8

Description Logics Systems

Ralf Möller

Volker Haarslev

Abstract

This chapter discusses implemented description logic systems that have played or play an important role in the field. It first presents several earlier systems that, although not based on description logics, have provided important ideas. These systems include Kl-One, Krypton, Nikl, and Kandor. Then, successor systems are described by classifying them along the characteristics discussed in the previous chapters, addressing the following systems: Classic ("almost" complete, fast); Back, Loom (expressive, incomplete); Kris, Crack (expressive, complete). At last, a new optimized generation of very expressive but sound and complete DL systems is also introduced. In particular, we focus on the systems Dlp, FaCT, and Racer and explain what they can and cannot do.

8.1 New light through old windows?

In this chapter a description of the goals behind the development of different DL systems is given from a historical perspective. The description of DL systems allows important insights into the development of the knowledge representation research field as a whole. The design decisions behind the well-known systems which we discuss in this chapter do not only reflect the trends in different knowledge representation research areas but also characterize the point of view on knowledge representation that different researchers advocate. The chapter discusses general capabilities of the systems and gives an analysis of the main language features and design decisions behind system architectures. The analysis of current systems in the light of a historical perspective might lead to new ideas for the development of even more powerful description logic systems in the future. References to previous descriptions of DL systems (e.g., in [MacGregor, 1991a; Woods and Schmolze, 1990; Horrocks, 1997a]) or publications on DL theory that also contain discussions about description logic systems (e.g., [Patel-Schneider, 1987a;

Nebel, 1990a; Schmidt, 1991]) are included where appropriate. For references to other systems not mentioned here see also [Woods and Schmolze, 1990] and [Nebel, 1990b, p. 46f., p. 63f.].

In Chapter 2 basic concept and role constructors were already introduced (see also the appendix for a summary of syntax and semantics of DL constructors). However, before starting the discussion about DL systems it is appropriate to introduce some notation for language constructors in order to keep this chapter self-contained. It is assumed that the reader is familiar with the basic description logics $\mathcal{AL}$ and $\mathcal{ALC}$. In a similar way as in Chapter 2, further language features are indicated by different letters. The letter $\mathcal{N}$ is used for simple number restrictions and the letter $\mathcal{Q}$ is used for qualified number restrictions. $\mathcal{H}$ is used for role hierarchies with multiple parents whereas h is used for role hierarchies with single inheritance only. In some languages, no role hierarchies but role conjunctions are provided. Role conjunctions are indicated with the letter $\mathcal{R}$ in the following. In addition, the abbreviations $\mathcal{F}$ and f are used for features with and without equality for feature chains (i.e., agreements), respectively. The index R^+ is used to indicate support for transitive roles. Language constructors for an extensional specification of concepts using nominals (or individuals) are denoted by the letters $\mathcal{O}$ and $\mathcal{B}$ (see Chapter 2 or the appendix for details). If inverse roles are supported by a DL system, this is indicated either with a superscript $^{-1}$ or with the letter $\mathcal{I}$. The latter variant is used in order to allow for a convenient pronunciation of the DL language.

8.2 The first generation

Inspired by research on human cognitive behavior, proposals for knowledge representation languages were first discussed in the late sixties. E.g., [Quillian, 1967] is one of the first publications of these languages called "semantic networks" (see also [Quillian, 1968]). Originally, *semantic network formalisms* were seen as alternatives to first-order logic. In a similar spirit, [Minsky, 1981] introduced the initial notion of a *frame system*. The motivation of these representation formalisms was to mimic human reasoning in the sense of achieving "cognitive adequacy". Thus, the idea was to support problem solving with appropriate representation structures that somehow "resemble" representation structures assumed in human information processing. The exploitation of inheritance was a predominant idea in frame systems. The specification of knowledge bases should be simple and the use of the representation structures should be intuitive ("epistemological adequacy"). However, as pointed out by [Woods, 1975], it was not at all simple to specify what an inference system was supposed to actually compute. The late seventies saw initial research on the relation of frame systems and first-order logic [Hayes, 1977; 1979] which revealed that some aspects of frame-based systems can be considered as special “instantiations” of first-order reasoning. Hayes argued that frame-based reasoning was not an entirely new way of knowledge representation with particular advantages over first-order reasoning. Specific features of frame systems beyond first-order reasoning (e.g., defaults) were not very well understood at that time. The consequence of these publications was that many researchers did not consider frame systems and semantic network systems as possible alternatives to logic-based approaches any more.

The criticisms of early frame systems and semantic network formalisms stimulated research on the development of mathematical structures and techniques for defining the semantics of representational constructs supported by different representation languages. For instance, in early frame systems there was no clear distinction between constructs for representing “generic” knowledge about sets of individuals and knowledge about “specific” individuals. Furthermore, frames were often used as data structures in procedural programs. For these programs a formal specification of what they were expected to compute was rarely provided. Rather than interpreting frame structures as data structures, [Woods, 1975] suggested to use a formal semantics to clearly specify what is to be computed by inference algorithms.

Kl-One

Inspired by critics such as [Woods, 1975], Brachman started to develop a new representation system (called Kl-One) that inherently included the notion of inferring implicit knowledge from given declarations [Brachman, 1977b; 1979]. Although the initial approach was not logic-based, Kl-One started the era of logic-based representation systems which can be used to formalize application problems as inference problems over the constructs supported by the representation language. One of the prevailing inference patterns is centered around inheritance [Brachman, 1983]. The final report on the Kl-One language is published in [Brachman and Schmolze, 1985].

One of the core ideas behind Kl-One as a representation language for the “epistemological level” resulted from problems with languages offering built-in primitives for general representation purposes (e.g., CD theory [Schank, 1975]). Rather than providing general built-in primitives, in Kl-One, for a specific representation problem a set of adequate primitives was defined by the user. The primitives were denoted by so-called *concept names*. The next idea was to use *concept-forming operators* to build new concepts from basic concepts. These compound concepts were also referred to as “concepts”, “concept terms” or “concept descriptions”. *Generic concepts* were intended to denote classes of individuals and *individual concepts* were intended to denote individuals (see also [Nebel, 1990a, p. 42]). Individuals were related by so-called *roles* which, in turn, could be primitive roles (role names) or roles described with role constructors [Brachman and Schmolze, 1985].

In Kl-One, concepts and roles are the building blocks for representational purposes. The main idea behind concepts and concept constructors in Kl-One is that the *meaning of a concept* is derived only from the meaning of its superconcepts and other restrictions associated with a concept [Brachman and Schmolze, 1985]. A Kl-One generic concept consists of a set of superconcept names, a set of role descriptions, and a set of structural descriptions [Patel-Schneider, 1987a, p. 58f.].[1] Roles can be viewed as potential relationships between an individual of a certain class and other individuals in the world [Nebel, 1990a, p. 42].

Role descriptions could be either restrictions or differentiations. The former restricted the class of permitted fillers (value restrictions) or the number of fillers (number restrictions). Role differentiations were used to describe a subrole with possible value or number restrictions. So-called structural descriptions were used to state relationships between the fillers of roles (see also [Patel-Schneider, 1987a, p. 58f.]). Descriptions for individual concepts consisted simply of a set of values for roles plus a set of generic concepts. Individual concepts were seen as *instances* of these generic concepts, i.e., an individual concept had to satisfy all restrictions (and differentiations) inherited by the generic concepts. On the other hand, individual concepts were also *subsumed* by their generic concepts. However, the semantics of individuals was never completely worked out (see [Schmolze and Brachman, 1982, p. 23–31] cited after [Nebel, 1990a, p. 64]).

The representation structures offered by Kl-One were similar to those offered by semantic networks or frames. Although, initially, the structures offered by Kl-One were called "structural inheritance networks" [Brachman, 1977b; 1979], in [Brachman and Levesque, 1984] the authors talk of "frame structures".[2] In accordance with [Nebel, 1990a, p. 45] we argue that in contrast to, e.g., CD theory [Schank, 1975], providing a (large) set of primitive representation structures (names) for all kinds of representation purposes was not the development goal of Kl-One. As Nebel points out [Nebel, 1990a, p. 45], more important and unique for Kl-One is the core idea of proving ways to specify *concept definitions*, i.e., the possibility to let a knowledge engineer declare the relation of "high-level concepts" to "lower-level primitives".

A concept definition was an assignment of a (unique) name to a concept term. In Kl-One the well known distinction between the two kinds of concept definitions,

[1] Note that, in Kl-One-like languages, there are specific syntactic constructs for specifying superconcepts. These specific constructs are no longer present in logic-based concept languages of the nineties.

[2] There are large differences between frame systems and description logic systems: if for i the restriction $\forall R.C$ holds, and we set i into relation to j via the role r, then every Kl-One-based system concludes that j is an instance of C. In standard frame-based systems, j can only be set into relation to i via R if it is already known that j is an instance of C. Otherwise, in frame systems at least a warning is issued or even an error is signalled.

definitions with necessary and sufficient conditions and definitions with only necessary conditions (so-called primitive definitions), was investigated for knowledge representation purposes for the first time.[3] In the original approach no cycles were allowed in the set of concept definitions.[4] The most important consequence of the introduction of concept definitions with necessary and sufficient conditions was that reasoning about the relationships between concepts became important. In Kl-One there is still the notion of a "told subsumer" syntactically being explicitly mentioned in a list of so-called superconcepts but, according to the semantics, there are also additional *computed* subsumers which are concept names (direct subsumers or direct superconcepts). Note that inferences in Kl-One were based on the open-world assumption. Hence, rather than with frame systems where the names as superconcepts are always given explicitly, Kl-One introduced the idea that the set of direct superconcepts (i.e., concept names) for a given concept must be inferred.

Direct superconcept/subconcept relationships (also called parent/children relationships) are dependent on the concept terms used in the definitions of a TBox. In particular, the notion of defined concepts (with necessary and sufficient conditions) led to the idea of *classifying* a TBox. The idea was to compute the subsumption hierarchy (sometimes also called "inheritance hierarchy") of parents and children for each concept name mentioned in a TBox during a so-called *classification* process. The intention was that a model for a specific application domain could be verified by a knowledge engineer based on the subsumption hierarchy. Considering the subsumption hierarchy, i.e., the lattice of direct superconcepts, the idea was also that concept *terms* could be automatically "inserted" between named concepts in the hierarchy. Hence, concept terms could be set into relation to "predefined" concept names (and, indirectly, other concept terms). This feature has been used in many projects for implementing application functionality.

The first development of an algorithm for computing the subsumption hierarchy of a TBox (the "classifier") is described in [Schmolze and Lipkis, 1983]. Another inference component called "realizer" computes for each individual mentioned in an ABox the most-specific atomic concepts (or concept names) of which the individual is an instance. One of the first algorithms for computing the realization of an ABox is described in [Mark, 1982]. Initial Kl-One systems were implemented in Interlisp [Lipkis, 1982] and Smalltalk [Fikes, 1982]. The Consul project [Kaczmarek *et al.*,

[3] In the literature, some authors use the word "definition" as a synonym for concept terms themselves (e.g., [Schmidt, 1991], see also [Woods, 1991, p. 65]). In this case, "primitive" concepts with only necessary conditions were introduced with a specific marker to be used in concept terms.

[4] The semantics of cycles was analyzed in [Baader, 1990b; 1991; Nebel, 1990a; 1991]. The so-called *descriptive semantics* provided many advantages compared to so-called *fixed point semantics*. For details see [Nebel, 1990a]. One of the first publications of an expressive description logic supporting cyclic axioms with a descriptive semantics and a sound and complete calculus is [Buchheit *et al.*, 1993a]. Cyclic axioms are usually not considered as concept definitions.

1986] was one first projects in which classifier and realizer inference services were first exploited.

First investigations about defaults and exceptions were published in [Brachman, 1985]. Nowadays, the semantical theory of defaults in description logics is much clearer, see [Baader and Hollunder, 1992; 1993; Baader and Schlechta, 1993; Padgham and Zhang, 1993; Padgham and Nebel, 1993; Baader and Hollunder, 1995a; 1995b; Donini *et al.*, 1997b].

At the first KL-ONE workshop [Schmolze and Brachman, 1982] it became clear that the informal specification of the semantics of KL-ONE concept and role constructors led to serious problems. The development of the classifier [Schmolze and Lipkis, 1983] was based on the intuitive meaning of the KL-ONE formalism [Nebel, 1990a, p. 46]. Attempts to logically reconstruct the representation constructs, e.g., [Schmolze and Israel, 1983; Israel and Brachman, 1984], resulted in a deeper understanding of the formalism. Given the formal semantics, implemented algorithms for classification and realization were shown to be incomplete. Later investigations revealed that KL-ONE (with the formal semantics given in the logical reconstruction approaches) is undecidable (e.g., this holds for the combination of conjunction, value restrictions and role-value-maps [Schmidt-Schauß, 1989]). In [Brachman and Levesque, 1984] the first thoughts about tractability of subsumption for sublanguages are discussed. Terminological reasoning with concept definitions even for sublanguages with low expressiveness were shown to be inherently intractable in the worst case [Nebel, 1990b, p. 28, p. 71f.]. Proposals for a semantics based on many-valued logics (e.g., [Patel-Schneider, 1986; 1987a; 1987b; 1989a]) ensure tractable algorithms concerning concept consistency reasoning but also result in a weak expressiveness: many intuitive inferences are not sanctioned by this semantics (see also [Nebel, 1990a]).

Another result of [Schmolze and Brachman, 1982] was that the semantics of individual concepts was not quite clear (e.g., concerning coreference and unique name assumption, see above). Thus, at the first KL-ONE workshop [Schmolze and Brachman, 1982], the notions of a *hybrid* reasoning system consisting of a TBox (a set of concept definitions) and an ABox (a set of assertions concerning individuals) were made more precise. The change of the view on KL-ONE spelled out in [Schmolze and Brachman, 1982, pp. 8–17] (see also [Nebel, 1990a, p. 46]) can be summarized as follows: It is not the *names* of representation structures that are important but the functionality, i.e., the declaration and inference services which the system provided. It was first pointed out that inferences have to be formally defined based on the semantics of the representation formalism. This view led to the development of the functional view of knowledge representation as pursued with the development of the system KRYPTON.

KRYPTON

The knowledge representation system KRYPTON [Brachman *et al.*, 1983b; 1983b; 1985] can be seen as the first approach to define a new language of the KL-ONE family with a formal, Tarskian semantics. Furthermore, the goal was to overcome the problems with individual concepts in KL-ONE [Nebel, 1990a, p. 63]. The hybrid representation approach with a TBox and an ABox was first implemented in the KRYPTON system (see also [MacGregor, 1991a, p. 391]). Similar to KL-ONE the distinction between primitive and defined concepts and the computation of the most-specific atomic concepts which instantiate individuals is one of the core ideas of KRYPTON.

KRYPTON offered a concept language with low expressiveness. While the initial approach [Brachman *et al.*, 1983b] was too expressive to be tractable (see also [MacGregor, 1991a, p. 390]), in a revised version [Brachman *et al.*, 1985] the concept constructors of KRYPTON were defined as conjunction, value restrictions and role chains. Thus, subsumption checking was polynomial [Patel-Schneider, 1987a, p. 75]. For the ABox a full-fledged resolution-based FOPL theorem prover [Stickel, 1982] was proposed, i.e., the ABox reasoner of KRYPTON was incomplete. Another perspective is that KRYPTON started with a first-order logic theorem prover and augmented it with a special-purpose inference system for terminological reasoning to cut out some of the combinatorial search [Vilain, 1985]. KRYPTON can be regarded as one of the first efforts in combining knowledge representation and theorem proving techniques but was not used for industrial applications [Nebel, 1990a, p. 63f.].

Rather than dealing with specific representation structures and operations on them, KRYPTON offers a so-called "functional approach". Using the interface functions "tell" and "ask", a knowledge base can be defined and queries can be answered about it. In this sense, a "functional approach" means that a formal representation system does not necessarily have to maintain, for instance, frame structures, the subsumption hierarchy, or even an ABox as a graph structure. If, for the internal implementation purposes, graph structures are indeed used, they are nevertheless hidden from the user in order to avoid "procedural" operations to be carried out with internal record structures. Arbitrary procedural operations are usually not related to the semantics of the representation formalism such that, in this case, it is hard to characterize what is actually represented and what is computed as solutions to inference problems. Thus, the focus of KRYPTON was not on the structures to be maintained by the system but was centered around the question about what should the system do for the user, i.e., what services should be made available. In other publications this idea was described as the "knowledge level" [Newell, 1982]. In KRYPTON, inference services for concept terms are checks for concept consistency, disjointness, and subsumption. For a TBox, the most-specific subsumers (parent/children relation) can be computed, whereas for an ABox, consistency, instance checking, realization (direct types) and instance retrieval are offered as inference services. KRYPTON pioneered the idea that the user should only know, at some level not dependent on implementation details, what questions the system is capable of answering and what operations are permitted that allow new information to be provided to it. For instance, it is not important how the association between an individual and a certain role filler is actually represented in terms of memory arrangements (called the symbol level). What counted for the underlying implementation was what operations must be supported in order to answer queries at the semantical level. This view about KL-ONE-based representation systems was one of the major achievements of the KRYPTON project.

NIKL , PENNI , KL-TWO

At the same time as KRYPTON, the knowledge representation system NIKL was developed as a successor of KL-ONE. NIKL was a New Implementation of KL-ONE [Schmolze and Israel, 1983; Schmolze, 1985; Schmolze and Mark, 1991]. As discussed in [Kaczmarek *et al.*, 1986] in NIKL, roles are also ordered with respect to subsumption (see also [Schmidt, 1991, p. 13]).

The assertional components of KL-ONE were initially discarded in the NIKL system (see the NIKL user guide [Robins, 1986]). Compared to the initial KL-ONE implementation, the algorithms in the NIKL classifier were faster in the average case because "obvious" information was exploited to a larger degree (see [MacGregor, 1988, p. 405] or [MacGregor, 1991a, p. 392]). However, the subsumption algorithm of NIKL was incomplete and it was hard to characterize which inferences are omitted [Schmolze and Israel, 1983] (see also [Patel-Schneider, 1987a, p. 74]).

Later, an assertional reasoning component was added with the system PENNI which is based on RUP [McAllester, 1982]. The resulting system was called KL-TWO [Vilain, 1985] (see also [Schmidt, 1991, p. 15]). In KL-TWO a propositional reasoner with equality (the PENNI subsystem) was augmented with a so-called quantificational reasoning component (the NIKL subsystem). For the propositional part in the PENNI component, incremental additions and retractions were supported due to the facilities provided by RUP. However, as shown in [Patel-Schneider, 1989b] the concept language of NIKL contained concept and role constructs that render the satisfiability problem for NIKL concept terms undecidable (see also [Schmidt-Schauß, 1989]).

Concerning hybrid reasoning, i.e., the systematic integration of TBox and ABox reasoning, there were shortcomings as well. Because in RUP different constants do not necessarily denote different objects, the unique name assumption was not built into the assertional component PENNI. Thus, number restrictions imposed by NIKL concepts often did not have the intended effects concerning hybrid reasoning. Other sources of incompleteness were pointed out (see also the analysis of "inferential gaps" in [Nebel, 1990a, p. 63f.]). The research on the KL-TWO system demonstrated that hybrid reasoning is not just a matter of integrating reasoning subsystems at the software level. Hybrid reasoning requires a dedicated architecture implementing a sound and complete calculus which, in turn, can be developed only after a deep analysis of the semantics of the representation constructs. Nevertheless, the principle idea of exploiting subsumption information for resolution-based first-order reasoning has been integrated in many theorem proving systems.

KANDOR

Research on KANDOR [Patel-Schneider, 1984] was influenced by the KRYPTON architecture and the performance problems of the NIKL approach. The goal of KANDOR was to increase the expressive power of the terminological representation component in such a way that an efficient subsumption algorithm could be developed. Basically, KANDOR supported conjunction, value restriction and number restrictions as concept-forming operators. In minimum number restrictions, range-restricted roles could be used (hence, qualified minimum number restrictions are allowed, see also [Patel-Schneider, 1987a, p. 76]). In order to provide effective inference algorithms (e.g., for information retrieval scenarios) in the KANDOR approach the expressiveness of the assertional component was cut down to a representation system comparable to a database (without revision mechanisms). Subsumption in KANDOR was shown to be coNP-complete (see [Nebel, 1988], and [Nebel, 1990a, p. 90] for details). The initially proposed subsumption algorithm with polynomial runtime must have been incomplete.

KANDOR was called a frame-based system (which might be reasonable because of the expressiveness offered by the ABox language). A frame in KANDOR was essentially a specification of conditions for describing how an individual can be an instance of it (in terms of superframes and restrictions). KANDOR supported defined frames and primitive frames in the spirit of KL-ONE. The system adopted the "small interfaces" approach of KRYPTON, i.e., models were built using the declaration interface (tell interface), and application services were realized with the query interface (ask interface). Although called a frame system, frames were not treated as record structures to be manipulated by procedural programs. The authors of KANDOR argued for a small knowledge representation system that could be used as part of larger systems with different subcomponents. The main achievement of KANDOR was the introduction of a small-can-be-beautiful approach which, finally, led to the design of the system CLASSIC which will be discussed in detail in the next section.

8.3 Second generation Description Logics systems

Whereas the prototypical implementations of first generation systems were used to study knowledge representation problems, second generation DL systems have been more extensively used in serious applications. The implementations discussed in this section are not only prototypes but were much more stable. In addition, since the beginning of the nineties, the systems have been called description logic systems. We first discuss systems for (almost) tractable languages based on (almost) complete algorithms and investigate systems for expressive description logics afterwards

CLASSIC

The basic CLASSIC system supported the logic $\mathcal{ALNF}h^{-1}$ with TBoxes and ABoxes plus facilities for dealing with numbers [Borgida *et al.*, 1989]. We use the lower-case letter h to indicate that CLASSIC supports only role inclusion but no role conjunction, i.e., CLASSIC supports “single-inheritance” role hierarchies. CLASSIC is available for research purposes. Implementation languages for CLASSIC are COMMONLISP [Steele, 1990] and C. The interfaces are described in [Resnick *et al.*, 1995]. Full CLASSIC also contained the concept constructors $\mathcal{O}$ and $\mathcal{B}$ for referring to individuals in concept terms.

Subsumption in full CLASSIC was initially assumed to be polynomial [Borgida *et al.*, 1989]. Problems with individuals in full CLASSIC were recognized in [Patel-Schneider *et al.*, 1991]. At the same time, subsumption in CLASSIC was shown to be coNP complete [Lenzerini and Schaerf, 1991]. In the modified semantics for the concept constructors $\mathcal{O}$ and $\mathcal{B}$ (see [Borgida and Patel-Schneider, 1994]) the interpretation function maps individuals in concept terms to disjoint sets of domain objects. With this semantics concerning individuals the inference algorithms of the CLASSIC system could be shown to be complete [Borgida and Patel-Schneider, 1994]. However, given the non-standard semantics for the concept constructors $\mathcal{O}$ and $\mathcal{B}$, the same effect can be achieved with existential quantifications and disjunctions w.r.t. atomic concepts:[1] For each individual I a new atomic concept A_I can be introduced. Note that atomic concepts are also mapped to sets of individuals. Additionally, since CLASSIC imposes the unique name assumption, a set of axioms ensures that the new atomic concepts are disjoint. Now every term of the form $\exists R.I$ can be replaced by $\exists R.A_I$. Terms of the form $\{I_1, \ldots, I_n\}$ can be replaced by $A_{I_1} \sqcup \ldots \sqcup A_{I_n}$. In an ABox, for each individual I a concept assertion is added to

[1] Note that these concept constructors are not directly provided by CLASSIC.

ensure that the individual is an instance of the associated atomic concept A_I. Thus, only in an ABox, a real coreference between roles can be enforced. On the one hand, we can call the Classic system "almost" complete. "Almost" refers to non-standard semantics w.r.t. individuals being supported by current system implementations. On the other hand, the transformation makes clear that in Classic nevertheless a limited kind of disjunction (with concept names for which no definitions exist) can be expressed while retaining polynomial inference algorithms.

The recommended techniques for knowledge-based system development with Classic are outlined in [Brachman *et al.*, 1991]. As Brachman [Brachman, 1992, p. 256] points out, a tractable description logic does not guarantee that a system is useful in practice. Therefore, the Classic system was also carefully designed to meet practical requirements and to guarantee predictable system behavior. The context in which the system was expected to be used required that many queries were given to knowledge bases which rarely change. The architectural design of Classic supported a precomputation of index structures such that queries can be answered quickly (mostly by simple storage retrieval). The architecture is made possible by a careful selection of the concept and role constructors for the description logic language. Inference services for the description logic supported by Classic can be implemented by transforming concept expressions into a normal form ("structural subsumption"). Once the normal form is computed, queries can be answered by inspecting the data structures used to encode the normal form. It should be noted that, in Classic, retraction of told information is possible but not optimized.

Another facility offered by Classic is a rule system. Rules are applied to individuals explicitly named in the ABox. Furthermore, rules are applied in a forward-chaining way. Basically, a rule has a precondition (a concept) and a conclusion (also a concept). If it can be shown that an individual mentioned in the ABox is an instance of the precondition concept, a concept assertion for stating the membership of the individual in the conclusion concept is added to the ABox. In order to provide support for modeling, the rule base is statically checked for inconsistencies. For instance, if there are two rules whose preconditions subsume each other, the conclusions must not be disjoint.

Furthermore, Classic provides simple support for closed-world reasoning ([Resnick *et al.*, 1995], see also [Weida, 1996]). Closing a role for an individual means adding an appropriate maximum number restriction for the role. The maximum number of fillers is restricted to the largest integer such that the minimum number restriction with this integer (and the corresponding role) is entailed by the knowledge base. The problem with role closing is that in combination with rules, the exact sequence of several closing operations determines what actually holds in the resulting ABox. These and other problems concerning different closing operations have to be considered with default reasoning as theoretical background [Baader and Hollunder, 1995a; 1995b; Donini *et al.*, 1997b; Rosati, 1998]. For a specific approach concerning the integration of defaults into the CLASSIC system see also [Wahlöf, 1996; Lambrix *et al.*, 1998].

CLASSIC is one of the first systems that provided support for incorporating inferences over other domains. Consistency and subsumption checking for expressions of another domain (e.g., the reals) can be integrated into the CLASSIC system via an extension interface [Borgida *et al.*, 1996]. CLASSIC was one of the first description logic systems designed with respect to users which are non-experts in description logic theory. An important lesson learned by the CLASSIC approach and its applications was the importance of explanation and output pruning facilities [McGuinness and Borgida, 1995; McGuinness, 1996; Borgida and McGuinness, 1996]. Moreover, CLASSIC was the first system capable of supporting some reasonable form of error reporting [Brachman, 1992]. However, at the current state of the art there is hardly an adequate measure for the quality of these indispensable services [Brachman, 1992, p. 253].

Although CLASSIC was a very successful description logic modeling environment, the low expressiveness of the CLASSIC description logic made it hard to use the system in many kinds of applications. In many cases, users wanted more expressiveness [Patel-Schneider *et al.*, 1990]. In the following sections we discuss systems for (more) expressive description logics. As can be expected, increases in expressiveness came at a certain price. The predictability of the behavior of CLASSIC in terms of performance could not be reached by systems implementing complete algorithms for more expressive DLs. On the other hand, incomplete algorithms have the problem that results computed by a system cannot be trusted in general. Thus, the complete-incomplete debate for expressive description logic systems started at the end of the eighties and the beginning of the nineties. First, we describe the systems LOOM and BACK, which are based on incomplete algorithms. Afterwards, initial research on description logic systems based on complete algorithms is summarized with a discussion of the systems KRIS and CRACK.

LOOM

The LOOM architecture [MacGregor and Bates, 1987; MacGregor, 1991b] offers TBox and ABox reasoning facilities for a description logic that can be characterized by the name $\mathcal{ALCQRIFO}$ plus additional constructs for dealing with real numbers (see also [Brill, 1994] or [Horrocks, 1997a, p. 43]). LOOM is based on KL-ONE, i.e., concept definitions with necessary or with necessary and sufficient conditions play an important role in domain modeling with LOOM. It should be emphasized that truth maintenance facilities for revision were built into the LOOM architecture right from the beginning and have influenced the design of the whole system [MacGregor, 1988;

MacGregor and Brill, 1992]. While first Loom versions were based on description logics [MacGregor and Brill, 1992] in later versions an attempt was made to develop a "description classifier for the Predicate Calculus" [MacGregor, 1994]. For instance, facilities for dealing with definitions for relations were added. The current version of Loom is implemented in CommonLisp and is available for research purposes. A new system (called PowerLoom) for CommonLisp as well as C and Java-based platforms can be licensed as well.

A distinguishing design goal of Loom was the incorporation of an expressive query language for retrieving ABox individuals. Another design goal of Loom was to support rule-based programming [Yen *et al.*, 1991b; 1991a; MacGregor and Burstein, 1991]. Based on the rule system, it is possible to specify additional necessary conditions for individuals which (i) are explicitly mentioned in the ABox and (ii) are derived to be instances of a certain defined concept. The additional necessary conditions are called "implications" in Loom [MacGregor, 1988]. The additional necessary conditions specified by rules are not exploited, for instance, for TBox reasoning. Note that an "implication" $A \rightarrow B$ stated by a Loom rule does not mean that $\neg B \rightarrow \neg A$ holds, i.e., rule-based "implications" are not to be confused with true logical implications as provided by generalized concept inclusions that are now standard in newer systems (see below).

In order to meet the performance requirements of the applications for which Loom was developed (e.g., natural language and image interpretation), incomplete algorithms for concept consistency and subsumption are implemented. Concerning ABox reasoning, Loom applications required specific strategies to avoid the computation of unused results. Rather than employing the usual forward-chaining strategy of computing the most-specific atomic concepts of which the ABox individuals are instances, Loom uses a scheme that considers the queries being posed to the system. Thus, backward-chaining strategies for query answering are used in the implementation [MacGregor and Brill, 1992]. However, for the rule system, it is important to detect whether an individual is an instance of a concept that is used as a precondition of a rule. In this case, forward-chaining techniques are exploited [MacGregor, 1991b; MacGregor and Brill, 1992]. The combination of forward-chaining and backward-chaining inferences can be specified for a certain application problem by "marking" concepts accordingly. The user can control the inference process by these means but is also responsible for estimating the effects of these declarations.

The arguments for the Loom approach can be summarized as follows: The intractability of the representation language can hardly be avoided to fulfill the requirements of users. Therefore, the idea is to support the features in one system rather than as a set of application-specific ad hoc supplements ("Where resides the scruffiness?" [MacGregor, 1991a, p. 396]). Obviously, incompleteness is no problem as long as the answers of the inference system are interpreted in the right way (i.e., "no" answers should not be trusted). Several researchers argued that there is always the inherent danger that non-expert users either do not know this or might not recognize this as a potential danger (cf. the work on complete systems [Baader and Hollunder, 1991a; 1991b] discussed below). However, if a combinatorial explosion occurs in a complete algorithm, in practice, no result is available as well. Concerning incomplete algorithms for decidable description logics, similar arguments as for other modeling environments based on first-order logic can be mentioned: If, in a certain application, concept terms are checked for consistency and a combinatorial explosions occur in complete algorithms, incomplete algorithms at least might provide some support, e.g., for building a TBox. Just signalling a timeout during the execution of a complete algorithm that runs into a combinatorial explosion might result in less information. In this case, an incomplete algorithm might succeed in finding at least some inconsistencies. Note however, that in modern inference system technologies supporting complete reasoning, incomplete reasoners are used as "preprocessors" in order to speed up inferences (see the next chapter).

LOOM supports different kinds of individuals (classified instances, light instances, CLOS instances). For different kinds of instances different levels of inference services are supported, e.g., for classified instances, the set of most specific atomic concepts of which the classified individual is an instance is computed once new assertions are specified. Thus, for classified instances, the rule-based forward chaining engine is triggered and possibly new assertions are automatically added to an ABox (for details see [MacGregor and Brill, 1992]).

A problem with the LOOM approach is that from a user perspective it is hard to characterize the source of the incompleteness of the LOOM reasoning algorithms (see the discussion in [Horrocks, 1997a, p. 42]). Although the inference techniques used in LOOM are characterized in [MacGregor, 1991b, p. 90], once a system is incomplete, there is no adequate measure for the "quality of service" in terms of an implementation-independent characterization. For instance, in CLASSIC the characterization of the incompleteness of the inference system concerning individual reasoning was given in terms of a weak semantics for the offered representation constructs (see above). It should be noted that specifying the incompleteness on the semantical level is by no means a trivial task. Not only incompleteness issues are important in this context. For instance, the theoretical background for giving a semantics for rule-based computations was only investigated recently [Donini *et al.*, 1992b; 1994a; 1998a].

Incomplete reasoning facilities might lead to unexpected behavior. We demonstrate with an example that incomplete inference algorithms can have effects in situations a user might not be aware of. LOOM also supports closed-world reasoning. The strategy for closing a role for an individual is to count the number of known role fillers. However, in addition to the individuals explicitly mentioned in the ABox, existential quantifications and minimum number restrictions have to be considered. Assuming too few of these individuals might result in an inconsistency. This is demonstrated with a simple knowledge base example with the following ABox $\{(\exists R.A \sqcap \exists R.B \sqcap \exists R.C)(i),\ R(i,j)\}$. Let us assume, in the TBox there exist axioms such that A is *implicitly* declared as disjoint from both concepts, B and C. In the LOOM system, specific reasoning techniques (e.g., a technique called "conditioning" [MacGregor, 1991b]) are implemented to compute the number of necessary fillers. Closing the role R for i by adding $(\leq 1\, R)(i)$ makes the ABox inconsistent. However, since LOOM is incomplete, it might be the case that the disjointness of A and B as well as A and C is not detected and, therefore, too few fillers are assumed to exist in the closing process. Thus, the added maximum number restriction might be too restrictive, i.e., the system is unsound if closed-world reasoning is employed. Note that the semantic basis of automatic closing of roles as offered by LOOM is hard to characterize for expressive representation languages. Obviously, closing the role R for i with $(\leq 2\, R)(i)$ might be a candidate. However, closing the role R for i with $(\leq 3\, R)(i)$ might also be possible. In this case we have more individuals but with less specific constraints.

BACK and FLEX

Research on BACK (Berlin Advanced Computational Knowledge representation system) started in 1985, approximately at the same time as work on the LOOM system was initiated. BACK was also called a knowledge representation environment [Quantz and Kindermann, 1990; Peltason, 1991; Hoppe *et al.*, 1993].

The description logic of the initial BACK system can be called $\mathcal{ALQR}^{-1}$. There was also support for reasoning with numbers and attribute sets. Research on the inference algorithms for the basic BACK language stimulated the development of theoretical results on the complexity of concept consistency reasoning (e.g., [Nebel, 1988; 1990a]) as well as the semantics of cycles [Nebel, 1991]. Additionally, not only terminological reasoning was considered but an investigation was made on the development of a hybrid architecture consisting of a TBox and an ABox. Issues of integration and balancing in hybrid knowledge representation systems, namely balanced expressiveness and tight coupling in hybrid systems, were analyzed in [Nebel and von Luck, 1987; 1988]. Research on the BACK system helped to shape the current view on balanced representation schemes with TBox and ABox. In order to provide an hybrid representation language, BACK was one of the first systems, in which TBox concept terms could also be used in an ABox to assert, e.g., disjunctive information about individuals. In addition, distinct individuals were assumed to denote distinct objects. Hence, the number of role fillers could be counted and compared against number restrictions (this was also done in KRYPTON as pointed out by [Woods and Schmolze, 1990, p. 165]). The algorithms used in BACK for instance checking and instance retrieval are described in [Nebel and von Luck, 1987; 1988; Kindermann and Randi, 1990]. In general, the discussion of the problems of incomplete algorithms that was sketched in the previous section also applies to the BACK system because the inference algorithms used in BACK are also known to be incomplete.

In order to provide a knowledge representation environment, the BACK architecture was designed to support incremental additions to the ABox. BACK was one of the first attempts to implement algorithms for reasoning about retractions of ABox assertions. BACK supported retraction of told information, also called literal retraction [Nebel, 1990a; Kindermann, 1992]. This is also supported in the LOOM system. ABox assertions can be retrieved from a database by automatically computing SQL queries [Schmiedel, 1993]. For the applications considered in the BACK project, reasoning about time was important. Therefore, an integration of temporal reasoning and terminological reasoning was investigated by several project members. Investigations about how to incorporate temporal reasoning into terminological reasoning are reported in [Schmiedel, 1988; 1990; Schild, 1993; Fischer, 1992; Neuwirth, 1993].

In the successor system FLEX [Quantz *et al.*, 1995], incomplete algorithms were implemented for the description logic $\mathcal{ALCQRIFO}$. Additionally, reasoning about equations and inequations concerning integers was supported. Furthermore, the FLEX system served as a testbed for investigating so-called weighted defaults [Quantz and Royer, 1992]. The initial implementation of FLEX was developed in Prolog. FLEX++ was a reimplementation in C++. The implementation was faster, but for application knowledge bases the performance was not sufficient. Appropriate optimization techniques (see the next chapter) had not been investigated in the context of description logics at the time of the development of the FLEX implementation.

In general, it is quite difficult to compare different systems and knowledge representation environments because the services being offered and the representation languages are not standardized (see [Patel-Schneider and Swartout, 1993] for a proposal on standardizing representation languages and inference services). Experiences with system implementations indicated that either limited expressiveness or incompleteness of reasoning could possibly lead to problems in applications. Therefore, other researchers investigated the implementation of systems based on sound and complete algorithms (published at the end of the eighties and beginning of the nineties). One can consider [Schmidt-Schauß and Smolka, 1991] as a starting point of this development (see also [Donini *et al.*, 1991a]). Based on tableaux calculi, practical description logic implementations were developed. We discuss the architectures of the systems KRIS and CRACK.

Kris

The development of sound and complete reasoning systems for more expressive description logics started at the end of the eighties. One of the main developments in this direction was the system Kris. The approach of Kris was to implement sound and complete algorithms for an expressive description logic and to develop optimization techniques for TBox reasoning so that, in practice, reasonable performance could be expected. The description logic of Kris is $\mathcal{ALCNF}$ [Baader and Hollunder, 1991a; 1991b]. As an addition, Kris provides enumerated types ($\mathcal{O}$ operator) and an experimental interface for reasoning about so-called concrete domains [Baader and Hanschke, 1991a; 1991b; 1992] (e.g., linear inequations over the reals). Role conjunctions were supported with a prototype implementation. The focus of the work in the Kris project was on TBox-classification. Nevertheless, Kris was one of the first systems also supporting sound and complete ABox reasoning in expressive description logics. Even multiple ABoxes could be handled. The implementation language of Kris was CommonLisp (see [Hollunder *et al.*, 1991] for a User's Guide and [Achilles *et al.*, 1991] for a description of the graphical user interface).

The idea behind optimizing TBox classification was to exploit "obvious" information concerning "told" superconcepts and primitive concepts. In many concept definitions of application knowledge bases the right-hand side is a conjunction with concept names and concept terms. The conjuncts which are concept names on the right-hand side are defined as the "told" subsumers. Another important point was to avoid recomputation of subsumption relations found in preceding computation steps. Thus, caching and propagation techniques were implemented. The idea was that information can be propagated in the subsumption lattice such that expensive subsumption tests can be avoided where possible. Kris was the first system for which systematic empirical tests were carried out. The algorithms evaluated in [Baader *et al.*, 1992a; 1994] are still in use in modern description logic systems (see below). Extensions such as defaults were investigated as well (see also [Baader and Hollunder, 1992; 1993; Hollunder, 1994a]) but have not been implemented in Kris.

Although the benchmarks considered in [Baader *et al.*, 1994] revealed that the performance of Kris for TBox reasoning was comparable to that of other systems of that time, the more or less direct implementation of *nondeterministic* tableaux algorithms that were developed for proving the decidability of problems in the field of theoretical computer science with chronological backtracking as in Kris led to performance problems for many applications. One of the main results of the Kris project was that sound and complete inference algorithms are an important starting point for research on optimized sound and complete algorithms for practical system development.

Crack

One of the main research goals of the system Crack was to implement sound and complete algorithms for dealing with inferences about individuals in concept terms. Rather than providing a non-standard semantics as in Classic (individuals are mapped onto sets of domain objects), in Crack, individuals are mapped to elements of the domain. Thus, coreferences also have to be considered in concept terms. Crack supports the description logic $\mathcal{ALCRIFO}$ [Bresciani *et al.*, 1995]. The implementation of Crack is based on CommonLisp. Crack provided a web interface.

In a similar way as in Kris, obvious information is exploited in the architecture to some extent but, nevertheless, Crack is a direct implementation of the tableaux rules of the underlying calculus. In the middle of the nineties it became clear that sound and complete reasoning is needed for many applications but the employed inference techniques which had been developed for (manually) deriving decidability results, e.g., with tableaux algorithms, were not suited for direct implementation. Thus, at the beginning of the nineties it became clear that there is a long way to go from a decidability proof to a working system, which has good performance in the average case.

Other systems

The list of systems we have discussed in this chapter is certainly incomplete. The large number of projects involved in the development of knowledge representation systems shows the importance of this area. Usually description logic systems are built around a core engine which is a consistency checker. However, there are other services to be supplied which are also important to make the systems usable in larger application projects. We present an overview of some additional systems with interesting features developed at the beginning of the nineties.

Among other points, the graphical manipulation of representations was investigated in the Sb-One project [Allgayer, 1990; Kobsa, 1991b; 1991a]. The implementation language was CommonLisp. Techniques for graphical interfaces to support knowledge base development with Sb-One are described in [Kalmes, 1988; 1990] (see also [Abrett and Burstein, 1987] for a description of the Kreme system). Furthermore, in Sb-One the use of contexts (also called partitions) was explored for user modeling applications in natural language generation.

Another important point for DL inference systems is persistence and transaction management. We have already discussed the BACK approach [Schmiedel, 1993] (see also [Borgida, 1995]). Additional investigations were also made with the K-REP system [Mays *et al.*, 1991a; 1991b].

Summary: standard inference services of Description Logics systems

Before discussing successors of the second generation systems presented in this section it is appropriate to summarize the main inference problems that are now assumed as standard for DL systems. The inference services provided by DL systems for concept consistency and TBox reasoning can be summarized as follows.

- *Concept consistency* (w.r.t. a TBox)
- *Concept subsumption* (w.r.t. a TBox)
- Another important inference service for practical knowledge representation is to check whether a certain concept name is inconsistent w.r.t. a TBox. Usually, inconsistent concept names are the consequence of modeling errors. Checking the consistency of all concept names mentioned in a TBox without computing the parents and children is called a TBox *coherence check*.
- The problem of computing the most-specific concept names mentioned in a TBox that subsume a certain concept is known as computing the *parents* of a concept. The *children* are the most-general concept names mentioned in a TBox that are subsumed by a certain concept. We use the name *concept ancestors* (*concept descendants*) for the transitive closure of the parents (children) relation. The computation of the parents and children of every concept name is also called *classification* of the TBox. This inference is needed to build a hierarchy of concept names w.r.t. specificity and is known as TBox classification.

If a system supports ABox reasoning, the following inference services are provided:

- *ABox consistency* (w.r.t. a TBox)
- *Instance test* w.r.t. a TBox and an ABox
- The most-specific concept names mentioned in a TBox $\mathcal{T}$ of which an individual is an instance are called the *direct types* of the individual w.r.t. a TBox and an ABox.
- The *retrieval* inference problem is to find all individuals mentioned in an ABox that are an instance of a given concept C w.r.t. a TBox.
- The set of *fillers* of a role R for an individual i w.r.t. a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ is defined as $\{x \,|\, (\mathcal{T}, \mathcal{A}) \models (i, x) : R\}$ where $(\mathcal{T}, \mathcal{A}) \models ax$ means that all models of $\mathcal{T}$ and $\mathcal{A}$ are also models of ax.
- The set of *roles* between two individuals i and j w.r.t. a knowledge base $(\mathcal{T}, \mathcal{A})$ is defined as $\{R \,|\, (\mathcal{T}, \mathcal{A}) \models (i, x) : R\}$.

In many DL systems, there are some auxiliary queries supported: retrieval of the concept names or individuals mentioned in a knowledge base, retrieval of the set of roles, retrieval of the role parents and children (defined analogously to the concept parents and children, see above), retrieval of the set of individuals in the domain and in the range of a role, etc. As we have discussed in this section, DL systems of the second generation offer more or less all or these inference services. An exception is a language for specifying retrieval queries that goes beyond the simple retrieval inference problem mentioned above (see e.g., the discussion about Loom).

8.4 The next generation: FaCT , Dlp and Racer

The declarative nature of description logic modeling is even more important when problems are treated for which languages are required that are no longer tractable. Inspired by theoretical advances, e.g., for handling number restrictions, role conjunctions, generalized concept inclusions as well as cyclic axioms with descriptive semantics ($\mathcal{ALCNR}$ [Buchheit *et al.*, 1993a]), transitive roles ($\mathcal{ALC}_{R^+}$ [Sattler, 1996]), role hierarchies and features ($\mathcal{ALCH}f_{R^+}$ [Horrocks, 1998b]), as well as inverse roles, qualified number restrictions, and role hierarchies ($\mathcal{SHIQ}$ [Horrocks *et al.*, 1999] also called $\mathcal{ALCQHI}_{R^+}$, pronounced ALC-choir), the development of another generation of sound and complete description logic systems was started at the end of the nineties.

FaCT

Initially, research on practical implementations of description logic systems for expressive description logics started with a focus on concept and TBox reasoning. However, rather than directly implementing the tableaux calculus used for the theoretical decidability proofs and complexity analyses, a rigorous investigation into methods for informed search was made for developing the next generation of description logic systems. In particular, average-case optimization techniques have been investigated with the system FaCT ([Horrocks, 1997a; 1998b; Horrocks and Patel-Schneider, 1999] see also the subsequent chapter for details). At the time of this writing, two versions of FaCT are available. One version supports TBox reasoning for the description logic $\mathcal{ALCH}f_{R^+}$ [Horrocks, 1997a; 1998b]. Furthermore, a newer version of FaCT also supports TBox reasoning with inverse roles and qualified number restrictions ($\mathcal{SHIQ}$ [Horrocks, 1999; Horrocks *et al.*, 1999]). At the time of this writing, FaCT does not support ABoxes.

It was the FaCT system that first demonstrated the usefulness of expressive description logics for developing practical applications. It was shown that, although runtime behavior can be exponential in the worst case, in practical contexts, optimization techniques can be found that prevent a DL system from running into combinatorial explosion. Nevertheless, the algorithms are still sound and complete. Indeed, after several years of experiences with less expressive systems such as Classic, research on FaCT stimulated many research activities for developing optimized DL system implementations for expressive description logics.

The system FaCT is implemented in CommonLisp and can be downloaded with source code for research purposes. A CORBA interface guarantees seamless integration into network-aware applications. Various input formats are supported by FaCT (e.g., for XML-based notations of TBoxes). The graphical interface OilEd for developing TBoxes in the spirit of frame systems is described in [Bechhofer *et al.*, 2001b].

Dlp

Based on similar techniques as FaCT, the system Dlp utilizes extended techniques for optimizations [Horrocks and Patel-Schneider, 1998c; 1998d; Patel-Schneider, 1999]. Dlp supports concept consistency reasoning for the description logic $\mathcal{ALCN}_{reg}$. From a modal logic perspective, $\mathcal{ALCN}_{reg}$ can also be called Propositional Dynamic Logic (PDL) with a restricted form of graded modalities, i.e., simple number restrictions.

Dlp has succeeded in many performance competitions [Horrocks, 1998a; Horrocks and Patel-Schneider, 1998c; Patel-Schneider, 1999]. It was shown that tableaux-based approaches can be implemented such that the performance for satisfiability testing for $\mathcal{ALC}$ or modal logic K_m is comparable to traditional approaches used in the community [Giunchiglia and Sebastiani, 1996b; Giunchiglia *et al.*, 1999].

However, in the current version of Dlp TBox classification is not provided as an inference service. In particular, no generalized concept inclusions and no TBoxes with forward references are supported (i.e., algorithms for dealing with generalized concept inclusions are not implemented in Dlp). ABoxes are not supported as well. Dlp is implemented in SML.

Racer

For many applications, besides concept consistency and TBox reasoning, ABox reasoning is also important. Calculi for ABox consistency have been presented for the above-mentioned representation constructs: $\mathcal{ALCNR}$ [Buchheit *et al.*, 1993b], $\mathcal{ALCNH}_{R^+}$ [Haarslev and Möller, 2000], $\mathcal{ALCQHI}_{R^+}$ ($\mathcal{SHIQ}$) [Horrocks *et al.*, 2000c]. Based on theoretical results, a practical implementation of ABox calculi was developed with the full TBox and ABox description logic system Racer [Haarslev and Möller, 1999; 2001e]. Racer supports all optimization techniques that are incorporated into FaCT. Some new optimization techniques investigated with the RACER system (e.g., for dealing with number restrictions and ABoxes) are mentioned in the next chapter. In RACER, the unique name assumption for ABox individuals is imposed. In order to demonstrate the usefulness of DL systems for practical applications, high performance reasoning for large TBoxes is discussed in [Haarslev and Möller, 2001c].

Initial versions of the RACER system supported the logic $\mathcal{ALCNH}_{R^+}$. In later versions reasoning was extended to ABox reasoning with the logic $\mathcal{ALCQHI}_{R^+}$ ($\mathcal{SHIQ}$). In addition, RACER supports concrete domains without so-called feature chains (see [Baader and Hanschke, 1991a] and the discussion of the KRIS system). In particular, predicates representing linear inequalities about the reals are handled by RACER (see [Haarslev *et al.*, 2001; Haarslev and Möller, 2001b] for details).

RACER dynamically selects appropriate optimization techniques due to a static analysis of input TBoxes, ABoxes and queries. As a distinguishing feature, which is important for many applications, it should also be mentioned that RACER supports multiple TBoxes and ABoxes (see also the KRIS system). Assertions can be added to ABoxes after queries have been answered. In addition, for instance, RACER also provides support for retraction of assertions from ABoxes.

RACER can be downloaded for research purposes as a server program for standard operating systems with no additional licenses. A socket-based network version with Java interface is available. The implementation language of RACER is COMMON-LISP.

8.5 Lessons learned

Considering the evolving technology of description logic systems it becomes clear that at the end of the nineties there is an enormous interest in description logic reasoning systems. This is demonstrated by the quite large number of system implementations. Currently, all modern DL systems are based on sound and complete algorithms. Thus, system developers can really rely on all answers computed by a DL system. This positive trend has been initiated by the development of optimization techniques that ensure stable runtimes for average-case inputs for real-world problems even if the worst-case complexity is exponential (see also below). The trend has been initiated by the landmark system FaCT.

The original idea of the tell and ask interface of KRYPTON is still realized in modern systems. However, at the time of this writing, the systems support only some kind of batch-oriented behavior. A knowledge base (TBox and ABox) is passed to the systems (tell interface). Afterwards, queries can be answered (ask interface). But, no incremental additions to the knowledge base are possible after the first query is answered. The difficulty is that complex transformations on the knowledge bases are necessary in order to compute an internal representation that can be used for relatively fast query answering (see the discussion on optimization techniques in subsequent chapters). The price to pay is that algorithms for appropriately handling incremental additions to a knowledge base are not yet known. Other features, e.g., explanation facilities, retraction, etc. still have to be developed for expressive DLs as well.

As a second and quite important lesson one can see that description logics with more expressiveness and sound and complete algorithms impose a different view in modeling. Concept definitions as known from, for instance, Classic are no longer the central modeling device if generalized concept inclusions (representing cyclic implications or equalities) are available.[1]

A third lesson we can learn from considering description logic systems and their development is that the implementation language is hardly important for the magnitude of speed (compared to the expressiveness of the description logic). What really counts is the set of optimization strategies, the implementation of index data structures and the selection of clever heuristics. There are first attempts to provide a distributed implementation of a description logic system. However, performance problems in network communication lead to server-based solutions, i.e., a knowledge base is being processed at a single workstation computer (but may be accessed from different clients). Benchmark generators and standardized application knowledge bases are used for metering system performance. Thus, different system implementations can be compared.

With Racer we have discussed a state-of-the-art description logic system that also supports ABoxes and concrete domains. However, only simple query languages are currently available. For description logics without inverse roles and number restrictions (i.e., $\mathcal{ALCH}f_{R^+}$), [Tessaris, 2001] developed the theoretical basis for supporting so-called conjunctive in DL systems. However, for DLs as expressive as $\mathcal{SHIQ}$ much less is known.

Another lesson is that the development of techniques for practically incorporating facilities for the representation of space and time into description logics is still an open issue. The necessity of a semantics-based integration of temporal and terminological reasoning has been emphasized in first investigations in the Back project. However, early approaches (e.g., [Schmiedel, 1990]) have been shown to be undecidable [Halpern and Shoham, 1991; Schild, 1993]. In the context of planning, the opportunities of an integrated environment combining temporal and terminological reasoning were clearly demonstrated with the RHET system [Allen, 1991]. It has been shown that spatial reasoning (e.g., about topological relations) induces non-obvious subsumption relationships between concepts [Haarslev *et al.*, 1998;

[1] Nevertheless, description logics can still be called object-based representation formalisms, although there are some approaches to deal with n-ary relations [Schmolze, 1989; Calvanese *et al.*, 1998d] as well.

1999]. The work presented in [Artale *et al.*, 2001] demonstrates that the decidability barrier is achieved if temporal operators are integrated into expressive description logics. Nevertheless, [Artale *et al.*, 2001] identify a fragment that allows for a limited kind of practical modeling. Initial experiments concerning an implementation of a description logic that supports operator for linear time temporal reasoning are discussed in [Günsel and Wittmann, 2001].