(\Delta') = \{F_1, \ldots, F_j\}$. For each extension E we have the set of generating rules $GD(E, \Delta)$. We should consider two cases.

1. There is no extension $E \in ext(\Delta)$ for which $D_t \in GD(E, \Delta)$
2. There is such an extension E.

For the case (1) it is obvious that $ext(\Delta) = ext(\Delta')$, as by Lemma 4.1 the transformation result δ is not in any set of generating defaults of Δ'.

Consider case (2). Let F be an extension of Δ'. If $\delta \notin GD(F, \Delta')$, then there exists an extension E of Δ such that $GD(E, \Delta) = GD(F, \Delta')$, and since both default theories have the same set of facts W, by Equation 3.1 we have $E = F$.

If, on the other hand, $\delta \in GD(F, \Delta')$, then by Lemma 4.1 also $D_t \in GD(F, \Delta')$ and there exists an extension E of Δ such that $GD(E, \Delta) \cup \{\delta\} = D_F$. By Lemma 4.2 and Theorem 2.5 (Reiter) we conclude that $E = F$.

This ends the proof of $ext(\Delta) = ext(\Delta')$ having proved $E = F$ for all cases. □

Let us define the sequence of default transformations, denoted by $\mid\!\sim_*$, as follows. The sequence of default transformations $\langle D, W\rangle \mid\!\sim_* \delta$ occurs, when the transformation $\langle D_n, W\rangle \mid\!\sim \delta$ can be applied, where $D_0, \ldots, D_n$ is a sequence such that $D_0 = D$ and $D_i = D_{i-1} \cup \{\delta_i\}$ where δ_i is obtained by applying a basic transformation on $\langle D_{i-1}, W\rangle$.

Theorem 4.1. *Given default theories $\Delta = \langle D, W\rangle$ and $\Delta' = \langle D', W\rangle$, where $D \subseteq D'$ and $\forall_{\delta \in D' \setminus D} D \mid\!\sim_* \delta$, we have $ext(\Delta) = ext(\Delta')$*

Proof. We prove the theorem inductively. For $D' \setminus D = \{d_1, \ldots, d_n\}$ let

$$D_0 = D$$
$$D_i = D_{i-1} \cup \{d_i\} \mid_{i=1 \ldots n}$$

This makes $D_n = D'$.

1. We have $ext(\langle D_0, W \rangle = ext(\Delta)$ as $D_0 = D$.

2. Assume $ext(\langle D_{i-1}, W \rangle) = ext(\Delta)$.

3. By Lemma 4.3 $ext(\langle D_i, W \rangle) = ext(\langle D_{i-1}, W \rangle)$ since $D_i = D_{i-1} \cup \{\delta\}$, where $\langle D_{i-1}, W \rangle \mathrel{|\!\!\sim} \delta$.

This proves that $ext(\langle D_i, W \rangle) = ext(\Delta)$ for any i. Since $D' = D_n$, we have $ext(\Delta') = ext(\Delta)$. □

4.3 Reasoning with default transformations

In a multi-agent system the peers exchange knowledge by means of querying each other and utilising the answers in order to reach conclusions. Following the inference procedure for Distributed Description Logic proposed in [68], the query, which is passed between ontologies is the subsumption query in the form $A \sqsubseteq B$, which in FOL is denoted as $A(x) \rightarrow B(x)$. Here, we will concentrate on this type of query and we will denote it by writing $A \sqsubseteq B$? to distinguish it from a DL statement.

For a query $A \sqsubseteq B$? to a default theory $\Delta = \langle D, W \rangle$ we will presume there are three possible answers:

- TRUE if $W \models A \sqsubseteq B$,

- FALSE if $W \not\models A \sqsubseteq B$,

- TRUE BY DEFAULT if the default $\frac{A \,:\, B \sqcap J}{B}$ can be generated using the default transformations, where A and B are the concepts from the query and J is the justification.

The first two answers are strict and do not require further processing. The last answer can be treated as a partial result and the final answer can be inferred when the justifications are checked.

Throughout the algorithm there are references to a DL reasoning procedure in the form $W \models A \sqsubseteq B$. These steps can be treated as calls to an inference procedure for Description Logics such as the tableau reasoning algorithm [10].

Algorithm 4.1: query

```
Input: Theory $\Delta = \langle D, W \rangle$
Input: Query $A \sqsubseteq B$?
begin
1   if $W \models A \sqsubseteq B$ then
        return TRUE;
2   $\mathcal{E} \leftarrow$ findExtensions($\langle D, W \rangle$);
    result $\leftarrow \emptyset$;
3   foreach $E \in \mathcal{E}$ do
        if $A \sqsubseteq B$ is consistent with $E$ then
            $\bar{D} \leftarrow$ findDefaults($GD(\Delta, E)$, $W$, $A \sqsubseteq B$?);
            result $\leftarrow$ result $\cup\ \bar{D}$;
4   if result $= \emptyset$ then
        return FALSE;
    result' $\leftarrow \emptyset$;
5   foreach $\delta \in$ result do
        $\delta' \leftarrow$ reduceJustification($\delta$, $W$);
        result' $\leftarrow$ result' $\cup\ \{\delta'\}$;
    return result';
```

Algorithm 4.1 shows the main idea of answering a query such as proposed above. Line 1 checks for a trivial answer based on the factual knowledge. If such an answer cannot be given, the next step is to find all extensions of the default theory (Line 2). This is done using an algorithm such as described in [12]. Iterating over all extensions (Line 3), the procedure gathers defaults in the form $\frac{A\,:\,B \sqcap J}{B}$, possibly from different extensions. This is done by transforming the generating defaults of each extension. If no such defaults are found, a negative answer is returned (Line 4). Finally, the resulting defaults are processed, applying the *reduce justifications* transformation (Line 5).

Algorithm 4.2: findDefaults

```
  Input: Defaults $\hat{D}$
  Input: Facts $W$
  Input: Query $A \sqsubseteq B$?
  begin
1     $D_0 \leftarrow \{\delta \in \hat{D} \mid W \models A \sqsubseteq Pre(\delta)\}$;
      result $\leftarrow \emptyset$;
      foreach $\delta \in D_0$ do
2         if $W \models Con(\delta) \sqsubseteq B$ then
              result $\leftarrow$ result $\cup \{\delta\}$;
          else
3             $\bar{D}$ = findDefaults($\hat{D} \setminus \delta$, $W \cup \{Con(\delta)\}$, $Con(\delta) \sqsubseteq B$?);
              foreach $\bar{\delta} \in \bar{D}$ do
4                 $\delta$' $\leftarrow$ mergeDefaults($\delta$, $\bar{\delta}$);
                  result $\leftarrow$ result $\cup \{\delta'\}$;
      return result;
```

The procedure of finding defaults that can be treated as intermediate answers to the given query is expressed in Algorithm 4.2. This procedure applies default transformations (a), (b) and (c) from Definition 4.2. Line 1 selects the defaults that are qualified for applying the *prerequisite substitution* transformation. Then, each of the selected defaults is checked whether it can be returned as the default answer to the given query (Line 2). If this is not the case, the algorithm is executed recursively (Line 3) to find a sequence of defaults that having applied additionally the *default transitivity* transformation will produce an appropriate default form. Line 4 merges the sequenced defaults to generate the final result using the formula presented in Algorithm 4.3.

Algorithm 4.3: mergeDefaults

```
  Input: Defaults $\delta_1, \delta_2$
  Output: Merged default
  begin
      return $\frac{Pre(\delta_1)\ :\ Jus(\delta_1) \wedge Jus(\delta_2)}{Con(\delta_2)}$;
```

Algorithm 4.4 shows the application of the *justification reduction* default transformation. The default's justifications are confronted with the known facts from

Algorithm 4.4: reduceJustification

```
  Input: Default $\delta$
  Input: Facts $W$
  Output: Default with reduced justification
  begin
      Assume $Jus(\delta) = \beta_1 \wedge \ldots \wedge \beta_n$;
1     $J \leftarrow \{\beta_i \mid W \not\models \beta_i\}$;
      $\beta \leftarrow \bigwedge_{\beta \in J} \beta$
      return $\frac{Pre(\delta)\ :\ \beta}{Con(\delta)}$;
```

the knowledge base and if any of them proves to be true in W, then it is removed from the default (Line 1).

In effect, the query algorithm generates one of three possible answers, which can be True, False or a set of defaults which are in the form $\frac{A\ :\ B \sqcap J}{B}$.

Chapter 5

DDL$\mathfrak{D}$-based Multi-Agent System

Agents are autonomous entities designed to pursue their goals. In a multi-agent system one of the actions used by agents is exchanging information between each other. One of the most basic communication schemes is the query-answer scenario. One agent needs to obtain certain information that another agent posesses and sends a query to that agent. The second agent responds with an answer to the query.

Agents are more than only servers of information. An agent may choose to act differently depending on its state or the credentials of the querying agent. It can selectively show information instead of simply answering queries to its knowledge base. For instance, an agent connected with a university will not publicly share personal information about the students, however it may answer queries whether a specific person is a student of the university. On the other hand, one could prevent the agent from giving out a full list of students. Such fine-grained control over what information is shared would not be possible by directly sharing a complete knowledge base.

Agents can act on behalf of a user by answering the user's queries or searching for information. Such tasks may require information from external sources such as online ontologies and other agents. In order to be able to send queries to other agents, it has to be able to find appropriate peers and formulate queries in a language that the remote peer will understand.

The environment of the Semantic Web, in which we place our agents is a large and dynamic network of interlinked information. In such conditions it is necessary to accept the open world assumption when discovering knowledge in the Web as it is impossible to gather absolutely complete information. However, it can often be beneficial to make assumptions in the light of incomplete information. Default logic is a formalism that makes it possible to reason with assumptions in an environment which is known to contain incomplete information.

In the previous chapter we defined a DL$\mathfrak{D}$ knowledge base, which is constructed using default logic embedded into description logic. Agents, which use DL$\mathfrak{D}$ knowledge bases utilize defaults locally to resolve the issues with incomplete information. However, as argued in the previous chapter, in a multi-agent system it is advantageous to communicate the assumptions made during reasoning between the agents. This chapter will describe a model of a multi-agent system, which makes use of the notion of returning defaults as reasoning inferences.

5.1 Motivating examples

To illustrate the need to answer queries with defaults, let us consider the following examples. The first example shows the usefulness of exchanging defaults between agents.

Example 5.1. The knowledge bases of two universities A_{univ1} and A_{univ2} contain information about their students. For the purpose of each university a default rule is kept, which enforces the closed world assumption stating that it is assumed that x is not a student if it does not exist in the local database. Consider the following contents of the knowledge bases:

Agent A_{univ1}	**Agent** A_{univ2}
$\frac{:\neg\mathsf{Student}}{\neg\mathsf{Student}}$	$\frac{:\neg\mathsf{Student}}{\neg\mathsf{Student}}$
$\mathsf{Student(JOHN)}$	$\mathsf{Student(MARY)}$

A third agent A_0 sends the query $\mathsf{Student(JOHN)}$? to both university agents. Answering by both agents with a definite statement leads to inconsistency, since the answers $\mathsf{Student(JOHN)}$ and $\neg\mathsf{Student(JOHN)}$ are contradictory.

$$\begin{tikzcd}
A_{univ1} \arrow[dr, bend left, "\mathsf{Student(JOHN)}"] & & A_{univ2} \arrow[dl, bend left, "\neg\mathsf{Student(JOHN)}"] \\
& A_0 \arrow[ul, bend left, "\mathsf{Student(JOHN)?}"] \arrow[ur, bend left, "\mathsf{Student(JOHN)?}"] &
\end{tikzcd}$$

On the other hand, if agent A_{univ2} answers with a default $\frac{:\neg\mathsf{Student(JOHN)}}{\neg\mathsf{Student(JOHN)}}$, then agent A_0 can add both answers to its knowledge base and reach the expected conclusion that John is a student.

$$\begin{tikzcd}
A_{univ1} \arrow[dr, bend left, "\mathsf{Student(JOHN)}"] & & A_{univ2} \arrow[dl, bend left, "\frac{:\neg\mathsf{Student(JOHN)}}{\neg\mathsf{Student(JOHN)}}"] \\
& A_0 \arrow[ul, bend left, "\mathsf{Student(JOHN)?}"] \arrow[ur, bend left, "\mathsf{Student(JOHN)?}"] &
\end{tikzcd}$$

The second example, apart from showing the advantages of returning defaults as answers to queries, shows the use of default transformations in the process of distributed reasoning.

Example 5.2. Consider three agents with the following knowledge bases.

Agent 1	**Agent 2**	**Agent 3**
Penguin(PAT)	$\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ $\frac{\mathsf{Bird}:\mathsf{Flies}}{\mathsf{Flies}}$	$\mathsf{Penguin} \sqsubseteq \neg\mathsf{Flies}$

In this example Agent 1 aims to discover whether Pat the penguin is able to fly. The answer to the question Flies(PAT) cannot be given using the first agent's knowledge base. However, the two other agents posess information that may help. If Agent 1 asked Agent 3, it would receive a straightforward answer that penguins cannot fly and thus Pat cannot fly.

Agent 2, however, possesses information that *penguins are birds* and *typically birds can fly*. Using this knowledge base and default reasoning, it should conclude that *penguins can fly* as the default's justification is consistent with the local knowledge base. Using the answer from Agent 2, Agent 1 would have the answer that Pat can fly, which is contrary to the answer from Agent 3.

Using default transformations, Agent 2 is able to return an answer in the form of the default $\frac{\mathsf{Penguin}\ :\ \mathsf{Flies}}{\mathsf{Flies}}$. This answer is not inconsistent with the third agent's answer and if Agent 1 receives answers from both Agent 2 and Agent 3, it will properly conclude that Pat cannot fly.

5.2 Model of the system

The formalism of default transformations introduced in the previous chapter can be used to perform distributed reasoning with multiple knowledge bases, which include defaults. In this section we will present a model for knowledge representation and exchange in a specific multi-agent system with shared global knowledge.

Although the reasoning algorithms are independent of how the MAS is defined, we propose a model in which, apart from local knowledge, agents have access to a commonly accepted global knowledge. The architecture of our model is motivated by the structure of the Web, where agents can utilize publicly available knowledge, which is published in the form of ontologies.

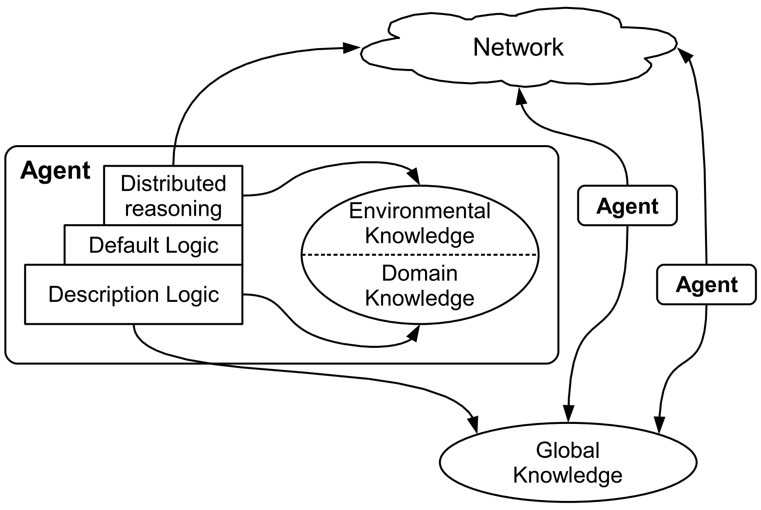

Figure 5.1: Model of the multi-agent system

Figure 5.1 shows an overview of the system. Each agent is an independent entity in the system and contains its own knowledge base and inference engine. Note that the agents may also perform other tasks, however, this work concentrates on the exchange of information within the multi-agent system.

Each agent has exclusive direct access to its *local knowledge* (LK). The agent can update this knowledge base either with information acquired from other agents or from other sources. Other agents do not have direct access to this data. The only way of obtaining the information is by querying its owner. The agent's local knowledge base is divided into two parts: *domain knowledge* (DK) and *environmental knowledge* (EK). The first contains information connected with the agent's primary activity, while the second includes information about other agents and information sources. Given the i-th agent in the MAS, the agent's knowledge is expressed as $LK_i = \langle DK_i, EK_i \rangle$. Apart from having local knowledge bases, all agents in the multi-agent system share *global knowledge* (GK), which sets the framework for the agent communication language.

5.2.1 Global knowledge

In the presented approach, global knowledge is shared among all agents in the multi-agent system. In this model, the role of global knowledge is to introduce a common "upper ontology", which is then extended by each agent to form its local domain knowledge. Apart from that, the global knowledge creates a common vocabulary for exchanging information between agents. In example 5.2 the concepts Bird and Flies ought to be defined in an external ontology, which is understood by all the agents.

5.2.2 Domain knowledge

The agent's domain knowledge is its internal representation of the information needed for local reasoning. In our approach, domain knowledge does not depend on the environmental knowledge and can be used for local reasoning.

The local domain knowledge (DK) and the global ontologies (GK) are expressed in terms of description logic with defaults. The local reasoning processes take into account both of these knowledge bases ($DK \cup GK$). This makes it possible to formulate queries to other agents in terms of the common vocabulary. The component responsible for distributed reasoning utilizes the environmental knowledge in order to find information required for the current reasoning task by communicating with other agents.

It is assumed that an agent's local domain knowledge is consistent with the global knoweledge. Otherwise, the overall knowledge base of the agent would be inconsistent and would not be useful for reasoning. Moreover, it is assumed that all concepts, roles and individuals are globally uniquely identified. This means that no two concepts with the same name can exist, which have different semantical meanings associated with them. Globally unique identifiers make communicated information unambiguous.

As an implementation remark, note that URI identifiers can be used to achieve globally uniquely identified terms. Such identifiers are used, among others, in the RDF framework.

5.2.3 Communication language

The agents in a multi-agent system exchange information using a communication language and a protocol for specifying types of messages and the available replies. For the purpose of distributed reasoning, we will use a communication protocol, where agent X sends a query agent Y asking if the other peer knows anything about the concept A being subsumed by concept B. We denote this query as $A \sqsubseteq B$?. Although in description logic the axiom $A \sqsubseteq B$ is equivalent to $\neg B \sqsubseteq \neg A$, the query $\neg B \sqsubseteq \neg A$? is considered as a different query due to the form of possible replies.

Agent Y replies to a query $A \sqsubseteq B$? with one of the following statements:

- True – Agent Y entails $A \sqsubseteq B$ and Agent X can add $A \sqsubseteq B$ to its knowledge base,

- False – Agent Y does not entail $A \sqsubseteq B$ and Agent X does not gain any new knowledge,

- set of defaults d_i in the form $\frac{A : J_i \sqcap B}{B}$, where J_i are the defaults' justifications – Agent X can add these to its knowledge base.

The communication language assumes that the concepts appearing in the query are atomic description logic concepts. Moreover, it has to be ensured that the communicating agents both understand the concepts being transmitted.

Let us define the vocabulary of a knowledge base.

Definition 5.1. *The* **vocabulary** V *of a knowledge base* KB *is the set of all concept names occurring in* KB *and is denoted as* $V(KB) \subseteq \mathbb{T}$*, where* $\mathbb{T}$ *is the set of all possible concept names.*

The vocabulary of the i-th agent in the multi-agent system will be denoted as:

$$V_i = V(DK_i \cup GK) = V(DK_i) \cup V(GK)$$

where $V(DK_i)$ is the vocabulary of the i-th agent's local domain knowledge and $V(GK)$ is the vocabulary of the global knowledge.

The *common vocabulary* of two agents i and j is the intersection of the agents' vocabularies $V_i \cap V_j$. Note that the common vocabulary will always contain the vocabulary of the global knowledge $V(GK) \subseteq V_i \cap V_j$ and may also contain additional common concept names, which both agents understand.

5.2.4 Environmental knowledge

For an agent to know, with which agents it can communicate about which topics, it needs to possess environmental knowledge about peers it may connect to. Two agents can exchange messages only if both of them share a common vocabulary. An agent may send the query $\mathsf{Penguin} \sqsubseteq \mathsf{Flies}$? only to other agents that understand the concepts $\mathsf{Penguin}$ and Flies.

In order to manage the information about its peers, each agent maintains environmental knowledge, which provides information about how to interact with the environment and other agents. The distributed reasoning procedure utilizes both types of knowledge to execute inference tasks in a distributed environment.

The environmental knowledge consists of two parts. Firstly, it incudes the vocabulary of the agent in the form of a set of concept and role names, about which the agent can be queried. Secondly, the environmental knowledge contains information about other agents and their vocabularies.

Environmental knowledge can be expressed as the relation between the set of agents and the set of concepts $EK \subseteq MAS \times \mathbb{T}$, where MAS is the set of agents in the multi-agent system and $\mathbb{T}$ is the set of all possible concept names. In practice, when searching for those agents that can answer a certain query, a data structure connecting concept names to agents is needed. Let us define the function *agents*, which returns a set of agents, which understand the concept name C as follows:

$$agents(C) = \{A \in MAS \mid (A, C) \in EK\}$$

In this work we will assume that the environmental knowledge is given a priori. However, it is a valuable topic to investigate how to acquire such information. The work by Ryżko [62] describes a multi-agent system using explanation based learning to acquire environmental knowledge through learning.

5.3 Distributed reasoning

The distributed reasoning component of the agent deals with identifying queries to be issued to remote information sources, choosing agents to communicate with and then sending appropriate queries.

An agent has the aim to identify such queries, for which the answers could change the inferences. It is unnecessary to ask questions, which have no influence on the outcome of the current inference task. The agent uses the tableau method to perform inferences on the local knowledge base (see Chapter 3 for a detailed description). In short, this algorithm tries to create a model of the description logic knowledge base by building a set of ABoxes. A disjunction in the KB creates two branches, which have to be expanded and checked for consistency. The algorithm ends when one of the branches is complete and does not contain an obvious contradiction (a clash) or all branches are closed containing clashes. The outcome of the tableau algorithm can be changed if adding a new piece of information causes an open branch to close.

In consequence, the goal of issuing queries to other agents is to close branches, which would not be closed only basing on the local knowledge base. Suppose the ABox

$\mathcal{A}$ contains two assertions $A(a)$ and $B(a)$. In the local KB they do not produce a clash. However, if another source can provide information that $(\neg B \sqcup \neg A)(a)$, the branch would be closed and a different conclusion could be reached. The assertion $(\neg B \sqcup \neg A)(a)$ will be added to the KB if either the subsumption relation $B \sqsubseteq \neg A$ or $A \sqsubseteq \neg B$ is asserted. This leads to the conclusion that when the assertions $A(a)$ and $B(a)$ are encountered, the agent should send the queries $A \sqsubseteq \neg B$? and $B \sqsubseteq \neg A$? to other agents and if it receives a positive or a default answer, the results of reasoning may change.

In Example 5.2, the first agent's tableau will contain the ABox $\mathcal{A} = \{\mathsf{Penguin}(\mathsf{PAT}), \neg\mathsf{Flies}(\mathsf{PAT})\}$. The first assertion is taken from the knowledge base and the second is the negation of the query. Now, the two queries that can be sent to other agents are $\mathsf{Penguin} \sqsubseteq \mathsf{Flies}$ and $\neg\mathsf{Flies} \sqsubseteq \neg\mathsf{Penguin}$. Any of these queries answered positively would directly cause the answer to the query $\mathsf{Flies}(\mathsf{PAT})$ to become positive. However, neither Agent 2 nor Agent 3 will answer positively. Agent 2 will nonetheless provide the answer $\frac{\mathsf{Penguin} : \mathsf{Flies}}{\mathsf{Flies}}$. The answer in the form of a default causes the asking agent to assimilate the default to its knowledge base. The agent must recompute the extensions, since adding a default can result in changing the number of extensions. Agent 1 can add this default to its knowledge base and compute a single extension $\{\mathsf{Penguin}(\mathsf{PAT}), \mathsf{Flies}(\mathsf{PAT})\}$, which can answer the query $\mathsf{Flies}(\mathsf{PAT})$? positively.

The third agent's knowledge can, however, change this outcome since it knows that penguins cannot fly. The default reasoning procedure, before deciding to apply the default (which came from Agent 2), will check its justification. Since this is done using the same tableau algorithm, the ABox $\mathcal{A} = \{\mathsf{Penguin}(\mathsf{PAT}), \mathsf{Flies}(\mathsf{PAT})\}$ will be tested for consistency. The first assertion of $\mathcal{A}$ comes from the knowledge base and the second one is the tested justification. This ABox will produce the queries $\mathsf{Penguin} \sqsubseteq \neg\mathsf{Flies}$ and $\mathsf{Flies} \sqsubseteq \neg\mathsf{Penguin}$. The first one can be answered by Agent 3, causing the tableau to close with a clash and disallowing the default $\frac{\mathsf{Penguin} : \mathsf{Flies}}{\mathsf{Flies}}$ to be applied for PAT. The final result is the answer FALSE to the query $\mathsf{Flies}(\mathsf{PAT})$.

The exchange of knowledge between agents is realized by extending the tableau reasoning procedure by adding a new tableau rule. The rule in Figure 5.2 says that if no other rules can be applied, pairs of concepts (A, B) are chosen to be formed into a query. The query is formed by creating two subsumptions $A \sqsubseteq \neg B$ and $B \sqsubseteq \neg A$. If a query succeeds, it results in making the tableau branch inconsistent.

If no other rules can be applied and $\mathcal{A}$ contains A and B, and the query (A, B) has not been sent yet,
then prepareQuery(A, B).

Figure 5.2: Tableau rule for issuing queries

For each pair of assertions $A(a)$, $B(a)$, where A and B are atomic concepts and a is an individual, the procedure prepareQuery(A, B) is run. If either of the atomic concepts subsumes the other in the local knowledge base (i.e. $A \sqsubseteq B$ or $B \sqsubseteq A$), the queries are not sent, because a positive answer would lead to inconsistency. The subsumption test should be possible to be made very efficient by indexing the concept lattice [48], since only atomic concepts may appear in queries. The procedure sendQuery(a, q) asynchronously sends the query q to the agent a. Thus, the procedure prepareQuery returns immediately and does not wait to receive answers from queried agents.

Algorithm 5.1: prepareQuery

```
Input: Concepts $A$, $B$
begin
    if $A \sqsubseteq B$ or $B \sqsubseteq A$ is entailed by the local knowledge base then
        return;
    targetAgents = $agents(A) \cap agents(B)$;
    foreach $a \in$ targetAgents do
        sendQuery($a$, $A \sqsubseteq \neg B$);
        sendQuery($a$, $B \sqsubseteq \neg A$);
```

The answers from other agents are recieved asyncronously. After all agents respond or a timeout is reached, if there are any answers, which are not FALSE, the local query has to be recomputed. In the process, new queries, which were not issued before, can be sent. The process continues until no answers to queries are received. The process is gurarantеed to finish, since no query is issued twice and the set of agents and their vocabularies are finite.

Algorithm 5.2: answerReceived

```
Input: answer $a$
begin
    if $a$ is a default $\frac{A : J \sqcap B}{B}$ then
        Add $\frac{A : J \sqcap B}{B}$ to the current knowledge base;
        Recompute extensions;
    else if $a$ is a positive answer $A \sqsubseteq B$ then
        Add $\neg A \sqcup B$ to each $\mathcal{A}_i$ in the tableau;
```

The answerReceived function is a callback function called asynchronously after each answer from a remote agent is received. Note that the recomputation of extensions can be postponed until more answers are received in order to reduce the number of times the extensions are determined. When the agent is no longer waiting for answers from other agents, the answer to the query is determined by the local default reasoning procedure.

5.4 Handling inconsistency

Although it is assumed that each agent's own knowledge base is consistent, it may occur that if the distributed knowledge bases were combined, they would be found inconsistent. In a message passing setting, an agent can receive a message containing information, which is inconsistent with its current knowledge. This situation has to be handled, otherwise, no meaningful inferences could be reached.

In our approach an answer from one agent cannot cause inconsistency, because the tableau rule shown in Figure 5.2 does not generate queries, for which positive answers would make the knowledge base inconsistent. However, since multiple queries are sent asynchronously to other agents, it may occur that two agents answer a different query positively and these two answers cause inconsistency. In the simplest case, an agent can send the following two queries: $A \sqsubseteq B$? and $A \sqsubseteq \neg B$?. The positive answers to these queries are together inconsistent.

To solve this problem, one of these answers has to be ignored in favour of the other. The choice of which agent's answer to ignore can be made arbitrarily or the decision can be based on some other means of assesing the value of the answers. One solution, which is used in the work by Ryżko [62] would be to employ a learnt confidence measure, where the information from the agent, which is trusted less would be discarded.

There is also a situation, which does not cause inconsistency but can be considered suspicious. In particular, when an agent sends the query $A \sqsubseteq B$? to multiple agents and receives both TRUE ($A \sqsubseteq B$) and default ($\frac{A : J \sqcap B}{B}$) answers, the semantics of these answers are in a sense contrary. The first answer says that absolutely all As are also Bs, while the second says that usually As are Bs but there can exist exceptions. It is difficult to state, which of these answers should be treated with a higher priority. If both answers are added to the knowledge base, the certain rule will override the default and will not allow exceptions. However, it can be also presumed that the agent, which allows exceptions to a rule, is more knowledgeable than the agent, which does not allow exceptions. In order to prefer the default answer, to positive answer would have to be discarded. Again, this type of situation could be handled by a confidence or trust measure, which agents would assign to their peers.

This work does not tackle these problems and provides the most straightforward solutions of arbitrarily discarding inconsistent answers and allowing the certain answer to shadow the default answers. Nonetheless, this is an interesting topic for future research.

5.5 Knowledge assimilation and caching

An agent may have to issue many queries to remote agents in order to achieve its task. Thus, it is important to have a possibility of preventing unnecessarily repeating a query sent to another agent. During the reasoning process, the agent assimilates all answers to queries into its domain knowledge for the purpose of solving the current task. However, if different tasks would require the same information, it would improve performance to reuse the once acquired information instead of repeating a query.

This method of caching query results requires a data structure to store axioms, assertions and defaults obtained from remote agents. The cache data structure is composed of triples $\langle K, A, T\rangle$, where K is the piece of knowledge, which is stored, A is the source agent, and T is the time the information was last updated. The time is needed to invalidate old records. The triples have to be indexed by K to be efficiently searchable.

Once a query is generated in order to be sent to other agents, the cache is checked, and if an answer is found, the query does not have to be sent to those agents, whose answers are already contained in the cache. Such a solution can significantly increase the efficiency of reasoning by reducing the amount of queries exchanged between agents.

5.6 Agent network topology

Given the assumption that we are dealing with the environment of the Internet and the World Wide Web and the fact that agents cannot be controlled by a central entity, we do not have any impact on the topology of agent connections. Looking at the characteristics of the Web, we can only conjecture that the agents will form a topology of a small-world network with a small amount of hub nodes.

Regardless of how the agents actually connect to each other, the problem of cycles in the connection graph has to be addressed. The occurrance of cycles can lead to an exponential growth of the number of messages passed between the agents or in the worst case it can lead to a failure in termination. Previous works dealing with distributed reasoning [63, 17, 20] assumed that an external entity can stratify the knowledge base a priori, thus preventing cycles from occurring. In our case, cycle have to be detected dynamically. Adjiman' algorithm for distributed reasoning [2] deals with cycles by including the history of queries in the query messages. Here, we propose a method, which only requires the inclusion of an identifier with the query messages.

The propagation of queries in a network of agents can be seen as a breadth-first search in the connection graph. This imposes that the processing is stopped, when a node has already been visited. If the original query to the first agent is given a universally unique identifier (UUID), then an agent receiving a query with the same UUID can discard this query. However, cycles in the reasoning process are not harmful in themselves. The problem is when they cause infinite loops in terms of sending messages between agents. We can detect a situation leading to an infinite loop by discarding duplicate messages identified by the UUID of the task together with the query. Since the number of terms in an agent's vocabulary is finite and the queries contain only atomic concepts, the number of possible queries is finite. Assuming the number of agents is also finite, we can state that the number of messages passed during the inference process is finite. This leads to the conclusion that in any network topology the distributed reasoning terminates.

Chapter 6

Implementation and evaluation

Chapters 4 and 5 have introduced algorithms and methods for reasoning in distributed knowledge bases expressed in description logic with defaults. This chapter presents the implementation and evaluation of the key issues presented in this work. The implemented system experimentally verifies that the algorithms return expected results. Moreover, we have tested the scalability of such a multi-agent system with respect to a growing number of agents involved in the reasoning process of answering a query.

Section 6.1 presents the architecture and key components of the implemented distributed reasoning system. In Section 6.2 we present tests, which have been performed in order to verify the validity of the proposed algorithms and to test the scalability of such a system.

6.1 System architecture

The prototype implementation of the distributed reasoning system has been fully implemented in the Scala programming language in version 2.8.0 [56], taking advantage of the language features such as higher-order functions, functional data structures, closures, parser combinators and pattern matching. Moreover, the Scala language is compiled into Java Virtual Machine bytecode, which makes it compatible with existing Java libraries.

The multi-agent system has been implemented using actors in the scala standard library. This solution is adequate for testing purposes. Howeve, for a production system, an advanced multi-agent platform would have to be used. Our multi-agent system is designed in such a way, that moving to a different MAS platform would only require changing the interface to the underlying agent platform. JADE [15] is a good example of a MAS platform, which is compatible with our implementation.

The experimental system provides a set of homogeneous agents, which differ only by the knowledge they posess. However, as long as an agent understands and can reply to the same set of messages, the actual implementation of an agent can be exchanged, creating a heterogeneous multi-agent system.

Each agent contains a knowledge base expressed with description logic with defaults. Agents have environmental knowledge, which contains knowledge about the addresses of remote agents and the vocabularies they understand. This information is used to select agents to query. The agent's knowledge base has a layered structure, which is shown in Figure 6.1.

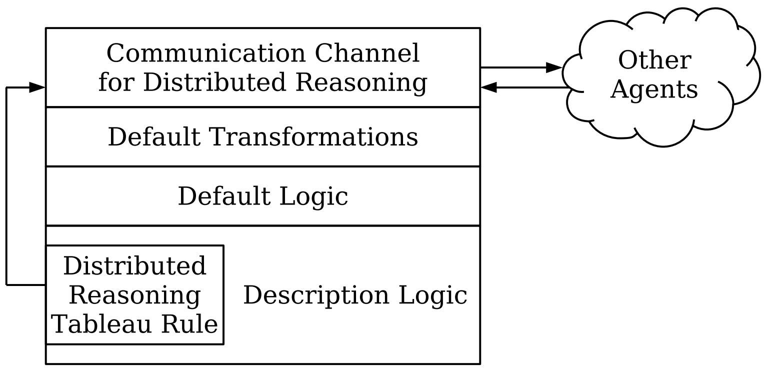
Communication Channel for Distributed Reasoning
Default Transformations
Default Logic
Distributed Reasoning Tableau Rule
Description Logic
Other Agents

Figure 6.1: Agent's knowledge layers

Each agent can answer three types of queries. Firstly, the DefaultQuery is the query, which is used in the distributed reasoning algorithm and it is in the form $A \sqsubseteq B?$. The answer is constructed as a set of statements. The set can be empty, contain a single statement $A \sqsubseteq B$ or one or more defaults $\dfrac{A : B \sqcap J}{B}$, where J is the default's justification.

The agent can also answer an entailment query, which can be answered credulously or skeptically (CredulousQuery, SkepticalQuery). This query is usually used to answer a user's query. All types of queries trigger the distributed reasoning procedure and cause the asked agent to send additional queries to other agents.

6.1.1 The Description Logic Reasoner

The description logic reasoner is the smallest building block of the reasoning system. It implements the tableau calculus for description logics [66] and currently supports the constructors of the $\mathcal{ALC}$ langugage, which are conjunction, disjunction, negation of arbitrary concepts and both quantifiers. The implementation heavily relies on the Scala language immutable data structures, which are used to represent the tableaux. By utilizing immutable lists and maps, it is easy to implement a backtracking algorithm, since all previous states of the tree are kept intact in memory. Moreover, our implementation allows to add subsequent tableau expansion rules. Hence, the reasoning engine can be expanded to work with more expressive description logic languages. This feature is also used to add distributed reasoning capabilities to the inference engine.

The DL reasoner is capable of answering subsumption, consistency and instance checking queries. All these query types are reduced to solving the consistency problem. For a real-world implementation, it would be suggested to use a highly-optimized reasoner, such as Pellet [70] or HermiT [53]. These are production-ready implementations of the tableau algorithm, which support the full OWL DL language, which has more constuctors than the $\mathcal{ALC}$ language.

In order to simplify the evaluation of the algorithms, the Manchester OWL syntax [35] has been used as the language for importing knowledge bases into the system. This syntax provides the following constructors:

Manchester OWL syntax	DL expression
`C and D`	$C \sqcap D$
`C or D`	$C \sqcup D$
`not C`	$\neg C$
`R some C`	$\exists R.C$
`R only C`	$\forall R.C$
`TOP`	$\top$
`BOTTOM`	$\bot$

where C and D are concept descriptions and R is a role name.

6.1.2 The Default Logic Reasoner

The default logic reasoner is built on top of the description logic reasoner. The algorithms described in Chapter 3 have been implemented to find sets of generating defaults, which identify the extensions of a default theory. Having computed the extensions, both skeptical and credulous queries can be answered.

The default logic reasoner is further extended by the algorithms proposed in Chapter 4 for default transformations.

Our implementation of the default logic reasoner can be adapted to use a different description logic reasoner, such as Pellet or HermiT. However, the distributed reasoning layer changes the functionality of the DL reasoner, such that it cannot be easily replaced.

The defaults are expressed using the following notation:

```
A : B / C
```

where `A`, `B` and `C` are concept expressions written using the Manchester OWL syntax. For instance, the default stating that typically a bird can fly unless it has a broken wing can be written as:

```
Bird :  Fies and not BrokenWing / Flies
```

6.1.3 Distributed Reasoning

The agents have been implemented using the *Scala actors* model. This programming model allows to create independent and asynchronous processing units, which communicate between each other by passing asynchronous messages.

In order to send queries to other agents in the process of reasoning, a new tableau expansion rule has been introduced to the description logic reasoner according to the description in Chapter 5. This new rule generates queries, which are sent to agents, which understand all vocabulary terms in the query. The asking agent caches answers from other agents to reduce the network activity. The answers for these queries are added to the agent's knowledge base and the reasoning engine can take into account new defaults or facts.

The distributed reasoning layer requires a modified tableau reasoner. In consequence, an existing description logic reasoner would require modifications to be used with the distributed reasoning component. The extension comprises of a new tableau expansion rule. As reasoning engines Pellet and HermiT both are based on tableau algorithms and they are both open source, we presume it would be feasible to extend them with the required functionality.

6.2 Test cases

In this section we describe the evaluation of the correctness and the scalability of the proposed algorithms in terms of different agent topologies. The experiments described in this section show several representative agent network topologies and address the problems referring to each of them. The implemented system verifies that the expected results are reached. Apart from these scenarios, the system is well covered with more than eighty unit tests for all implemented components.

Since we are dealing with a potentially large agent system connected with Internet resources, it is important not to introduce additional computational complexity associated with exchanging queries. In the first two example scenarios, the number of exchanged queries is proportional to the number of agents, which is an anticipated result. This result in turn makes it possible to scale the system by distributing agents onto different processors and thus gaining the possibility to run processes in parallel. The third example shows how the implemented system works with an ontology, which has the properties of a real-life ontology.

6.2.1 Chain of agents

The first scenario presents a chain of agents, in which a sequence of agents pass a query along the chain. The final knowledge item is placed at the end of the chain and is propagated back to the first agent, which initiated the first query. Figure 6.2 illustrates the connections between agents and the messages sent between them. A_i, $i \in \{0..n\}$, B and C are atomic concepts and z is an individual.

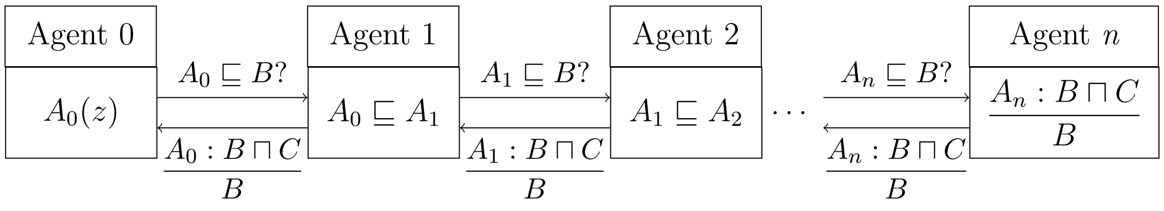

Figure 6.2: Chain of agents

The first agent (Agent 0) is issued the skeptical query $B(z)$. It is a question whether z is an instance of B. All agents must have B in their vocabulary so that they can receive queries containing the symbol B.

The query is propagated to subsequent agents until the last agent is reached, which can answer its query with a default in its knowledge base. This default is then transformed and sent as a reply by all agents back to the first agent, which receives the default $\frac{A_0 : B \sqcap C}{B}$. This default can be used to answer the original skeptical query. The concept C, which is propagated from the last agent in the default's justification is now checked whether the default can be applied. This is performed by issuing the query $A_0 \sqsubseteq \neg C$?. In this scenario, however, there is no agent, which could process such a query.

By running the scenario in the implemented system, we have verified that increasing the number of agents leads to linearly increasing number of queries exchanged between the agents. The time of computing a single query, which is propagated through a chain of agents, cannot be decreased by running the reasoning processes in parallel, as each agent has to wait for the answer from its neighbour. However, running many such queries can lead to an increased efficiency, as an agent waiting for response can use its processing power to run other inferences.

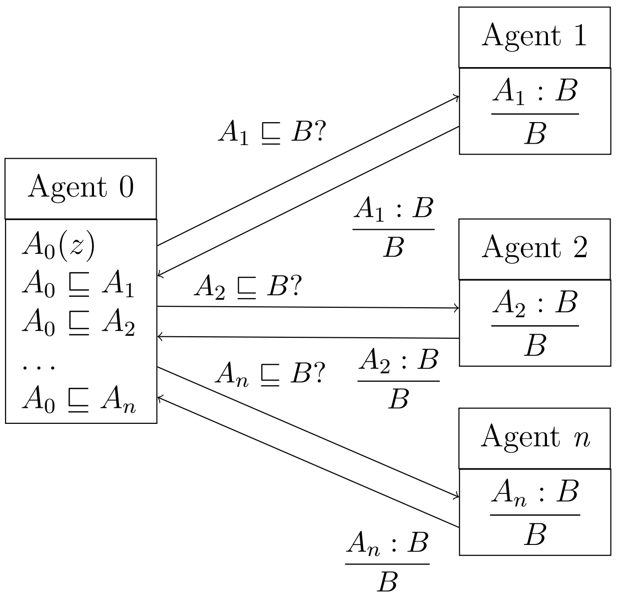

Figure 6.3: Star topology

6.2.2 Star topology

The star topology is described as a situation, where one agent directly sends queries to a number of other agents. Figure 6.3 shows the the queries exchanged between the agents. The queries in this test case are sent simultaneously and are processed by the agents concurrently. This case provides the best setting for increasing efficiency because all queries can be processed in parallel. Furthermore, in this experiment, only one answer is necessary for Agent 0 to produce a default extension containing $B(z)$. Nevertheless, the agent will still wait for other replies, since it always receives a default answer, which could later be invalidated by a negative fact. If, however, some agents do not reply in time, producing a timeout, the result will not change and the first agent will return a valid answer.

The number of queries exchanged in this scenario depends linearly on the number of agents. This means that additional agents, which are queried by one agent do not increase the overall complexity of the reasoning process. Moreover, in case all queries are processed independently, they can be run concurrently.

In most cases, the actual reasoning topology will be a combination of the chain and the star topologies, as both types of situations will occur. Firstly, there are situations, where queries are propagated if an agent does not contain all necessary information. Secondly, an agent may have to query multiple sources to collect all needed data.

6.2.3 Pizza ontology

In order to illustrate the distributed reasoning concept using an existing ontology, we have chosen the Pizza ontology[1], which is used by the Manchester University for tutorial purposes. The ontology contains concepts related to different types of pizzas with their bases and toppings. Since the implemented reasoner only operates on the $\mathcal{ALC}$ subset of the OWL-DL language, several axioms had to be ommited from the original ontology, which used the cardinality constuctors. Figure 6.4 presents an overview of the concepts in the pizza ontology.

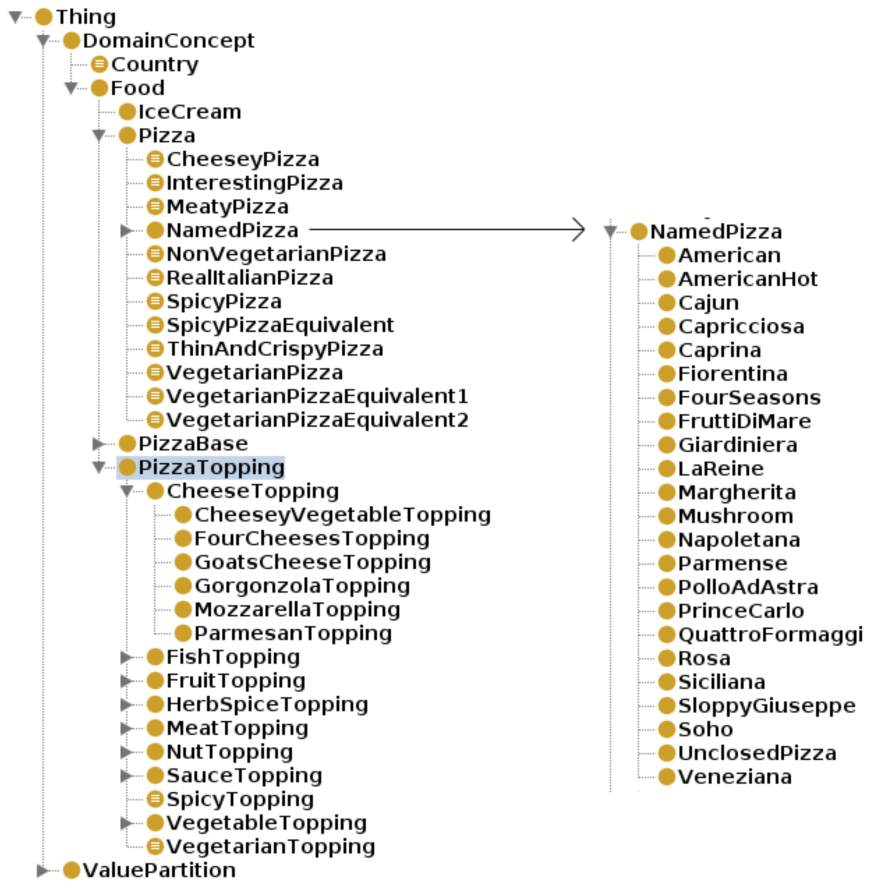

Figure 6.4: Pizza ontology overview

[1]`http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl`

Different pizza names are defined by the types of toppings they can have. For instance, a Margherita has two toppings: MozzarellaTopping and TomatoTopping. The pizza ontology does not include any defaults, however by browsing the concepts, one can find that the majority of pizzas have a cheese topping. There are six classes defined, which are subclasses of CheeseTopping. The only pizza that is defined not to include cheese is FruttiDiMare. The class of all pizzas, which have a cheese topping is named CheeseyPizza. This leads to the conclusion that the following default can be formulated:

$$\frac{\mathsf{Pizza} : \mathsf{CheeseyPizza}}{\mathsf{CheeseyPizza}}$$

In order to distribute the ontology among agents, we have chosen to extract the information about the twenty three named pizzas from the ontology and further divide the named pizzas into two parts. In result, we have defined the pizza ontology without named pizzas as the *global knowledge* available for every agent and two agents with disjoint sets of named pizzas, specifically Pizzas A−L and Pizzas M−Z. Moreover, the default knowledge is included within a separate agent – Pizza Defaults. Thus, the knowledge about pizzas is divided among three agents. Note that the problem of partitioning large knowledge bases is not covered by this work, as it is assumed that the algorithms work on existing knowledge partitions.

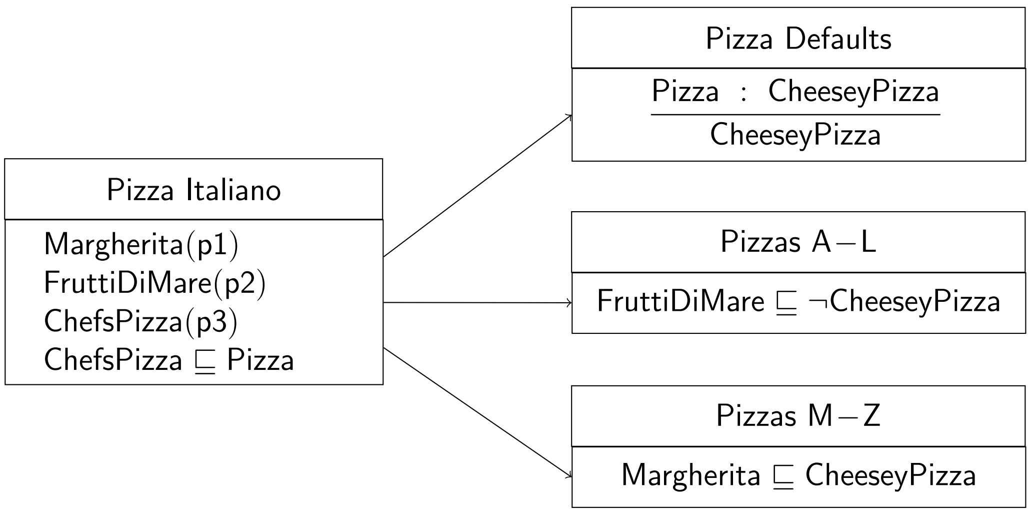

Figure 6.5: Pizza example

Let us define a new agent that will use the above agents' knowledge. Assume there is an agent called Pizza Italiano, which is connected with a restaurant and contains knowledge about the serverd meals, among which there are several types of pizza. In particular, two pizzas, Margherita and FruttiDiMare, exist in the pizza ontology, and one, ChefsPizza, is a newly defined concept in the restaurant's ontology. Figure 6.5 shows all defined agents with the parts of their knowledge bases, which take part in the presented scenario.

The Pizza Italiano agent is interested in obtaining additional information about the pizzas, which are served. In this example, the agent is asked the question about each of the offered pizzas, whether the pizza contains cheese. The restaurant agent does not contain this information itself and has to send queries to agents, which are specialized in describing pizzas.

The Pizza Defaults agent will always answer that *typically, a pizza contains cheese*, regardless of the actual type of pizza. The agents posessing information about named pizzas can in turn give answers according to their knowledge. The default knowledge allows to answer queries with default knowledge, when the exact knowledge is unavailable or does not exist. If the restaurant defines a new pizza type, it will be automatically inferred to be a CheeseyPizza if not stated otherwise.

The following conclusions can be made from this example:

1. The agent containing default knowledge can answer a query, which is correct in most cases.

2. The Pizza Defaults agent can provide default information about entities, which are not completely defined, such as ChefsPizza in our example.

3. Since the answer is in the form of a default, the asking agent can verify and invalidate this information by receiving answers from other sources.

4. There is no time overhead connected with including the Pizza Defaults agent in the reasoning process, since queries sent to all three agents are processed concurrently.

5. The overall pizza ontology can be expressed in fewer axioms, as the fact that most pizzas have a cheese topping can be expressed as a default, and only exceptions need to be explicitly named.

6. The queries for cheesey pizzas can be calculated more efficiently.

6.3 Summary

The proposed multi-agent system has been successfully implemented and tested. The results prove that such a system is feasible and can aid in modelling information in the Semantic Web. The implementation is based on a standard syntax for expressing description logic expressions. The knowledge base inference engine has a layered architecture, which makes it more robust and allows to substitute any of the layers with a different implementation. In particular, the DL reasoner, which has the greatest impact on performance, can potentially be replaced with one of the existing production-ready solutions. The implementation lacks some possible optimizations, which would increase the overall system effictiency. Nonetheless, these optimizations would not change the core methods utilized in the system but would make it more difficult to analyze and evaluate.

The experiments, which were performed show that complex reasoning tasks can be performed using multiple distributed agents equipped with their own reasoners. Moreover, agents can be connected in an arbitrary topology and queries can be propagated through the network of agents until peers with final answers are found.

The first two experiments executed in the prototype system demonstrate that the number of queries exchanged in the entire system grows linearly with the number of agents. This result shows the ability to scale such a system into the realities of the Internet, as real-world knowledge bases are undoubtedly very complex. The last scenario uses an existing ontology to model information distributed into several agents and containing defaults. This example shows the usefulness of using defaults to express information in a multi-agent system.

Chapter 7

Conclusions

The Semantic Web is a rapidly developing research area, which combines several aspects of the evolution of the Web. Among others, this field involves knowledge representation formalisms and agents, which use semantically rich information.

This dissertation contributes to the research on the methods of exchanging knowledge between agents in a multi-agent system. The work presents a method for representing and exchanging knowledge, which may be incomplete. The formalism of default logic is used in order to express typical situations and assumptions for drawing inferences. The work is based on description logic with defaults. In this formalism, defaults are embedded into the $\mathcal{ALC}$ description logic. A method of communicating assumptions between agents is introduced. Any additional information can be utilized to invalidate facts, for which the assumptions are proved to be incorrect.

In order to achieve this goal, the formalism of default transformations is introduced together with an important property of the transformations, which says that transformed defaults do not alter the knowledge base's inferences. This enables returning defaults as query results. Basing on the defined default transformations, a reasoning algorithm is introduced, which provides a possibility of answering queries either with definitive positive and negative answers, or transformed defaults.

Furthermore, a model of a multi-agent system for exchanging knowledge in the Semantic Web is presented. An agent in this model consists of domain and environmental knowledge. The agent can also utilize global knowledge, which is shared among agents. Moreover, a distributed reasoning algorithm is described, in which the agents exchange queries and answers to reach a specified goal. The algorithm extends the tableau method for description logic by introducing a rule that triggers sending queries to remote agents. This method makes use of the previously introduced default transformations.

Finally, the introduced algorithms have been implemented in a prototype system. The implementation is well covered with unit tests and several usage scenarios are described to assess the correctness and the scalability of the proposed solutions. The test cases show that description logic with defaults is an appropriate knowledge representation formalism for the Semantic Web and makes it easier to manage incomplete knowledge. Moreover, employing the proposed algorithms in a multi-agent system leads to the improvement of efficiency due to the possibility to distribute knowledge among remote peers.

As future work, we plan to extending the proposed formalisms to more expressive description logics than $\mathcal{ALC}$. Moreover, specifying an extension to the OWL language would enable to encode defaults into existing ontologies. By developing an extension to an ontology editor, such as Protégé, using defaults with ontologies could be made more widespread.

Furthermore, we intend to work on including a learning aspect to the communication of defaults between agents. Defaults, which come from remote agents, may conflict with other defaults or strict rules. The task of learning would be to modify the defaults and strict rules taking into account a confidence measure towards different information sources.

Abbreviations and Symbols

Δ	default theory
$ext(\Delta)$	set of extensions of the default theory Δ
$GD(E, \Delta)$	set of generating defaults for extension E of theory Δ
$Pre(d)$	prerequisite of default d
$Jus(d)$	justification of default d
$Con(d)$	consequence of default d
$PRE(D)$	set of prerequisites of defaults D
$JUS(D)$	set of justifications of defaults D
$CON(D)$	set of consequences of defaults D
$Th(E)$	deductive closure of E
ABox	assertional component of a DL knowledge base
TBox	terminological component of a DL knowledge base
$\top$	top concept
$\bot$	bottom concept

AI	artificial intelligence
$\mathcal{ALC}$	Attributive Concept Language with Complements
CWA	closed world assumption
DAI	distributed artificial intelligence
DDL	distributed description logic
D$\mathfrak{D}$L	distributed default logic
DDL$\mathfrak{D}$	distributed description logic with defaults
$\mathfrak{D}$L	default logic
DL	description logic
DL$\mathfrak{D}$	description logic with defaults
FOL	first-order logic
KB	knowledge base
KR	knowledge representation
MAS	multi-agent system
OWA	open world assumption
OWL	Web Ontology Language
SW	Semantic Web
WWW	World Wide Web
XML	Extensible Markup Language

References

[1] Agent communication language specifications, 2002. `<http://www.fipa.org/repository/aclspecs.html>`.

[2] Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *J. Artif. Intell. Res. (JAIR)*, 25:269–314, 2006.

[3] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artif. Intell.*, 162(1-2):49–88, 2005.

[4] Grigoris Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.

[5] Grigoris Antoniou. A nonmonotonic rule system using ontologies. In *RuleML*, 2002.

[6] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2. edition, 2008.

[7] Grigoris Antoniou and Gerd Wagner. Rules and defeasible reasoning on the semantic web. In *RuleML*, pages 111–120, 2003.

[8] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *IJCAI*, pages 446–451, 1991.

[9] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. In *Advances in Artificial Intelligence, 18th Annual German Conference on Artificial Intelligence*, pages 51–62, 1994.

[10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications.* Cambridge University Press, New York, NY, USA, 2007.

[11] Franz Baader and Philipp Hanschke. Extensions of concept languages for a mechanical engineering application. In Hans Jürgen Ohlbach, editor, *GWAI-92: Advances in Artificial Intelligence*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer Berlin / Heidelberg, 1993.

[12] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. Technical Report RR-93-20, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1993.

[13] Jie Bao, George Voutsadakis, Giora Slutzki, and Vasant Honavar. Package-based description logics. In *Modular Ontologies*, pages 349–371. Springer, 2009.

[14] Dave Beckett. RDF/XML syntax specification (revised), 2004. `<http://www.w3.org/TR/rdf-syntax-grammar/>`.

[15] F. Bellifemine, A. Poggi, and G. Rimassa. JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108, 1999.

[16] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[17] Alexander Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *J. Data Semantics*, 1:153–184, 2003.

[18] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *AAAI*, pages 34–37, 1984.

[19] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema, 2004. `<http://www.w3.org/TR/rdf-schema/>`.

[20] Pawel Cholewinski. Reasoning with stratified default theories. In *LPNMR '95: Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 273–286, London, UK, 1995. Springer-Verlag.

[21] Keith L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[22] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. DAML+OIL (march 2001) reference description, 2001. `<http://www.w3.org/TR/daml+oil-reference>`.

[23] Dieter Fensel, Ian Horrocks, Frank Van Harmelen, Deborah McGuinness, and Peter F. Patel-Schneider. OIL: Ontology infrastructure to enable the semantic web. *IEEE Intelligent Systems*, 16:200–1, 2001.

[24] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[25] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jörg Müller, Michael Wooldridge, and Nicholas Jennings, editors, *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 21–35. Springer Berlin / Heidelberg, 1997.

[26] M. R. Genesereth and R. E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[27] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularizing owl ontologies. In *Proceedings of the 4th International Semantic Web Conference (Poster Track)*, 2005.

[28] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[29] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and a role-forming predicate operator. *J. Log. Comput.*, 9(3):351–384, 1999.

[30] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.*, November 2004.

[31] J. Heflin and H. Muñoz-Avila. LCW-based agent planning for the semantic web. In *Ontologies and the Semantic Web Workshop at AAAI-02*, pages 63–70, November 2002.

[32] Martin Hepp and Andreas Radinger. eClassOWL – the web ontology for products and services, 2010. `<http://www.heppnetz.de/projects/eclassowl/>`.

[33] S. Heymans and Dirk Vermeir. A defeasible ontology language. In *On the Move to Meaningful Internet Systems*, pages 1033–1046, London, UK, 2002. Springer-Verlag.

[34] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 335–346, 1991.

[35] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester owl syntax. In *OWLED*, 2006.

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

[37] Ian Horrocks, Bijan Parsia, Peter Patel-Schneider, and James Hendler. Semantic web architecture: Stack or two towers? *Principles and Practice of Semantic Web Reasoning*, pages 37–41, 2005.

[38] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. In *Description Logics*, 1998.

[39] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[40] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[41] Ulrich Junker and Kurt Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *AAAI*, pages 278–283, 1990.

[42] Mariusz Kaleta, Piotr Palka, Eugeniusz Toczylowski, and Tomasz Traczyk. Electronic trading on electricity markets within a multi-agent framework. In Ngoc Thanh Nguyen, Ryszard Kowalczyk, and Shyi-Ming Chen, editors, *ICCCI*, volume 5796 of *LNCS*, pages 788–799, 2009.

[43] Aditya Kalyanpur. *Debugging and repair of OWL ontologies.* PhD thesis, University of Maryland, 2006.

[44] Yarden Katz and Jennifer Golbeck. Social network-based trust in prioritized default logic. In *Proceedings of the Twenty-First National Conference onArtificial Intelligence (AAAI-06*, 2006.

[45] Vladimir Kolovski, Bijan Parsia, and Yarden Katz. Implementing OWL defaults. In *OWL: experiences and directions workshop*, 2006.

[46] Michal Laclavik, Zoltan Balogh, Marian Babik, and Ladislav Hluchý. AgentOWL: Semantic knowledge model and agent architecture. *Computers and Artificial Intelligence*, 25(5), 2006.

[47] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[48] Jacek Lewandowski and Henryk Rybinski. A hybrid method of indexing multiple-inheritance hierarchies. In *Proceedings of the 18th International Symposium on Foundations of Intelligent Systems*, ISMIS '09, pages 211–220, Berlin, Heidelberg, 2009. Springer-Verlag.

[49] Frank Manola and Eric Miller. RDF primer, 2004. `<http://www.w3.org/TR/rdf-primer/>`.

[50] J McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39, 1980.

[51] Deborah L. McGuinness, Richard Fikes, Lynn Andrea Stein, and James A. Hendler. Daml-ont: An ontology language for the semantic web. In *Spinning the Semantic Web*, pages 65–93, 2003.

[52] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview, 2004. `<http://www.w3.org/TR/owl-features/>`.

[53] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[54] Ulf Nilsson and Jan Maluszynski. *Logic, Programming and Prolog.* Wiley, 1990.

[55] Donald Nute. Defeasible logic. In *INAP*, pages 87–114, 2001.

[56] Martin Odersky. *The Scala Language Specification, Version 2.8.* Programming Methods Laboratory, EPFL, Switzerland, 2010.

[57] David Poole. Default logic. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of logic in artificial intelligence and logic programming*, volume 3, pages 189–215. Oxford University Press, 1994.

[58] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for rdf, 2008. `<http://www.w3.org/TR/rdf-sparql-query/>`.

[59] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13:81–132, 1980.

[60] R. Reiter and G. Criscuolo. On interacting defaults. In *Readings in nonmonotonic reasoning*, pages 94–100, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[61] Vincent Risch and Camilla Schwind. Tableaux-based characterization and theorem proving for default logic. *J. Autom. Reasoning*, 13(2):223–242, 1994.

[62] Dominik Ryżko. *Default logics for reasoning and knowledge sharing in multi-agent systems*. PhD thesis, Politechnika Warszawska, 2007.

[63] Dominik Ryżko and Henryk Rybiński. Distributed default logic for multi-agent system. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 204–210, Washington, DC, USA, 2006.

[64] Dominik Ryżko, Henryk Rybiński, and Przemysław Więch. Learning mechanism for distributed default logic based mas - implementation considerations. In *Proceedings of the International IIS 08 Conference*, pages 329–338, 2008.

[65] Ulrike Sattler. A concept language extended with different kinds of transitive roles. In *Advances in Artificial Intelligence, 20th Annual German Conference on Artificial Intelligence*, pages 333–345, 1996.

[66] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[67] James G. Schmolze, Bolt Beranek, and Newman Inc. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[68] Luciano Serafini and Andrei Tamilin. Local tableaux for reasoning in distributed description logics. In *Description Logics*, 2004.

[69] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED 2007: Proceedings of the Third International Workshop on OWL: Experiences and Directions, Innsbruck, Austria*, 2007.

[70] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, 2007.

[71] Kent A. Spackman, Ph. D, Keith E. Campbell, Ph. D, Roger A. Côté, and D. Sc. (hon. SNOMED RT: A reference terminology for health care. In *J. of the American Medical Informatics Association*, pages 640–644, 1997.

[72] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383, 2000.

[73] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data Knowl. Eng.*, 25(1-2):161–197, 1998.

[74] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006*, pages 292–297. Springer, 2006.

[75] Przemysław Więch and Henryk Rybiński. A novel approach to default reasoning for mas. In *Proceedings of the Rough Sets and Current Trends in Computing - 7th International Conference, RSCTC 2010*, pages 484–493, 2010.

[76] William A. Woods. What's in a Link: Foundations for Semantic Networks. In D.G. Bobrow and A. Collins, editors, *Representation and Understanding.* Academic Press, 1975.

Appendix A

Software technical documentation

The appendix presents technical details of the implemented software, which is described in Chapter 6. Each layer of the system provides a set of Application Programmer's Interface (API) classes and methods. They can be used to embed the implemented components into other projects, in which description logic with defaults can be used for reasoning. The implementation is divided into four main components, which comprise the four layers depicted in Figure 6.1. The description logic reasoner provides fundamental DL inferences for subsequent layers. The default logic reasoner provides the ability to build the sets of generating defaults for a DL$\mathfrak{D}$ knowledge base and provides credulous and skeptical inferences. The default transformations layer implements the algorithms proposed in Chapter 4. Finally, the distributed reasoning layer consists of a multi-agent system and reasoning procedures described in Chapter 5.

A.1 Description logic reasoner

The description logic reasoner implements the tableau algorithm for the $\mathcal{ALC}$ variant of description logic. The package provides the functionality to parse a knowledge base and input queries to form an internal data representation and allows executing queries on the data.

At the core of this component lies the data representation of DL concept descriptions. Figure A.1 shows the classes, which are used to form a *parse tree* of

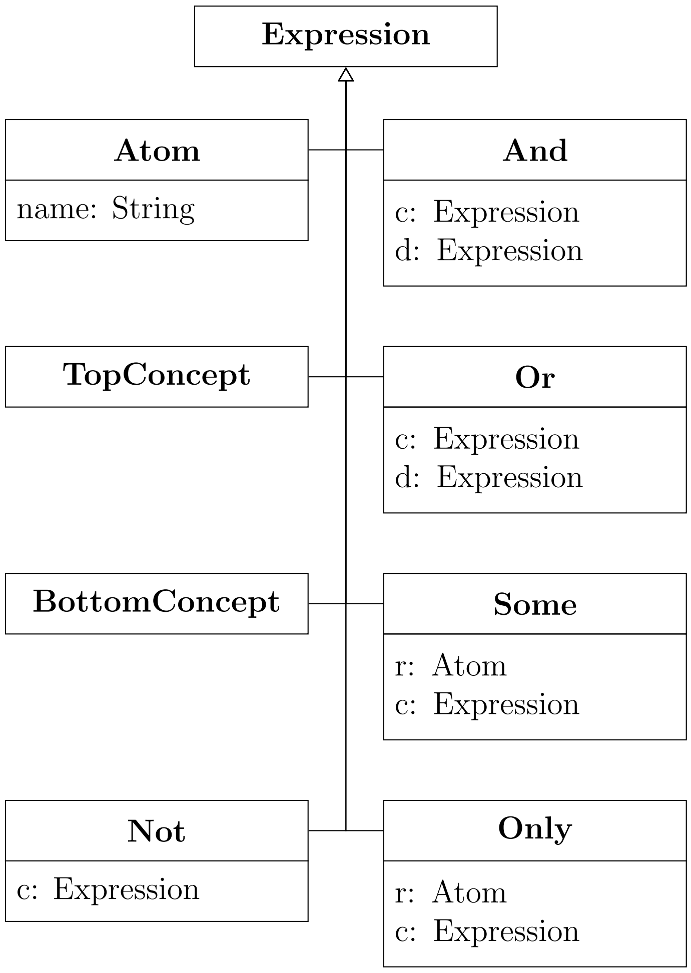

Figure A.1: Expression classes

a concept description. At the leafs of the tree are `Atom` objects representing concept names or `TopConcept` or `BottomConcept`, which represent the top and bottom concept.

The `Expression` and `Atom` classes is further used to form facts, which comprise the knowledge base. The `Isa` class represents the fact that an object a belongs to a concept c ($C(a)$). The `Relation` class represents a concrete relation r between two objects a and b ($R(a,b)$). The `Gci` is the general concept inclusion relation between two concept descriptions ($C \sqsubseteq D$). The `Equiv` and `Definition` represent the equivalence relation between two concept descriptions ($C \equiv D$), where the former allows any concept descriptions and the latter allows only an atomic concept name on the left hand side of the relation.

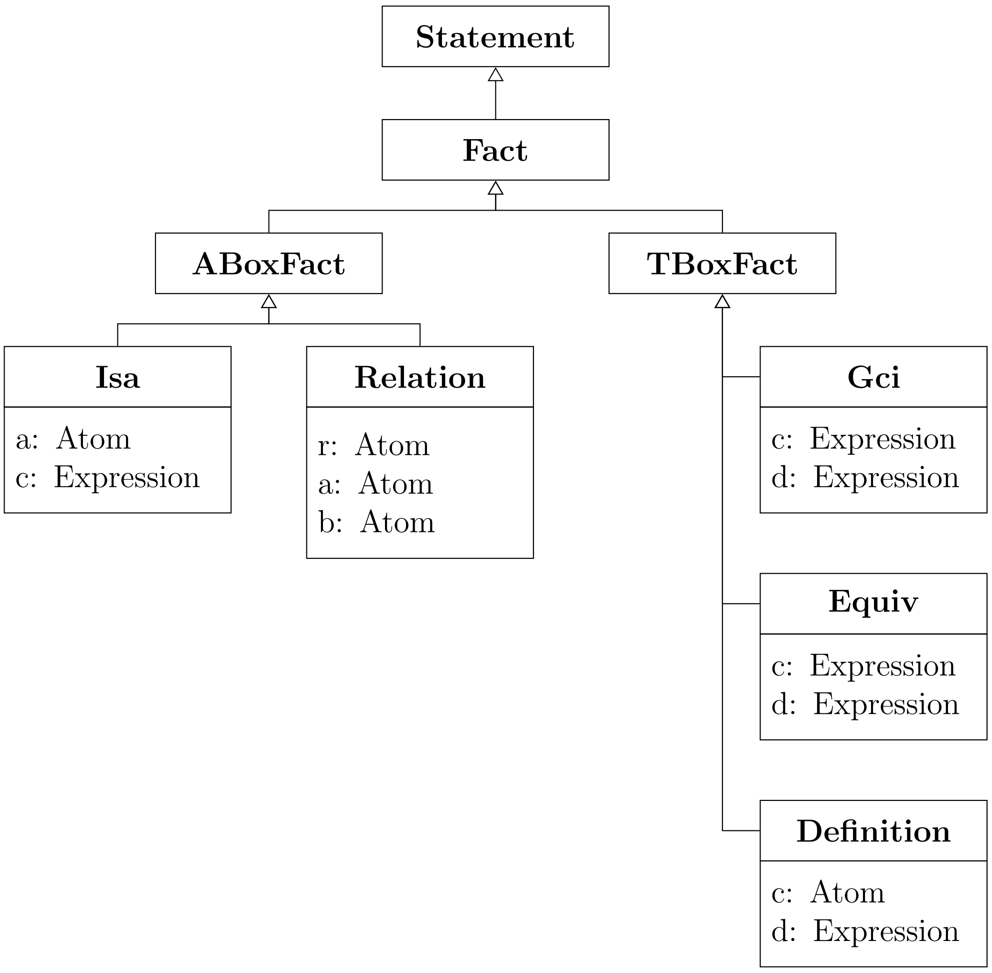

Figure A.2: Fact classes

The knowledge base is defined in terms of the ABox and TBox facts and its classes are declared as follows:

```
case class ABox(val isas: List[Isa], val relations: List[Relation])
case class TBox(val gcis: List[Gci], val equivs: List[Equiv])
case class DLKnowledgeBase(val abox: ABox, val tbox: TBox)
```

It has to be noted that all of the above mentioned classes are definded as persistent data structures, which means that an object, once created is never changed. All methods, which modify these immutable object actually create a new object with the modified information. This method is widely used in functional programming and simplifies the implementation of algorithms without any serious overhead.

The description logic reasoner is implemented in terms of the following classes:

```
trait DLReasoner {
  def entails(query: Fact): Boolean
  def isConsistent: Boolean
}

class TableauDLReasoner extends DLReasoner

trait DLReasonerFactory {
  def create(kb: DLKnowledgeBase): DLReasoner
}
```

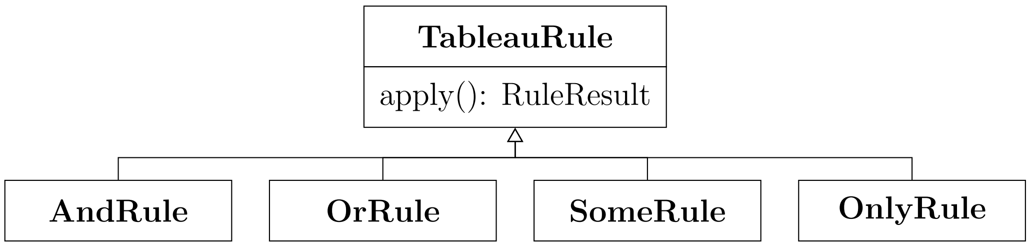

Figure A.3: Rule classes

The implementation of the `DLReasonerFactory` trait creates an instatnce of the `DLReasoner` trait. The `TableauDLReasoner` class is the implementation of the tableau algorithm presented in Section 3.1.2. The reasoner operates on a list of tableau rules shown in Figure A.3. The reasoner tries to apply each rule from the list in the given order. If none of the rules can be applied, the current tableau branch is treated as open and the input knowledge base is consistent. If a clash is detected, the algorithm tries to apply the tableau rules to subsequent branches. The results of applying a tableau rule are defined in terms of the `RuleResult` class, which is defined as follows:

```
class RuleResult
case object NotApplied extends RuleResult
case class Applied(tab: Tableau) extends RuleResult
case class Branch(tab1: Tableau, tab2: Tableau) extends RuleResult
```

The rule may either be unapplicable, or it may produce one or two resulting tableaux. If two tableaux are generated, they form two branches, which have to be expanded.

In order to load a knowledge base from a file or to parse a user input query, the `DLParser` class provides the functionality to parse one fact or multiple facts defined one in each line. The parser accepts the Manchester OWL syntax for defining concepts.

The following snippet of code shows an example usage of the description logic reasoner component:

```
val parser = new DLParser
val lines = scala.io.Source.fromFile("kb.txt").getLines
val kb = parser.parseFacts(lines)
val dlReasoner = TableauDLReasonerFactory.create(kb)
val queryString = "Woman(MARY)"
val query = parser.parseFact(queryString)
println(queryString + ": " + dlReasoner.entails(query))
```

A.2 Default logic reasoner

The default logic reasoner component builds on top of the description logic reasoner by adding the capabilities to represent defaults and provide inferences from a knowledge base with defaults. Figure A.4 shows the `Default` class, which represents a default, where all elements are concept descriptions. Furthermore, the `ClosedDefault` class combines an open default with a concrete individual forming a closed default.

The DL$\mathfrak{D}$ (description logic with defaults) knowledge base is defined as a class aggregating a DL knowledge base and a set of defaults.

```
case class DLDKnowledgeBase(dl: DLKnowledgeBase, defaults: Set[Default])
```

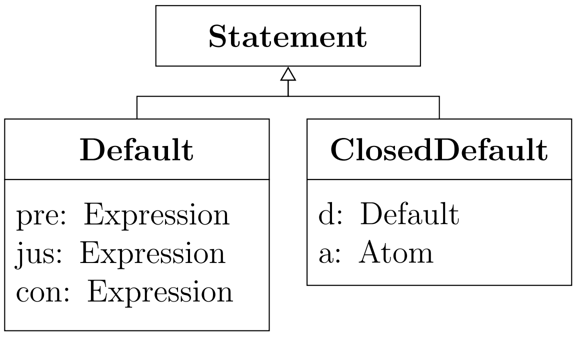

Figure A.4: Default classes

The reasoning service for default logic is defined as follows:

```
class DLDReasoner(val kb: DLDKnowledgeBase,
                  val dlReasonerFactory: DLReasonerFactory) {
  def entails(query: Fact, skeptical: Boolean): Boolean
}
```

The `DLDReasoner` uses the supplied DL reasoner and provides the `entails()` method, which can answer skeptical and credulous queries, as indicated by the second argument. Similarly to the `DLParser`, the `DLDParser` class can be applied to parse a DL$\mathfrak{D}$ knowledge base containing both DL statements and defaults. One of the more important functions executed while determining the extensions of a default theory is `generatingDefaultSets()`, which computes the sets of generating defaults for all extensions. It is an implementatoin of the Algorithm 3.1 presented in Chapter 3 and is declared as follows:

```
def generatingDefaultSets(kb: DLDKnowledgeBase):
  Iterable[Iterable[ClosedDefault]]
```

This function actually returns a set of extensions, each expressed as a set of closed defaults. Based on this information answers to both credulous and skeptical queries can be derived.

A.3 Default transformations

The default transformations component implements Algorithm 4.1. The `query` method answers a query in the form $A \sqsubseteq B$? returning one of the following results:

- An empty set representing False
- A single-element set containing the subsumption statement $A \sqsubseteq B$
- A set of defaults in the form $\frac{A : B \sqcap J}{B}$

The `DefaultTransformations` class is defined as follows:

```
class DefaultTransformations {
  def query(query: Fact, kb: DLDKnowledgeBase): Set[Statement]
}
```

A.4 Distributed reasoning

The distributed reasoning implementations is defined in terms of the following classes:

```
class SendQueryRule extends TableauRule
class QueryingDLReasoner extends TableauDLReasoner // includes SendQueryRule
class ReasoningAgent(val kb: DLDKnowledgeBase) extends Actor
class ReasoningAgentTask(val query: Query, var kb: DLDKnowledgeBase)
```

The `QueryingDLReasoner` is an extension to the tableau reasoner, which adds an additional rule presented in Figure 5.2 and implemented as the `SendQueryRule` class. The `ReasoningAgent` class implements a single agent, which possesses its own knowledge base and can connect to other agents in the process of reasoning. The `ReasoningAgentTask` encapsulates one resoning task and is initialized with a query received by the agent.

Figure A.5 presents the hierarchy of classes with the base class `Query` represent the possible queries, which an agent can receive. The following simple code is used to send a query to an agent and store the result value.

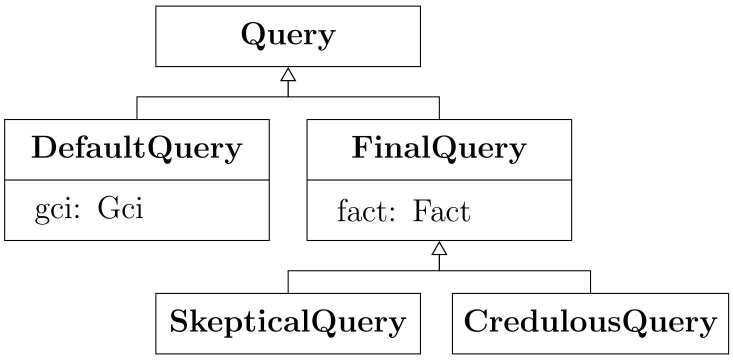

Figure A.5: Query classes

```
val agent: ReasoningAgent
val query: Query
val response = agent !? query
```

The `!?` operator is a standard method to send messages and receive replies in the Scala Actors library. This code stores the agent's reply in the `response` variable.