D W Q

Foundations of **D**ata **W**arehouse **Q**uality

National Technical University of Athens (NTUA)
Informatik V & Lehr- und Forschungsgebiet Theoretische Informatik (RWTH)
Institute National de Recherche en Informatique et en Automatique (INRIA)
Deutsche Forschungszentrum für künstliche Intelligenz (DFKI)
University of Rome «La Sapienza» (Uniroma)
Istituto per la Ricerca Scientifica e Tecnologica (IRST)

F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf

AL-log: Integrating Datalog and Description Logics

Journal of Intelligent Information Systems,Vol. 10, 1998.

DWQ : ESPRIT Long Term Research Project, No 22469
Contact Person : Prof. Yannis Vassiliou, National Technical University of Athens,
15773 Zographou, GREECE Tel +30-1-772-2526 FAX: +30-1-772-2527, e-mail: yv@cs.ntua.gr

Journal of Intelligent Information Systems, , 1–27 ()

$\mathcal{AL}$-log: Integrating Datalog and Description Logics

FRANCESCO M. DONINI, MAURIZIO LENZERINI, DANIELE NARDI AND ANDREA SCHAERF

{donini,lenzerini,nardi,aschaerf}@dis.uniroma1.it

Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza"
Via Salaria 113, I-00198, Roma, Italy

Editor:

Abstract. We present an integrated system for knowledge representation, called $\mathcal{AL}$-log, based on description logics and the deductive database language Datalog. $\mathcal{AL}$-log embodies two subsystems, called structural and relational. The former allows for the definition of structural knowledge about classes of interest (concepts) and membership relation between objects and classes. The latter allows for the definition of relational knowledge about objects described in the structural component. The interaction between the two components is obtained by allowing constraints within Datalog clauses, thus requiring the variables in the clauses to range over the set of instances of a specified concept. We propose a method for query answering in $\mathcal{AL}$-log based on constrained resolution, where the usual deduction procedure defined for Datalog is integrated with a method for reasoning on the structural knowledge.

Keywords: Description Logics, Deductive Databases, Datalog, Object-based Knowledge Representation, Query Answering

1. Introduction

Hybrid systems are a special class of knowledge representation systems which are constituted by two or more subsystems dealing with distinct portions of a knowledge base and specific reasoning procedures (Frisch and Cohn, 1991). The characterizing feature of hybrid systems is that the whole system is in charge of a single knowledge base, thus combining knowledge and reasoning of the different subsystems in order to answer user questions. The motivation for building hybrid systems is to improve on two basic features of knowledge representation formalisms, namely representational adequacy and deductive power.

It is very often the case that a knowledge representation formalism together with the associated reasoning procedures is adequate for representing certain kinds of information, but unsuited or at least heavy to use for others. The possibility of using different formalisms in the same framework is pursued in hybrid systems in order to overcome the limitations of a single representation paradigm.

The improvement in deductive power of hybrid systems is in terms of both the inferences the system is able to make, and the efficiency of the deductive process. With regard to the first point, the set of conclusions a hybrid system should be able to draw is not just the union of the conclusions derivable within each subsystem, because any subsystem can take advantage of the results obtained by other sub-

systems. With regard to the second point, the use of specialized reasoners makes it possible to improve the deduction process, since specialized reasoners typically perform better than general purpose deduction procedures.

Based on the above arguments, several two-component systems has been proposed (see for example, Brachman and Pigman Gilbert, 1985, Nebel and von Luck, 1988, Borgida *et al*, 1989, Baader and Hollunder, 1991), where the subsystems have been called terminological and assertional. The terminological subsystem (sometimes called TBox) provides the ability to create a hierarchical structure of concepts (or classes) by using a so-called description logic, whereas the assertional subsystem (ABox) provides the user with a language for expressing properties of individuals.

Closely related to terminological/assertional systems are the proposals arising from the study of many-sorted logics, where a first order language is combined with a sort language, which can be regarded as an elementary description logic (Cohn, 1989, Frisch, 1989).

In this paper we present a two-component system, named $\mathcal{AL}$-log (from $\mathcal{A}$ttributive $\mathcal{L}$anguage (Schmidt-Schauß and Smolka, 1991) and Datalog), whose main feature is to add to the framework of deductive databases the structuring power of description logics.

More specifically, in $\mathcal{AL}$-log we have a *structural* subsystem based on the description logic $\mathcal{ALC}$ (Donini *et al*, 1991a, Schmidt-Schauß and Smolka, 1991) and a *relational* subsystem based on the database language Datalog (Ullman, 1988). The interaction between the two subsystems is realized by allowing the specification of constraints in Datalog clauses, where constraints are expressed using $\mathcal{ALC}$. Constraints on variables require them to range over the set of instances of a specified concept, while constraints on individual objects require them to belong to the extension of a concept.

The $\mathcal{AL}$-log hybrid reasoner computes answers to queries, based on the specification of both the structural and the relational component of the knowledge base. We propose a resolution-based hybrid reasoner, that finds refutations in the relational component by means of a top-down procedure, and then uses the constraints collected during the first step for a special check on the structural component.

$\mathcal{AL}$-log is innovative in several respects. First, each subsystem embodies a knowledge base with both an intensional and an extensional level. In fact, the structural subsystem by itself is highly expressive compared with usual terminological systems, since the knowledge base of concepts can be defined by specifying containment assertions between concepts and membership assertions on individual objects.

Secondly, a comparison of $\mathcal{AL}$-log with the classification of hybrid systems made in (Frisch and Cohn, 1991) reveals that $\mathcal{AL}$-log embodies features of both the substitutional framework (Frisch, 1989) and theory resolution (Stickel, 1985). With regard to the former, the structural subsystem can be seen as a background theory of the Datalog clauses, that selects among the ground instances of the clauses those instances satisfying the associated constraints. The similarity with theory resolution is in that refutations on the whole knowledge base can be found by proving the validity (w.r.t. the background theory) of certain disjunctive formulas whose disjuncts are taken from different clauses. In this respect, $\mathcal{AL}$-log follows the ap-

proach proposed in (Baader *et al*, 1990), specializing it to the case where the logical language is a deductive database language, and exploiting the use of the description logic in the definition of the structural component of the knowledge base.

Third, the resulting system provides a form of integration between objects and logic within the framework of databases. Recently, there has been much interest in investigating database systems providing both deductive and object-oriented capabilities, see for example (Aït-Kaci and Nasr, 1986, Abiteboul and Kanellakis, 1989, Abiteboul, 1990, Kifer *et al*, 1995, Ling *et al*, 1995). Compared with most of the proposals of this kind, $\mathcal{AL}$-log is more expressive in the structural component, and provides more sophisticated reasoning mechanisms on such component. Indeed, one of the main features of $\mathcal{AL}$-log is the representation of, and the reasoning on incomplete knowledge in the structural component. Representing incomplete knowledge is made possible by the rich set of constructs used for building concepts (named classes in the usual object-oriented terminology) expressions. For example, one can state that the concept $Person$ is a subset of the concept $Male \sqcup Female$, where the latter denotes the set of objects that are instances of either $Male$ or $Female$. Such an intensional assertion can be seen as an integrity constraint in most of the approaches to object-oriented databases. This means that, if a database state has one instance p of $Person$, such a state will be accepted if there is also the explicit assertion that p is an instance of $Male$ or the explicit assertion that p is an instance of $Female$. On the contrary, in our system, the above assertion can be seen as the specification of the possible models of the knowledge base, and not just an integrity constraint. This means that we accept a knowledge base with the assertion that p is an instance of $Person$, without the explicit information about whether it is an instance of $Male$ or $Female$, and the system can reason and answer queries on the basis of such an incomplete information. This feature is also shared by several recent papers on integrating Horn rules and Description Logics, which are actually based on the framework proposed in this paper, see (Levy *et al*, 1996, Levy and Rousset, 1996).

We note that such a high expressive power in the structural component forces us to stick with Horn clauses in the relational component. Indeed, $\mathcal{AL}$-log does not include negation in the relational subsystem. Moreover, since relational knowledge does not play any role in deducing new information about the structure of individual objects, the form of integration between the two subsystems is weaker than the one supported in most deductive object-oriented database languages. In this sense, our system can be seen as a special constraint logic programming system (Van Hentenryck, 1989), where Datalog is used as the logic programming language, and assertions about the structural component constitute the specification of the constraints. We should note, however, that the possibility of expressing incomplete information in the constraint part is not common in constraint logic programming. We show in the rest of this paper, that indeed this characteristic poses new problems for the task of devising a query answering method for $\mathcal{AL}$-log.

The paper is organized as follows. In Section 2 we describe $\mathcal{AL}$-log, by presenting the main features of the structural and the relational subsystems, and defining the semantics of a knowledge base in our system. In Section 3, we show one example

of knowledge base expressed in $\mathcal{AL}$-log and briefly discuss its expressive power in formulating queries. In Section 4 we present a method for hybrid reasoning in $\mathcal{AL}$-log. Finally, conclusions are drawn in Section 4.

2. The System $\mathcal{AL}$-log

As we mentioned in the introduction, $\mathcal{AL}$-log embodies two subsystems, called structural and relational. The former, described in Subsection 2.1, allows one to express knowledge about concepts, roles and individuals. The latter, described in Subsection 2.2, provides the user with a suitable extension of Datalog in order to express relational knowledge. The characterization of both the semantics of a knowledge base, and the set of inferences that can be carried out over it, is provided in Subsection 2.3. Finally, examples of $\mathcal{AL}$-log knowledge bases are discussed in Subsection 2.4.

2.1. The Structural Subsystem

The structural subsystem of $\mathcal{AL}$-log allows one to define what we call an $\mathcal{ALC}$-knowledge base. An $\mathcal{ALC}$-knowledge base is structured into two levels, intensional and extensional. The intensional level refers to the knowledge about the concepts of interest, and is constituted by a set of statements specifying the relevant properties of such concepts. The extensional level regards the knowledge about individual objects, and is constituted by a set of membership assertions between objects and concepts (e.g. a is an instance of C), and between pairs of objects and roles (e.g. a is related to b by the role R).

Therefore, the structural subsystem of $\mathcal{AL}$-log is itself a two-component system, where one component constitutes the intensional level, and the other constitutes the extensional level. We show later that the expressive power of the language constructs used in the structural subsystem is greater than in conventional terminological/assertional hybrid systems.

Both at the intensional and the extensional level, there is the need of a specific description logic for building concept expressions. In $\mathcal{AL}$-log, the description logic is $\mathcal{ALC}$ (Donini *et al*, 1991a, Schmidt-Schauß and Smolka, 1991). Concepts represent classes of objects in the domain of interest, while roles represent binary relations between concepts. Complex concepts can be defined by means of suitable constructs applied to primitive concepts (concepts denoted simply by a name) and roles. In particular, a concept in $\mathcal{ALC}$ can be formed by means of the following syntax (A denotes a primitive concept, R denotes a role, C and D denote arbitrary concepts):

$$
\begin{array}{rcll}
C, D & \longrightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(top)} \\
 & & \bot \mid & \text{(bottom)} \\
 & & C \sqcap D \mid & \text{(conjunction)} \\
 & & C \sqcup D \mid & \text{(disjunction)} \\
 & & \neg C \mid & \text{(negation)} \\
 & & \forall R.C \mid & \text{(universal quantification)} \\
 & & \exists R.C & \text{(existential quantification)}
\end{array}
$$

As an example, consider the following two concepts

$$FacultyMember \sqcap \forall Teaches.AdvancedCourses$$
$$Person \sqcap \exists Teaches.Course \sqcap \neg FullProfessor$$

The first one denotes faculty members that teach only advanced courses. The second one denotes persons that teach some courses and are not full professors.

Concepts are interpreted as subsets of a domain and roles are interpreted as binary relations over a domain. More precisely, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following equations are satisfied:

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

An interpretation $\mathcal{I}$ is a *model* for a concept C if $C^{\mathcal{I}}$ is nonempty. A concept is *satisfiable* if it has a model and *unsatisfiable* otherwise. We say that C is *subsumed* by D if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$, and C is *equivalent* to D if C and D are subsumed by each other.

We now turn our attention to the problem of defining the intensional level of an $\mathcal{ALC}$-knowledge base. As we said before, the intensional level specifies properties of the concepts of interest in a particular application. Syntactically, such properties are expressed in terms of a set of so-called *inclusion statements* (see (Nebel, 1990, Buchheit *et al*, 1993)). An inclusion statement (or simply inclusion) has the form

$$C \sqsubseteq D$$

where C and D are two arbitrary concepts. Intuitively, the statement specifies that every instance of C is also an instance of D. More precisely, an interpretation $\mathcal{I}$ satisfies the inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

For example, in the knowledge base K_{univ} defined in section 2.4 we have the following two inclusions

$$FullProfessor \sqsubseteq FacultyMember$$
$$NonTeachingFullProfessor = FullProfessor \sqcap \neg\exists Teaches.Course$$

where $C = D$ is a shorthand for the pair of inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$.

The first inclusion states that full professors are also faculty members, whereas the second statement defines the concept $NonTeachingFullProfessor$ as the set of full professors that do not teach courses.

Let $\mathcal{T}$ be a finite set of inclusions. An interpretation $\mathcal{I}$ is a *model* for $\mathcal{T}$ if $\mathcal{I}$ satisfies all inclusions in $\mathcal{T}$. Moreover $\mathcal{T}$ *logically implies* an inclusion statement γ, written $\mathcal{T} \models \gamma$, if γ is satisfied by all the models of $\mathcal{T}$.

Terminological systems usually provide the user with mechanisms for *concept definitions* of the form (Nebel, 1990):

$$A \sqsubseteq D \ (\textit{inclusion}), \text{ or}$$
$$A \doteq D \ (\textit{equivalence}).$$

In these systems, the definitional mechanisms are limited by the requirements that the left-hand side concept A be a primitive concept, and that at most one statement for each concept is allowed.

Moreover, most of the existing systems do not allow cycles in the terminological component to occur, i.e. do not permit a primitive concept A to occur neither directly nor indirectly within the definition of A. On the other hand, we do not impose any constraint on the form of inclusions, and therefore terminological cycles may occur in our system.

As shown in (Nebel, 1990), there are at least three types of semantics for terminological cycles, namely the least fixed point, the greatest fixed point, and the descriptive semantics. Among them, the latter is the most general, and is the one used in our system.

The language proposed here for defining the intensional level of a structural component is more powerful, because there is no limitation to the form of the concepts involved in the inclusions. In particular, notice that an equivalence of the form $A \doteq D$ can be easily expressed in our system using the pair of inclusions $A \sqsubseteq D$ and $D \sqsubseteq A$. On the contrary, an inclusion of the form $C \sqsubseteq D$, where C and D are arbitrary concepts, cannot be expressed with the restrictions imposed by other systems.

We now turn our attention to the mechanisms offered by the structural subsystem of $\mathcal{AL}$-log for the definition of the extensional level. As we said before, such a subsystem uses a rather limited mechanism, that essentially allows one to specify the instance-of relation between objects and concepts, and between pairs of objects and roles.

Let $\mathcal{O}$ be an alphabet of symbols, called *individuals*. Syntactically, instance-of relationships are expressed in terms of *membership assertions* of the form:

$$a\colon C,$$
$$aRb$$

where a and b are individuals, C is a concept, and R is a role. Intuitively, the first form states that a is an instance of C, whereas the second form states that a is related to b by means of the role R.

For example, we can state the following assertions (see the knowledge base K_{univ} in Section 2.4):

$$john\colon FullProfessor, \quad john\ Teaches\ ai$$
$$ai\colon AdvancedCourse, \quad mary\colon FullProfessor \sqcap \forall Teaches.AdvancedCourse$$

In order to assign a precise meaning to membership assertions, the extension function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ is extended to individuals by mapping them to elements of $\Delta^{\mathcal{I}}$ in such a way that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$. Notice that this restriction ensures that different individuals denote different objects in the world (unique name assumption).

An interpretation $\mathcal{I}$ *satisfies* the assertion $a\colon C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and satisfies aRb if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. $\mathcal{I}$ is a model for a set of membership assertions $\mathcal{A}$ if $\mathcal{I}$ satisfies all the assertions in $\mathcal{A}$.

We can now summarize the features of the structural subsystem of $\mathcal{AL}$-log as follows. The structural subsystem allows one to define an $\mathcal{ALC}$-knowledge base Σ, that is a pair

$$\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$$

where

- $\mathcal{T}$ (the intensional level) is a set of inclusions,
- $\mathcal{A}$ (the extensional level) is a set of assertions.

An interpretation $\mathcal{I}$ is a model of Σ if it is both a model of $\mathcal{T}$ and a model of $\mathcal{A}$. Notice that by virtue of the unique name assumption, we can focus on what we call $\mathcal{O}$-interpretations. An $\mathcal{O}$-interpretation for Σ is an interpretation for Σ such that $\mathcal{O} \subseteq \Delta^{\mathcal{I}}$ and for each $a \in \mathcal{O}$, $a^{\mathcal{I}} = a$. An $\mathcal{O}$-model is an $\mathcal{O}$-interpretation that is a model. From this point on, when we refer to interpretations (models) for an $\mathcal{ALC}$-knowledge base, we implicitly assume to deal with $\mathcal{O}$-interpretations ($\mathcal{O}$-models).

Σ is *satisfiable* if it has a model. Furthermore Σ *logically implies* , (written $\Sigma \models$,), where , is either an inclusion or a membership assertion, if every model of Σ satisfies , .

2.2. The Relational Subsystem

The relational subsystem allows one to express relational knowledge in terms of a set of so-called constrained Datalog clauses. Before dealing with them, we give a short summary of Datalog, in order to make the paper self-contained.

2.2.1. Datalog Datalog is a database query language based on the logic programming paradigm (see (Ceri *et al*, 1990, Ullman, 1988)). The interest in Datalog as a database query language stems from the fact that it is a purely declarative language, and it allows the definition of classes of queries which are not definable in other declarative query languages (e.g. the relational calculus).

A Datalog program consists of a set of definite clauses, each one of the form:

$$\alpha_0 \vee \neg\alpha_1 \vee \cdots \vee \neg\alpha_n$$

where $n \geq 0$ and each α_i is an atom of the form $p_i(t_1, \ldots, t_{k_i})$ such that p_i is a predicate symbol and each t_j is either a constant or a variable[1]. Constant, predicate and variable symbols belong to three mutually disjoint alphabets.

Note that each clause of a program has exactly one positive literal, and moreover, it is assumed that all the variables appearing in the clause are universally quantified. It is also required that all atoms with the same predicate symbol have the same number of arguments. Atoms and clauses containing no variables are called ground. A *substitution* θ is a finite set of the form $\{X_1/t_1, \ldots, X_n/t_n\}$, where X_i is a variable, t_i is a term, and $X_i \neq X_j$ for $i \neq j$. A *ground substitution* is a substitution where t_i is a constant for every $i \in \{1, \ldots, n\}$. The result of the *application* of a substitution θ to a clause γ, denoted by $\gamma\theta$, is the clause γ' obtained from γ by simultaneously replacing each occurrence of X_i with t_i.

A clause is often written in the form

$$\alpha_0 :- \alpha_1, \ldots, \alpha_n$$

where the left hand side, called the *head*, is constituted by the positive literal of the clause, and the right hand side, called the *body*, is constituted by the list of the atoms appearing in negative form in the clause. A *fact* is a clause where $n = 0$, and a *rule* is a clause where $n > 0$. We use the notation that constant and predicate symbols are strings beginning with a lower case letter, whereas variables are strings beginning with an upper case letter.

The predicates occurring in a Datalog program D are partitioned into two sets: the extensional predicates (EDB-predicates) and the intensional predicates (IDB-predicates). Intuitively, each EDB-predicate r corresponds to exactly one relation R of the database in such a way that each fact $r(c_1, \ldots, c_k)$ corresponds to the tuple $\langle c_1, \ldots, c_k \rangle$ of R. On the other hand, intensional predicates correspond to virtual relations, or views. It is required that the head predicate of each rule in D be an IDB-predicate.

Based on the above distinction between extensional and intensional predicates, a Datalog program D can be divided into two parts, called extensional and intensional. The extensional part, denoted as $EDB(D)$, is the set of facts D involving the extensional predicates, whereas the intensional part $IDB(D)$ is the set of all other clauses of D.

Any Datalog program D must satisfy the following *safety condition*: Each variable occurring in the head of a clause must also occur in the body of the same clause. It follows from this condition that each fact of D is ground. The safety condition guarantees that the set of all facts derivable from a Datalog program is finite.

The semantics of a Datalog program is based on the notion of Herbrand interpretation.

Let D be a Datalog program. The Herbrand base HB of D is the set of all atoms of the form $p(c_1, \ldots, c_n)$ such that p is a predicate of D and all the c_i are constants of D. We write EHB (resp. IHB) to denote the atoms of HB whose predicates are extensional (resp. intensional).

An Herbrand interpretation for D is a subset of the Herbrand base HB. Let H be an Herbrand interpretation for D. A positive ground literal l is satisfied by H if $l \in H$. A negative ground literal $\neg l$ is satisfied by H if $l \notin H$. An Herbrand interpretation H for D is said to be a model of D if for every clause γ of D, for every ground instance γ' of γ, at least one of the literals of γ' is satisfied by H. The meaning of a Datalog program D is the set of its models. The intersection of all the models of D is itself a model of D, and in particular is the so-called least Herbrand model, i.e. it is a subset of each Herbrand model of D. A list of ground atoms $g = g_1, \ldots, g_n$ (interpreted as a the conjunction of the atoms) is said to be a logical consequence of D, written $D \models g$ if each g_i is satisfied by all the models of D.

Note that an Herbrand interpretation H for D can also be seen as a mapping $\cdot^H$ such that for each $a \in N$, where N is the set of constants occurring in D, $a^H = a$, and for each n-ary predicate p occurring in D, p^H is a subset of N^n. This reformulation of an Herbrand interpretation will be used in the next section for the definition of the semantics of a knowledge base in $\mathcal{AL}$-log.

Let $cons(D)$ denote the set of all ground atoms which are logical consequences of D (i.e. are satisfied in all the models of D):

$$cons(D) = \{F \in HB \mid D \models F\}.$$

It is well known that $cons(D)$ is equal to the least Herbrand model. This means that D logically implies a ground atom α just in case the least Herbrand model of D satisfies α.

A query for a Datalog program D is a conjunction of atoms written as:

$$g_1, \ldots, g_n$$

where each g_i is an atom.

An *answer* to a query g is a substitution θ for the variables of g. The answer θ is *correct* with respect to the Datalog program D if $D \models g\theta$. The *answer* set to a query g is the set of answers to g that are correct with respect to D, i.e. $\{\theta \mid D \models g\theta\}$ and such that $g\theta$ is ground. In other words, the answer set to a query g is the set of all ground instances of g which are logical consequences of D.

2.2.2. Constrained Datalog Programs A constraint Datalog program is defined in an way analogous to a constraint logic program in constraint logic programming languages (Van Hentenryck, 1989). Specifically, a constrained clause is a clause with a set of restrictions on its variables and constants, expressed using the language $\mathcal{ALC}$.

More precisely, a constrained Datalog clause has the form:

$$\gamma \ \& \ \beta_1, \ldots, \beta_m$$

where $m \geq 0$, γ is a Datalog clause, and each β_i is a *constraint* of the form

$$s : C$$

where s is either a constant or a variable, and C is an $\mathcal{ALC}$-concept (the system CARIN allows also for constraints of the form sRt).

For example, the following constrained Datalog clause

$$curr(X, Z) \text{ :- } exam(X, Y), subject(Y, Z) \ \& \ X : Student, \ Y : Course, \ Z : Topic$$

states that for all X and Z it holds that X has Z in its curriculum if there exists Y such that X has passed the exam of Y and Z is a topic of Y, subject to the constraints that X is a student, Y is a topic, and Z is a course.

Intuitively, a constraint on a variable V restricts the values of V to range over the set of instances of the specified concept. Similarly, a constraint on a constant a, called *ground constraint*, imposes the condition that a denotes an individual which is an instance of the specified concept.

A set of constrained Datalog clauses is called a constrained Datalog program.

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

In $\mathcal{AL}$-log, a knowledge base K is defined as a pair

$$K = \langle \Sigma, \Pi \rangle$$

where

- Σ is an $\mathcal{ALC}$-knowledge base
- Π is a constrained Datalog program.

In order for a knowledge base to be acceptable, it must satisfy the following conditions:

1. The set of Datalog predicate symbols appearing in Π is disjoint from the set of concept and role symbols appearing in Σ.
2. The alphabet of constants used in Π coincides with the alphabet $\mathcal{O}$ of the individuals of Σ. Moreover, every constant occurring in Π appears also in Σ. Note that this is not a real limitation, because for each individual a not appearing in Σ one can add to Σ the assertion $a : \top$, resulting in an $\mathcal{ALC}$-knowledge base which is equivalent to Σ.
3. For every clause $\gamma \ \& \ \beta_1, \ldots, \beta_m$ in Π, every variable appearing in $\beta_1, \ldots, \beta_m$ appears also in γ.

The role played by the constraints in the Datalog clauses is critical for the system. In particular, the interaction between the structural and the relational component of a knowledge base $K = \langle \Sigma, \Pi \rangle$ is determined by the constraints specified in the clauses of Π. It is easy to see the similarity with many-sorted logics, where constants and variables are sorted. However, the language used in the structural subsystem of $\mathcal{AL}$-log is much more powerful than the ordinary languages for sorts.

With regard to the semantics of a knowledge base $K = \langle \Sigma, \Pi \rangle$, we define an interpretation $\mathcal{J}$ for K as the union of an $\mathcal{O}$-interpretation $\mathcal{I}$ for Σ and an Herbrand interpretation H for Π_D, where Π_D is the set of Datalog clauses obtained from the clauses of Π by deleting in each clause its constraint part (i.e. the symbol $\&$ together with every constraints).

A ground instance of a constrained clause of K is the constrained clause obtained by substituting each variable in the clause with an individual occurring in Σ. $\mathcal{J} = \langle \mathcal{I}, H \rangle$ is called a model of K if it is a model of Σ (i.e. $\mathcal{I}$ is a model of Σ), and for each clause $\gamma \ \& \ \beta_1, \ldots, \beta_m$, for each of its ground instances $\gamma' \ \& \ \beta'_1, \ldots, \beta'_m$, either there exists one $i \in \{1, \ldots, m\}$ such that β'_i is not satisfied by $\mathcal{J}$, or γ' is satisfied by $\mathcal{J}$.

K logically implies a ground atom α (resp. a ground constraint β), written $K \models \alpha$ (resp. $K \models \beta$), if every model of K satisfies α (resp. β).

A conjunction of ground atoms and ground constraints $\alpha_1, \ldots, \alpha_n \ \& \ \beta_1, \ldots, \beta_m$ is a logical consequence of K, written as $K \models \alpha_1, \ldots, \alpha_n \ \& \ \beta_1, \ldots, \beta_m$, if for each $i \in \{1, \ldots, n\}$, $K \models \alpha_i$ and for each $j \in \{1, \ldots, m\}$, $K \models \beta_j$.

A query Q to a knowledge base K is a sentence of the form:

$$q_1, \ldots, q_n \ \& \ \beta_1, \ldots, \beta_m$$

where $n \geq 0, m \geq 0, n + m > 0$, each q_i is an atom and each β_j is a constraint.

An *answer* to the query Q is a substitution θ for the variables of Q. The answer θ is *correct* with respect to the knowledge base K if $K \models Q\theta$. The *answer set* to a query Q in a knowledge base K is the set of answers to Q that are correct with respect to K, i.e. $\{\theta \mid K \models Q\theta\}$ and such that $Q\theta$ is ground.

It is easy to verify that if a ground query Q contains only constraints (i.e. has the form $\ \& \ \beta_1, \ldots, \beta_m$), then $K \models Q$ if and only if $\Sigma \models Q$. In other words, the knowledge expressed in the relational component does not play any role in deducing facts about the structural component.

2.4. Examples of $\mathcal{AL}$-log Knowledge Bases

Let K_{univ} be the knowledge base defined in Figure 1, where the two columns refer to the structural component and the relational component respectively, and the two rows refer to the intensional level and the extensional level.

The symbols in the structural component should be read as follows: FP = Full Professor, NFP = Non-teaching Full Professor, FM = Faculty Member, TC = Teaching, St = Student, Tp = Topic, Co = Course, AC = Advanced Course, BC = Basic Course.

Structural	Relational
$FP \sqsubseteq FM$ $NFP = FP \sqcap \neg\exists TC.Co$ $AC \sqcup BC = Co$ $AC \sqcap BC \sqsubseteq \bot$	$curr(X,Z) :-$ $exam(X,Y), subject(Y,Z)$ $\&\ X\colon St,\ Y\colon Co,\ Z\colon Tp$ $mayDoThesis(X,Y) :-$ $curr(X,Z), expert(Y,Z)$ $\&\ X\colon St,\ Z\colon Tp,\ Y\colon FM \sqcap \exists TC.AC$ $mayDoThesis(X,Y) :-$ $\&\ X\colon St,\ Y\colon NFP$
$john\colon FP,\ johnTCai,$ $mary\colon FP \sqcap \forall TC.AC,$ $paul\colon St,\ ai\colon AC,$ $kr\colon Tp,\ lp\colon Tp$	$exam(paul, ai),$ $subject(ai, kr),\ subject(ai, lp),$ $expert(john, kr),\ expert(mary, lp)$

Figure 1. The knowledge base K_{univ}

The intensional part of the structural component of K_{univ} specifies the following properties: 1) full professors are faculty members, 2) non-teaching full professors are defined as full professors that do not teach any course, 3) the set of courses is partitioned into basic courses and advanced courses.

The predicates in the intensional part of the relational component have the following intended meaning:
$mayDoThesis(X,Y)$: student X may do the thesis with professor Y;
$curr(X,Y)$: student X has studied topic Y in his/her curriculum;
$expert(X,Y)$: professor X is an expert in topic Y;
$exam(X,Y)$: student X has passed the exam Y;
$subject(X,Y)$: topic Y is treated in course X.

The clauses in the relational component specify the conditions for a student X to do the thesis with a professor Y.

Notice that in all the models of K_{univ}, $paul$ is a student having kr in his curriculum, and $john$ is a faculty member that teaches one advanced course, and is expert in kr. Therefore, looking at the first clause in the relational component, it is easy to see that $K_{univ} \models mayDoThesis(paul, john)$.

Less obviously, K_{univ} logically implies $mayDoThesis(paul, mary)$. Indeed, by looking at the first clause, one realizes that $mayDoThesis(paul, mary)$ is true in the set S_1 of models of K_{univ} in which $mary$ teaches at least one advanced course. On the other hand, it follows from the second clause that $mayDoThesis(paul, mary)$ is true in the set S_2 of models in which $mary$ is a non-teaching full professor. None of the two clauses alone is sufficient to prove that $K_{univ} \models mayDoThesis(paul, mary)$, but it is possible to see that $S_1 \cup S_2$ represent all the models of K_{univ}, and there-

Structural	Relational
$Re \sqcup Bl \sqcup Ye = \top$ $Re \sqsubseteq \neg Bl \sqcap \neg Ye$ $Bl \sqsubseteq \neg Ye$	$not3col \leftarrow$ $\quad arc(X,Y), sameColor(X,Y)$ $sameColor(X,Y) \leftarrow \& \; X:Re, \; Y:Re$ $sameColor(X,Y) \leftarrow \& \; X:Bl, \; Y:Bl$ $sameColor(X,Y) \leftarrow \& \; X:Ye, \; Y:Ye$
	$arc(a,b), \; arc(a,c),$ $arc(c,d), \; arc(c,e),$ $\ldots$ $\ldots$

Figure 2. The knowledge base K_{3col}

fore $mayDoThesis(paul, mary)$ is a logical consequence of K_{univ}. In order to see that every model of K_{univ} is in $S_1 \cup S_2$, consider any model M of K_{univ} such that $M \notin S_2$. In M, *mary* is not a non-teaching full professor, and therefore, since she is a full professor, she teaches at least one course, that must be an advanced course, and therefore $M \in S_1$.

We now present an example (taken from (Cadoli *et al*, 1996)) that highlights the expressivity of $\mathcal{AL}$-log. In particular, it shows that $\mathcal{AL}$-log can express queries that cannot be expressed in pure Datalog.

Graph 3-colorability is the problem of checking if all nodes of a given (undirected) graph can be associated with one out of three colors in such a way that no two adjacent nodes are colored with the same color. It is well-known that Graph 3-colorability is an NP-complete problem. As a consequence (see e.g. (Ullman, 1988)), it is not possible to write a Datalog query of polynomial size w.r.t. the size of the graph that checks for 3-colorability of a graph (represented by EDB-predicates).

Conversely, the 3-colorability of a graph can be checked by means of an $\mathcal{AL}$-log query. Specifically, the ground query *not3col* is logically implied by the knowledge base K_{3col} in Figure 2 if and only if the graph represented by the predicate *arc* is not 3-colorable. In K_{3col}, the graph is represented by the predicate *arc* and the three colors *red*, *blue*, and *yellow* are represented by the concepts *Re*, *Bl*, and *Ye*, respectively.

In addition, in case *not3col* is not logically implied by K_{3col}, the method described in Section 4 singles out a model of the knowledge base that directly provides a solution of the 3-colorability problem.

It is worth remarking that the size of all components, but the extensional relational one, is independent of the size of the graph. In fact, the graph is entirely represented by an EDB-predicate *arc*.

We do not discuss in details the expressivity of $\mathcal{AL}$-log, for which we refer to (Cadoli *et al*, 1996), where the expressivity of $\mathcal{AL}$-log and similar languages is formally addressed.

3. Deduction in the Structural Component

As we said in Section 2.1, the fundamental deduction to be performed in the structural component Σ of an $\mathcal{AL}$-log knowledge base consists of checking whether Σ logically implies , , where , is either an inclusion or a membership assertion. Obviously a yes answer to this problem is significant only if Σ itself is satisfiable. Since the problem of logical implication can be reformulated as an unsatisfiability problem, we only need to check the satisfiability of an $\mathcal{ALC}$-knowledge base Σ. Such a problem was first shown to be decidable in (Donini *et al*, 1991b), and the result was proved in a more general setting in (Buchheit *et al*, 1993). We reformulate here results from both (Donini *et al*, 1991b) and (Buchheit *et al*, 1993), and show that checking the satisfiability of an $\mathcal{ALC}$-knowledge base is a problem in NEXPTIME (i.e., it can be solved by a nondeterministic algorithm using exponential time).

Our method, which takes into account both the intensional and the extensional level, is an adaptation of first-order tableaux[2] calculus (Fitting, 1990) to description logics, which tries to build a model of an $\mathcal{ALC}$-knowledge base Σ. The correctness of the method is established in Theorem 1 by proving (in part 1) that the attempt to build the model preserves the satisfiability of Σ and (in part 2) by showing that if no contradiction shows up during the construction, then from the tableau branch resulting from the calculus a model of Σ can be constructed. Finally, termination is proved in Theorem 3 by showing that each tableau branch has a finite (exponential) size, and there are only finitely many tableau branches.

We introduce an alphabet of variable symbols $\mathcal{V}$ together with a well-founded total ordering '$\prec$' on $\mathcal{V}$. $\mathcal{V}$ is disjoint from other alphabets defined so far. The elements of $\mathcal{V}$ are denoted by the lowercase letters x, y, z, and they are not to be confused with the variables appearing in the Datalog clauses (denoted by uppercase letters). Variables account for individuals whose existence is required by some assertions, but which are not explicitly present in the knowledge base.

In the rest of the section we use the term *object* as an abstraction for individual and variable (i.e., an object is an element of $\mathcal{O} \cup \mathcal{V}$). Objects are denoted by the symbols s, t and, as in Section 2, individuals are denoted by a, b.

For the calculus, we use also assertions involving variables, i.e., an assertion is now a syntactic entity of one of the forms $s : C, sRt$, where C is a concept, R is a role and s, t are objects (either variables or individuals). Given an $\mathcal{O}$-interpretation $\mathcal{I}$, we interpret variables using an *$\mathcal{I}$-assignment* α, which is defined as a function that maps every variable in $\mathcal{V}$ to an element of $\Delta^{\mathcal{I}}$ (not necessarily injectively), and every individual to itself (i.e. $\alpha(a) = a^{\mathcal{I}}$ for $a \in \mathcal{O}$).

An assertion of the form $s : C$ is satisfied by the pair $(\mathcal{I}, \alpha)$ if $\alpha(s) \in C^{\mathcal{I}}$. An assertion of the form sRt is satisfied by $(\mathcal{I}, \alpha)$ if $(\alpha(s), \alpha(t)) \in R^{\mathcal{I}}$.

A *tableau branch* is a set of assertions (possibly involving variables) and inclusions. A tableau branch S is *satisfiable* if there is an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment

α such that $(\mathcal{I}, \alpha)$ satisfies every assertion and iclusion in S. Obviously, $\Sigma \models a\colon C$ if and only if the tableau branch $\Sigma \cup \{a\colon \neg C\}$ is unsatisfiable and $\Sigma \models C \sqsubseteq D$ if and only if $\Sigma \cup \{x\colon C \sqcap \neg D\}$ is unsatisfiable (where x is a variable not appearing in Σ).

In order to check a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ for satisfiability, our technique starts with a tableau branch $S = \mathcal{T} \cup \mathcal{A}$, and adds assertions to S until either a contradiction is generated or an interpretation satisfying S can be easily obtained from it. Assertions are added on the basis of a suitable set of so-called *propagation rules*.

To avoid the introduction of an infinite number of variables, we filter them through an equivalence criterion: Given a tableau branch S and two variables x and y appearing in S, we say that x and y are S-*equivalent*, written $x \equiv_S y$, if they are involved in assertions with the same concepts, i.e., $\{C \mid x\colon C \in S\} = \{C \mid y\colon C \in S\}$.

To reduce the number of rules, we assume concepts to be in negation normal form (NNF). Any $\mathcal{ALC}$ concept can be rewritten into an equivalent NNF concept by exploiting the well-known logical equivalences: $\neg\neg C \equiv C$, $\neg(C \sqcup D) \equiv (\neg C) \sqcap (\neg D)$, $\neg(C \sqcap D) \equiv (\neg C) \sqcup (\neg D)$, $\neg\exists R.C \equiv \forall R.(\neg C)$, $\neg\forall R.C \equiv \exists R.(\neg C)$.

The *propagation rules* are:

- $S \rightarrow_{\sqcap} \{s\colon C_1,\ s\colon C_2\} \cup S$
  if 1. $s\colon C_1 \sqcap C_2$ is in S,
  2. $s\colon C_1$ and $s\colon C_2$ are not both in S

- $S \rightarrow_{\sqcup} \{s\colon D\} \cup S$
  if 1. $s\colon C_1 \sqcup C_2$ is in S,
  2. $D = C_1$ or $D = C_2$,
  3. neither $s\colon C_1$ nor $s\colon C_2$ is in S

- $S \rightarrow_{\exists} \{sRx,\ x\colon C\} \cup S$
  if 1. $s\colon \exists R.C$ is in S,
  2. for every variable y in S, $y \prec x$ (hence x is not in S)
  3. if s is a variable, there is no variable y in S such that $y \prec s$ and $s \equiv_S y$
  4. there is no t such that both sRt and $t\colon C$ are in S

- $S \rightarrow_{\forall} \{t\colon C\} \cup S$
  if 1. $s\colon \forall R.C$ is in S,
  2. sRt is in S,
  3. $t\colon C$ is not in S

- $S \rightarrow_{\sqsubseteq} \{s\colon C' \sqcup D\} \cup S$
  if 1. $C \sqsubseteq D$ is in S,
  2. s appears in S,
  3. C' is the NNF concept equivalent to $\neg C$,
  4. $s\colon \neg C \sqcup D$ is not in S

- $S \rightarrow_{\bot} \{s : \bot\}$

  if 1. $s : A$ and $s : \neg A$ are in S, or
  2. $s : \neg\top$ is in S
  3. $s : \bot$ is not in S

The last condition of application of each rule ensures that a rule application is never repeated twice, and is often implicitly assumed in tableaux rules. Notice that the $\rightarrow_{\exists}$-rule requires the introduction of another variable in the tableau branch, and this could pose problems for the termination of a procedure that repeatedly applies the rules, due to the presence of the inclusions of the form $C \sqsubseteq D$. However, the application of the $\rightarrow_{\exists}$-rule is subject to Condition 3 involving the notion of S-equivalence between variables, and this condition is the basis for termination.

Rules are always applied to a tableau branch S either because in S there is an assertion $s : C$ (Condition 1), or, in the case of the $\rightarrow_{\sqsubseteq}$-rule, because of the presence of an object s in S (Condition 2). In this sense, we say that a rule is *applied to* the assertion $s : C$ or the object s (instead of saying that it is applied to the whole tableau branch S).

The above propagation rules are applied by adopting a particular strategy of application, which can be described as follows:

1. apply the $\rightarrow_{\exists}$-rule only if no other rule is applicable;
2. apply a rule to a variable only if no rule is applicable to an individual;
3. apply a rule to a variable x only if no rule is applicable to a variable y such that $y \prec x$.

These criteria are listed in descending order of priority: e.g., if $y \prec x$, the $\rightarrow_{\exists}$-rule is applicable to y, and another rule (say, $\rightarrow_{\sqcap}$) is applicable to x, producing a conflict between first and third criterion, then the first criterion overcomes the third one, and the $\rightarrow_{\sqcap}$-rule is applied to x.

The above strategy ensures that the variables are processed one at a time according to the ordering '$\prec$'. This achieves *stability* of the equivalence relation between variables: i.e., if two variables x and y appearing in a tableau branch S are S-equivalent and a rule can be applied to a third variable z such that both $x \prec z$ and $y \prec z$, then after the application of the rule x and y are S'-equivalent in the resulting tableau branch S'. Later on we show that stability is important in proving termination and complexity of the calculus.

We assume that the propagation rules are always applied according to the above strategy. Moreover, we call the $\rightarrow_{\sqcup}$-rule a *nondeterministic* rule (see Condition 2 of the rule). All the other rules are called *deterministic*. A tableau branch is *complete* if no propagation rule can be applied to it. A *clash* is an assertion of the form $s : \bot$.

Observe that the above strategy does not lead to a unique choice of application of rules. The subsequent theorems prove that whatever choice of application (respecting the strategy) is made, the calculus terminates and may yield a clash-free tableau branch if and only if the initial tableau was satisfiable.

THEOREM 1 (CORRECTNESS)

1. *Let S be a tableau branch. Then:*
   - *If S' is obtained from S by the application of a deterministic rule, then S is satisfiable if and only if S' is satisfiable.*
   - *If the nondeterministic rule $\rightarrow_{\sqcup}$ can be applied to S, and S' and S'' are the two tableau branches that can be obtained from S by the two choices in Condition 2 of the rule, then S is satisfiable if and only if either S' or S'' is satisfiable.*
2. *A complete tableau branch is satisfiable if and only if it contains no clash.*

Proof. The proof of the first part is a straightforward reformulation of soundness proofs for tableaux calculi. Regarding the second part, a tableau branch containing a clash is clearly unsatisfiable. Hence, S is satisfiable only if it contains no clash.

Suppose now S is complete and contains no clash. We are able to single out an $\mathcal{O}$-interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α that satisfy S. Such a pair $(\mathcal{I}, \alpha)$ is defined as follows:

- $\Delta^{\mathcal{I}} = \mathcal{O} \cup \mathcal{V}_S$, where $\mathcal{V}_S$ are the variables in S
- for every individual a: $a^{\mathcal{I}} = a$
- for every variable x:
  - $\alpha(x) = y$ if $x\colon \exists R.C \in S$, $x \equiv_S y$ and there is a variable z in S such that both yRz and $z\colon C$ are in S;
  - $\alpha(x) = x$ otherwise
- $A^{\mathcal{I}} = \{\alpha(s) \mid s\colon A \in S\}$
- $R^{\mathcal{I}} = \{(\alpha(s), \alpha(t)) \mid sRt \in S\}$.

We prove that $\mathcal{I}, \alpha$ satisfies every assertion in S, by induction on the length of the assertions.
Basic cases:

- if $x\colon A \in S$, then $\alpha(x) \in A^{\mathcal{I}}$ by definition.
- if $xRy \in S$ then $(\alpha(x), \alpha(y)) \in R^{\mathcal{I}}$ by definition.
- if $x\colon \neg A \in S$, then $x\colon A \notin S$, since S is clash-free. Moreover, if $x \in \mathcal{V}$, for all z such that $\alpha(x) = \alpha(z)$, $z\colon A \notin S$ since $\alpha(x) = \alpha(z)$ if and only if $x \equiv_S z$. Therefore, $\alpha(x) \notin A^{\mathcal{I}}$ by definition of $A^{\mathcal{I}}$.

Inductive cases:

$\sqcap$) if $x : C \sqcap D \in S$ then (since S is complete) $x : C \in S$, and $x : D \in S$. By inductive hypothesis, $\alpha(x) \in C^{\mathcal{I}}$, $\alpha(x) \in D^{\mathcal{I}}$. Hence $\alpha(x) \in (C \sqcap D)^{\mathcal{I}}$.

$\sqcup$) Similar to the previous case.

$\forall R.C$) if $x : \forall R.C \in S$, then since S is complete, for all y such that $xRy \in S$, $y : C \in S$. If $xRy \in S$, then $(\alpha(x), \alpha(y)) \in R^{\mathcal{I}}$ for the correspondent basic case. Moreover, if $y : C \in S$ then $\alpha(y) \in C^{\mathcal{I}}$ by inductive hypothesis. Therefore, $\alpha(x) \in (\forall R.C)^{\mathcal{I}}$.

$\exists R.C$) if $x : \exists R.C$, since S is complete one of the two following cases must occur:

1. there is an object w such that both xRw and $w : C$ are in S. Then $(\alpha(x), \alpha(w)) \in R^{\mathcal{I}}$, and $\alpha(w) \in C^{\mathcal{I}}$ by inductive hypothesis.
2. $x \in \mathcal{V}$ and there are $z, w \in \mathcal{V}$ such that $x \equiv_S z$, and both zRw and $w : C$ are in S. By definition of α, $\alpha(x) = z$. Moreover, $(\alpha(x), \alpha(w)) = (z, \alpha(w)) \in R^{\mathcal{I}}$, and $\alpha(w) \in C^{\mathcal{I}}$ by inductive hypothesis.

In both cases, we can conclude that $\alpha(x) \in (\exists R.C)^{\mathcal{I}}$.

We now consider inclusions. Suppose $C \sqsubseteq D \in S$. Since S is complete, for each object s in S, $s : C' \sqcup D \in S$, where C' is $\neg C$ rewritten in NNF. By the above induction, $\alpha(s) \in (C' \sqcup D)^{\mathcal{I}}$, that is, $\alpha(s) \in (\neg C \sqcup D)^{\mathcal{I}}$. Hence $\mathcal{I}$ satisfies $C \sqsubseteq D$.

In conclusion, $(\mathcal{I}, \alpha)$ satisfies all assertions in S, and $\mathcal{I}$ satisfies all inclusions in S. Therefore $(\mathcal{I}, \alpha)$ satisfies S. □

We now show that the number of non-equivalent variables in any tableau branch derived from a knowledge base Σ is limited. To this end we define the number of different concepts appearing in Σ.

The number of concepts appearing in an assertion $a : C$ is 1 plus the number of all subconcepts appearing in C, counting separately each occurrence of a concept. For example, if C is the concept $A \sqcap \forall R.\exists R.A$, then C contains the concept A (two times), and the concept $\exists R.A$, hence $a : C$ contains 4 concepts.

Similarly, the number of concepts appearing in an inclusion $C \sqsubseteq D$ is 1 plus the number of all subconcepts of $\neg C \sqcup D$, after $\neg C$ has been rewritten in an NNF concept—this corresponds to how such an inclusion is dealt with by the $\rightarrow_{\sqsubseteq}$-rule.

The *number of concepts* appearing in a knowledge base Σ is the sum of the numbers of concepts appearing in assertions and inclusions of Σ.

LEMMA 1 (FILTRATION) *Let k be the number of concepts appearing in a knowledge base Σ, and let S be a tableau branch derived from Σ by means of propagation rules, applied according to the strategy. If in S there are more than 2^k variables, then there are at least two variables that are S-equivalent.*

Proof. Inspecting propagation rules one can verify that each assertion $x : C \in S$ may contain only concepts appearing in Σ. Since there are k such concepts, given a variable x there cannot be more than k different assertions $x : C$ involving the same variable x in S. Hence there are no more than 2^k non-equivalent variables—as many as the subsets of concepts in Σ. □

Given a tableau branch S and an object s, we say that a variable x is a *filler* for s if for some role R, the assertion sRx is in S.

PROPOSITION 1 *Let k be the number of concepts appearing in a knowledge base Σ. In every tableau branch derived from Σ according to the strategy, for every object s there are at most k fillers for s.*

Proof. The assertion sRx can be introduced in S only by an application of the rule $\rightarrow_\exists$ to an assertion $s : \exists R.C$, and the rule can be applied at most once to the assertion (see Condition 4 of the rule). Moreover, each concept $\exists R.C$ is one of the k concepts appearing in Σ. Hence, given an object s, the number of such assertions is bounded by k. Therefore, given an object s there are no more than k assertions of the form sRx, that is, no more than k fillers for s. □

We now come to the key lemma for termination and complexity of the calculus.

LEMMA 2 *Let k be the number of concepts appearing in a knowledge base Σ, and let I be the number of individuals in Σ. In every tableau branch derived from Σ according to the strategy, there are no more than $I \cdot k + (k+1) \cdot 2^k$ variables.*

Proof. Let S be any tableau branch derived from Σ using propagation rules applied according to the strategy. Variables in S can be partitioned into equivalence classes according to the S-equivalence relation. We make use of this fact in the following three steps of the proof.

Step 1. Observe that the equivalence class of a variable x can change only because either a rule $\rightarrow_\sqcap$, $\rightarrow_\sqcup$, $\rightarrow_\sqsubseteq$, or $\rightarrow_\bot$ is applied to x, or Rule $\rightarrow_\forall$ is applied to the object s for which x is a filler. If s is an individual, then Criterion 2 of the strategy forces the application of Rule $\rightarrow_\forall$ to s before other rules are applied to x. If s is a variable, observe that Condition 2 in Rule $\rightarrow_\exists$ (which introduces x) forces $s \prec x$, hence also in this case Rule $\rightarrow_\forall$ is applied to s before all other rules are applied to x, now because of Criterion 3 of the strategy. Therefore, when a rule $\rightarrow_\sqcap$, $\rightarrow_\sqcup$, $\rightarrow_\sqsubseteq$, or $\rightarrow_\bot$ is applied to x, no rule is applicable any more to an individual or to a variable preceding x w.r.t. $\prec$. Moreover, Criterion 1 of the strategy forces the application of such rules before any variable z such that $x \prec z$ is introduced through application of Rule $\rightarrow_\exists$. Hence, the equivalence class of x does not change if there exists a variable z such that $x \prec z$.

Step 2. We now prove that for any tableau branch S, in each S-equivalence class only the the least variable w.r.t. $\prec$ may have fillers. Let x be the $\prec$-minimal variable in an S-equivalence class. By contradiction, suppose there exists a variable z such that $x \prec z$, and z has fillers. Such fillers have been introduced through Rule $\rightarrow_\exists$, applied in a previous tableau branch S'. However, from Step 1 we know that the equivalence classes of both x and z cannot change after S', since in S' other variables following z w.r.t. $\prec$ are introduced. Therefore, if x and z are S-equivalent, they were already S'-equivalent. But then, Condition 3 of Rule $\rightarrow_\exists$ was not satisfied in S', so the rule was not applicable in S', yielding a contradiction.

Step 3. Observe that variables in a tableau branch can be introduced only as fillers for an object. According to Proposition 1, the least variable in every equivalence class has at most k fillers. Since there are at most 2^k equivalence classes, from Step 2 there are at most 2^k variables with fillers, and such fillers are at most $k \cdot 2^k$ in

total. Moreover, each individual has at most k fillers. Globally, there are at most $I \cdot k + (k+1) \cdot 2^k$ variables, and the claim follows. □

From the above lemma, we now prove that any tableau branch obtained from a knowledge base Σ by the application of propagation rules has finite, exponential size w.r.t. the size of Σ. To this end, we need the following definitions.

The *size* of an assertion is the number of symbols appearing in the assertion. Observe that assertions of the form sRt have constant size, while the size of assertions of the form $s : C$ is linearly related to the number of concepts in C. Also the size of an inclusion $C \sqsubseteq D$ is the number of symbols in the inclusion, and is linearly related to the number of concepts appearing in $C \sqsubseteq D$. Finally, the size of a knowledge base is the sum of the sizes of all its assertions and inclusions, and similarly for tableau branches. Observe that also the size of Σ is linearly related to the number of concepts in Σ.

THEOREM 2 *Let n be the size of a knowledge base Σ. The size of every tableau branch derivable from Σ according to the strategy is $O(2^{cn})$, for some constant $c > 1$.*

Proof. First of all, both the number of concepts and the number of individuals in Σ is $O(n)$, hence from the above Lemma 2 the total number of objects in a tableau branch derived from Σ is $2^{c_1 n}$ for some constant c_1.

Each assertion of the form sRt has constant size. If t is an individual, then sRt is already present in Σ, hence the number of such assertions is $O(n^2)$. If t is a variable, then sRt has been added by the rule $\rightarrow_\exists$. From Condition 1 of the rule, one can verify that in this case t is involved in at most one assertion of the form sRt. Hence the number of such assertions is $O(2^{c_1 n})$. Hence globally, the number (and the size) of assertions of the form sRt is $O(2^{c_2 n})$.

Each assertion of the form $s : C$ has size proportional to the number of concepts in C. Since C itself is one of the concepts in Σ, the number of concepts in C is $O(n)$, and so is the size of $s : C$. The number of different assertions of this form is bounded by the number of objects $2^{c_1 n}$ times the number of concepts in Σ, which is $O(n)$. Hence the total size of these assertions is $O(2^{c_3 n})$, for some constant c_3.

Considering that the size of all inclusions is bounded by n, we conclude that the total size of a tableau branch derived from Σ is $O(2^{cn})$. □

From the above theorem, we conclude that our nondeterministic calculus requires exponential time to build a complete, clash-free tableau branch.

THEOREM 3 (DECIDABILITY AND COMPLEXITY) *Let Σ be an $\mathcal{AL}$-log knowledge base, and let , be either an inclusion or a membership assertion. Deciding $\Sigma \models$, is a problem in NEXPTIME.*

Our nondeterministic calculus can be turned into a deterministic one, by exploring the whole tableau one branch at a time. Since there are exponentially many complete tableau branches, we can conclude this section stating that deduction in the structural component is decidable in doubly exponential time. Actually, from correspondences with Propositional Dynamic Logic (De Giacomo and Lenzerini, 1994)

it can be proved that the problem is decidable in simple exponential time. However, the procedure stemming from such a result requires to always build an exponential-size tree automata, and then check its emptiness. Our tableaux-based method instead requires doubly exponential time only in the worst case.

4. Query Answering

In this section we describe the method used by $\mathcal{AL}$-log for hybrid deduction, i.e. for answering queries to a knowledge base. In particular, we first describe the notions of constrained SLD-derivation and constrained SLD-refutation, and then exploit such notions for devising a sound and complete method for hybrid deduction.

In the following we refer to a hybrid $\mathcal{AL}$-log knowledge base $K = \langle \Sigma, \Pi \rangle$ and to a query $Q = q_1, \ldots, q_n \ \& \ \beta_1, \ldots, \beta_m$ posed to K. If E is a constrained Datalog clause of the form $\alpha_0 :- \alpha_1, \ldots, \alpha_n \ \& \ \beta'_1, \ldots, \beta'_h$ we call α_0 the *head* of E, i.e. the positive literal of E.

Definition 1. [Constrained resolution] Let Q, E, α_0 be as above, and let θ be the most general substitution such that $\alpha_0\theta = q_i\theta$, where q_i is one of $\{q_1, \ldots, q_n\}$. The *resolvent* of Q and E with substitution θ is the query $\gamma'' \ \& \ \beta''_1, \ldots, \beta''_k$, where

- $\gamma'' = (q_1, \ldots, q_{i-1}, \alpha_1, \ldots, \alpha_n, q_{i+1}, \ldots, q_n)\theta$,
- the *new constraints* $\beta''_1, \ldots, \beta''_k$ are obtained from $\beta_1\theta, \ldots, \beta_m\theta, \beta'_1\theta, \ldots, \beta'_h\theta$ by applying the following simplification: if there are two constraints of the form $t\colon C, t\colon D$ they are replaced by the equivalent constraint $t\colon C \sqcap D$.

We can now provide the definition of constrained SLD-derivation.

Definition 2. A *constrained SLD-derivation* (or simply *derivation*) for a query Q_0 in K is a derivation constituted by:

1. a sequence of queries $Q_0, \ldots, Q_n$;
2. a sequence of constrained Datalog clauses $E_1, \ldots, E_n$;
3. a sequence of substitutions $\theta_1, \ldots, \theta_n$,

such that for each $i \in \{0, \ldots, n-1\}$, Q_{i+1} is the resolvent of Q_i and E_{i+1} with substitution θ_{i+1}. We call n the length of the derivation.

Figure 3 shows an example for a derivation of the query *mayDoThesis(paul,john)* in the knowledge base K_{univ} of Section 2.4.

A derivation may terminate with the last query of the form $\ \& \ \beta_1, \ldots, \beta_m$, which is called *constrained empty clause* (or simply empty clause).

PROPOSITION 2 *Let $Q_0, \ldots, Q_n$ be a constrained SLD-derivation for Q_0 in K. If $\mathcal{J}$ is a model of K such that $\mathcal{J} \models Q_{i+1}$, then $\mathcal{J} \models Q_i$ for $i = 0, \ldots, n-1$.*

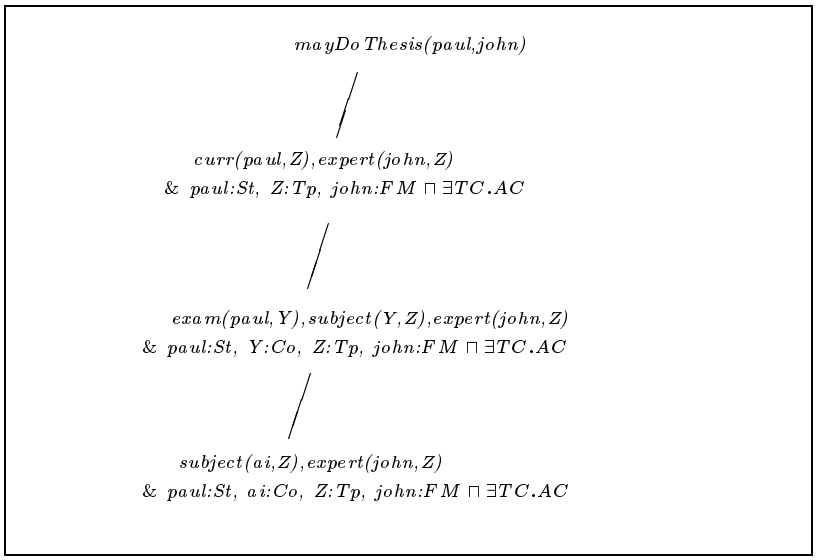

Figure 3. A constrained SLD-derivation

Proof. It follows from the soundness of SLD-resolution and the fact that the simplification of constraints preserves validity. □

As pointed out in (Baader *et al*, 1990, Bürckert, 1994), in our framework, a derivation of the empty clause with associated constraints , does not represent a refutation, because what is actually inferable from such a derivation is that the query is true in those models of K that satisfy , .

Therefore, in order to answer to a query, we must collect enough derivations ending with a constrained empty clause, such that every model of K satisfies the constraints associated with the final query of at least one derivation. Let us formalize this notion as follows.

Definition 3. A *constrained SLD-refutation* for a query Q in K is a finite set of constrained SLD-derivations $d_1, \ldots, d_m$ for Q in K such that, denoting as $Q^i_0 \ldots, Q^i_{n_i}$ the sequence of queries of the i-th derivation d_i, the following conditions hold:

1. for each i, $Q^i_{n_i}$ is of the form $\& \beta^i_1, \ldots, \beta^i_{q_i}$, i.e. the last query of each derivation is a constrained empty clause;
2. for every model $\mathcal{J}$ of K, there exists at least one $i \in \{1, \ldots, m\}$ such that $\mathcal{J} \models Q^i_{n_i}$; we write this condition $K \models disj(Q^1_{n_1}, \ldots, Q^m_{n_m})$.

Figure 4 shows a set of two derivations for the query *mayDoThesis(paul,mary)* in the knowledge base K_{univ}, where each derivation ends with a constrained empty

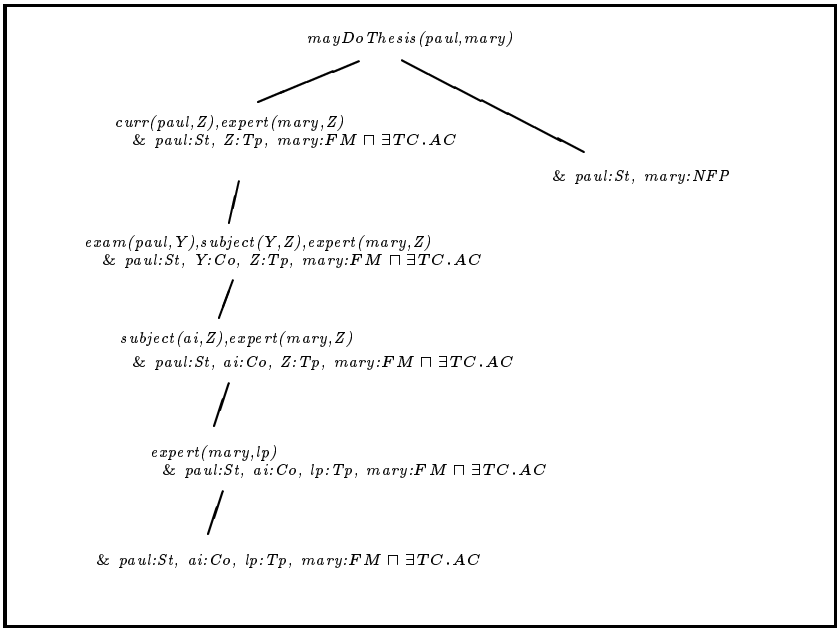

Figure 4. A constrained SLD-refutation

clause. It is possible to verify that in every model of K_{univ} either the constraints of the first empty clause (i.e. $paul : St,\ ai : Co,\ lp : Tp,\ mary : FM \sqcap \exists TC.AC$) or the constraints of the second one (i.e., $paul : St,\ mary : NF$) are true; therefore this set of constrained SLD-derivations constitutes a constrained SLD-refutation.

Let us now show how the notion of constrainted SLD-refutation can be used for query answering. Let Q be a query to a knowledge base K. Following a standard terminology, an answer θ to Q is called a *computed* answer if $Q\theta$ has a refutation in K. The set of computed answers of Q is called the *success set* of Q in K.

Let us first check whether looking for constrained SLD-refutations is a sound and complete method for checking if a *ground* query Q is a logical consequence of a knowledge base K. We write $K \vdash Q$, if there is a constrained SLD-refutation for Q in K.

LEMMA 3 *Let Q be a ground query to a knowledge base K. $K \vdash Q$ if and only if $K \models Q$.*

Proof. The usual steps for proving correctness and completeness of SLD-resolution in the logic programming framework can be reconstructed in our hybrid framework. "$\Rightarrow$" Suppose $K \vdash Q$, i.e. the ground query Q has a constrained SLD-refutation. Then, for each derivation, if $\mathcal{J}$ is a model of K that satisfies the constrained empty clause then it satisfies Q (by repeated application of Proposition 2); moreover, each model $\mathcal{J}$ of K satisfies at least one of the constrained empty clauses. Then each model of K satisfies Q, that is $K \models Q$.

"$\Leftarrow$" Suppose $K \nvdash Q$. Consider any derivation of Q whose last query is a constrained empty clause, and call $Q^i_{n_i} = \ \& \ \beta^i_1, \ldots, \beta^i_{q_i}$ its last query. If there is no constrained SLD-refutation, then there is a model $\mathcal{J}$ of K such that $\mathcal{J} \not\models Q^i_{n_i}$, that is, there is a model $\mathcal{I}$ of Σ such that $\mathcal{I} \not\models \beta^i_1, \ldots, \beta^i_{q_i}$.

Given the model $\mathcal{I}$ of the structural part Σ, a monotone mapping $T_{\Pi,\mathcal{I}} : 2^{HB} \rightarrow 2^{HB}$ can be defined as follows. Given a set $V \subseteq HB$ of ground atoms, $T_{\Pi,\mathcal{I}}(V)$ is the set of all ground atoms $\{L_1, \ldots, L_k\}$ such that L_i is the head of a ground instance δ' of a clause δ in Π, $\mathcal{I}$ satisfies all constraints in δ', and V contains all atoms in the body of δ'.

$T_{\Pi,\mathcal{I}}$ is monotone, and its least fixpoint $lfp(T_{\Pi,\mathcal{I}})$ can be obtained by iterating $T_{\Pi,\mathcal{I}}$ a finite number of times, starting from $\emptyset$. It can be verified that $lfp(T_{\Pi,\mathcal{I}})$, together with $\mathcal{I}$, identifies a model of K.

Hence consider the model $\mathcal{J}'$ characterized by $\mathcal{I}$ and $lfp(T_{\Pi,\mathcal{I}})$. It can be shown—by induction on the construction of $lfp(T_{\Pi,\mathcal{I}})$—that $\mathcal{J} \not\models Q$, and therefore $K \not\models Q$. □

The next theorem states that every answer in the success set for Q in K is correct with respect to K, and vice versa.

THEOREM 4 *Let Q be a query to a knowledge base K. The success set of Q in K coincides with the answer set to Q in K.*

Proof. Since every answer a in the answer set to Q in K is ground, it follows directly from Lemma 3 that a is in the success set of Q in K.

To prove that the converse holds we still apply Lemma 3, but, additionally we must show that every computed answer in the success set of Q in K is such that $Q\theta$ is ground. Because of the safety restriction we have that every variable in the query is either substituted by another variable that appears in the next query of the derivation when a conjunct is resolved with a rule, or bound to a constant when a conjunct is resolved with a fact. Therefore, the empty clause can be derived only if the variables in the query have been bound to constants. □

Let us now turn to the method for answering queries in $\mathcal{AL}$-log. First, it is easy to see that the usual reasoning methods for Datalog allow us to collect in a finite number of steps (actually in a number of steps which is polynomial with respect to the size of the extensional level of the constrained Datalog program) enough constrained SLD-derivations for Q in K to construct a refutation—if any.

PROPOSITION 3 *Let Q be a query to K. Then $K \vdash Q$ if and only if there is a constrained SLD-refutation built by collecting all the constrained SLD-derivations ending with the empty clause and whose length is bounded by a constant determined by K.*

Proof. It is well-known (see for example (Ceri *et al*, 1990, p.120)) that SLD-resolution for Datalog programs is complete if one computes the refutation trees up to a finite depth depending on K. Since there is a straightforward one-to-one mapping between constrained SLD-derivations and the Datalog derivations (obtained by ignoring the constraints), the claim follows. □

Given a refutation $\{d_1, \ldots, d_n\}$, where each derivation d_i ends with the constrained empty clause $Q^i_{n_i}$, it remains to verify whether $K \models disj(Q^1_{n_1}, \ldots, Q^m_{n_m})$. Clearly, once SLD-derivations have been collected, only the structural component Σ of the $\mathcal{AL}$-log knowledge base needs to be considered. Following Baader et al. (Baader *et al*, 1990), we have that this condition is verified if and only if for every set of assertions $a_1 : C_1, \ldots, a_m : C_m$ such that $a_i : C_i$ appears as a constraint in $Q^i_{n_i}$, $\Sigma \cup \{a_1 : \neg C_1, \ldots, a_m : \neg C_m\}$ is unsatisfiable. This problem has been proved to be decidable in the previous section, and one can perform the check using at most k^m unsatisfiability checks on Σ, where k is the maximum number of constraints in each $Q^i_{n_i}$, and m is the number of constrained SLD-derivations constituting the refutation.

THEOREM 5 *Query answering in $\mathcal{AL}$-log is decidable.*

Proof. In order to answer a query Q to the knowledge base K, one can consider each ground instance Q' of Q, collect the set of all constrained SLD-derivations $d_1, \ldots, d_m$ of bounded length (with $d_i = Q^i_0 \ldots, Q^i_{n_i}$) for Q' in K, and then check whether $K \models disj(Q^1_{n_1}, \ldots, Q^m_{n_m})$. □

5. Conclusions

We have presented $\mathcal{AL}$-log, a system integrating Datalog with description logics. We considered a structural subsystem based on the language $\mathcal{ALC}$ and a relational subsystem based on Datalog, extended by allowing $\mathcal{ALC}$-constraints in the body of the clauses.

It is worth noticing that the choice of the specific language $\mathcal{ALC}$ used in our architecture is not mandatory. In fact, other description logic languages, even outside the $\mathcal{AL}$-family, could be employed for our purposes. Obviously, the complexity of reasoning in the overall architecture could be different.

The main goal of our work was to study a form of hybrid deduction over a knowledge base with two components, one concerning structural knowledge on concepts and objects, and the other one concerning relational knowledge on objects. The resulting reasoning procedure is obtained by adding to the deduction procedure of Datalog special deduction steps on the structural component.

As pointed out in the introduction, while $\mathcal{AL}$-log is highly expressive in the structural subsystem, it is in fact still lacking some interesting features, such as the treatment of negation in the relational subsystem, and the ability to use the relational subsystem to infer knowledge about the structural component.

An extension of $\mathcal{AL}$-log to deal also with role assertions in the constrained Datalog clauses has been studied in (Levy *et al*, 1996) within the system CARIN. It is been shown that such extension is not straightforward, and that some restrictions must be imposed in order to retain decidability of the reasoning services.

Several developments of our work are possible. First of all, an analysis can be carried out in order to single out sublanguages of $\mathcal{ALC}$ that give rise to polynomial time of query answering. Second, the relational language can be extended with

a stratified form of negation, and both the semantics and the method for hybrid deduction can be adapted according to the characteristics of the enhanced relational component.

Acknowledgments

This work has been supported by "Data Warehouse Quality (DWQ)", Esprit Long Term Research, Project No. 22469 and by the Italian MURST 60%.

Notes

1. Since no function symbols other than constant symbols are allowed in Datalog, constants and variables are called terms.
2. A tableau branch was called *constraint system* in e.g., (Donini *et al*, 1991, Schmidt-Schauß and Smolka, 1991). To avoid confusion between constraints in constraint systems and constraints in Datalog clauses, in this paper we use the term "tableau branch" for a former constraint system.

References

Serge Abiteboul. Towards a deductive object-oriented database language. *Data and Knowledge Engineering*, 5:263–287, 1990.

Serge Abiteboul and Paris Kanellakis. Object identity as a query language primitive. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 159–173, 1989.

H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, and Jörg H. Siekmann. Concept logics. In John W. Lloyd, editor, *Computational Logics, Symposium Proceedings*, pages 177–201. Springer-Verlag, 1990.

Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge (PDK-91)*, number 567 in Lecture Notes in Artificial Intelligence, pages 67–86. Springer-Verlag, 1991.

Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 59–67, 1989.

Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)*, pages 532–539, Los Angeles, 1985.

Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

Hans-Jürgen Bürckert. A resolution principle for constrainted logics. *Artificial Intelligence*, 66(2):235–271, 1994.

Marco Cadoli, Luigi Palopoli, and Maurizio Lenzerini. Datalog and description logics: Expressive power. Preliminary report. In *Proc. of the 1996 Description Logic Workshop (DL-96)*, 1996.

S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases.* Springer-Verlag, Berlin, 1990.

A. G. Cohn. Taxonomic reasoning with many-sorted logics. *Artificial Intelligence Review*, 3:89–128, 1989.

Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 205–212. AAAI Press/The MIT Press, 1994.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 151–162. Morgan Kaufmann, Los Altos, 1991a.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. A hybrid system integrating Datalog and concept languages. In *Proc. of the 2nd Conf. of the Italian Association for Artificial Intelligence (AI*IA-91)*, number 549 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1991b. An extended version appeared also in the Working Notes of the AAAI Fall Symposium "Principles of Hybrid Reasoning".

Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.

Alan M. Frisch. A general framework for sorted deduction: Fundamental results on hybrid reasoning. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)*. Morgan Kaufmann, Los Altos, 1989.

Alan M. Frisch and A. G. Cohn. Thoughts and afterthoughts on the 1988 workshop on principles of hybrid reasonig. *AI Magazine*, 12:77–83, 1991.

Michael Kifer, Georg Lausen, and James Wu. Logical foundations of Object-Oriented and frame-based languages. *Journal of the ACM*, 42(3), 1995.

Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 40–47, 1996.

Alon Y. Levy and Marie-Christine Rousset. The limits on combining recursive Horn rules with description logics. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 577–584, 1996.

Tok Wang Ling, Alberto O. Mendelzon, and Laurent Vicille, editors. *Deductive and Object-Oriented Databases*, number 1013 in Lecture Notes in Computer Science. Springer-Verlag, 1995.

Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

Bernhard Nebel and Kai von Luck. Hybrid reasoning in BACK. In *Proc. of the 3rd Int. Sym. on Methodologies for Intelligent Systems (ISMIS-88)*, pages 260–269. North-Holland Publ. Co., Amsterdam, 1988.

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

Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.

Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, Potomac, Maryland, 1988.

Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

Francesco M. Donini received his PhD at the Università di Roma "La Sapienza" in 1992, where he is currently assistant professor. His research interests are in description logics and computational properties of knowledge representation formalisms.

Maurizio Lenzerini is a full professor of Computer Science at the Università di Roma "La Sapienza", Italy. He has been involved in several research projects on conceptual data modeling, databases and artificial intelligence, knowledge representation and reasoning, computational complexity of reasoning. His recent research work deals with the foundation of the object-oriented approach to knowledge representation and reasoning, and its application to database problems, such as conceptual data modeling, database integration, and cooperative information systems.

Daniele Nardi is an associate professor of Computer Science at the Università di Roma "La Sapienza", Italy. His research interests include theoretical aspects of knowledge representation and reasoning, in particular, concept description logics, non-monotonic reasoning as well as practical applications of knowledge-based system, in particular intelligent tutoring systems.

Andrea Schaerf received his PhD at the Università di Roma "La Sapienza" in 1994, where he is currently assistant professor. His research interests are in knowledge representation, scheduling problems, and constraint programming.