Logic Programming Languages for Databases and the Web

Sergio Greco[1] and Francesca A. Lisi[2]

[1] Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria
Via P. Bucci, 41C - Arcavacata di Rende (CS), Italy
greco@deis.unical.it
[2] Dipartimento di Informatica, Università degli Studi di Bari "Aldo Moro"
Via E. Orabona, 4 - 70125 Bari, Italy
lisi@di.uniba.it

Abstract. This chapter contains a reference selection of Italian contributions in the intersection of Logic Programming (LP) with databases and the (Semantic) Web. More precisely, we will survey the main contributions on deductive databases such as the coupling of Prolog systems and database systems, evaluation and optimization techniques, Datalog extensions for expressing nondeterministic and aggregate queries, and active rules and their relation to deductive rules. Also we will illustrate solutions employing LP for querying the Web, manipulating Web pages, representing knowledge in the Semantic Web and learning Semantic Web ontologies and rules.

1 Introduction

Deductive databases started more than 30 years ago and this area has been characterized by intensive research for the past years. It stemmed from earlier work on logic and databases [37,39] that was reviewed in an excellent paper by Gallaire et al. [38]. Deductive databases extend the power of relational systems in several ways [96] by allowing:

- the capability to express, by means of logical rules, recursive queries and efficient algorithms for their evaluation against stored data;
- support for the use of nonmonotonic features such as negation;
- the expansion of the underlying data domain to include structured objects;
- extensions beyond first-order logic for the declarative specification of database operations as updates;
- the development of optimization methods that guarantee the translation of the declarative specifications into efficient access plans and their termination when executed.

Although deductive databases have not found widespread adoptions outside academia, some of their concepts are used in many fields where databases and information systems are used. Over the years the research in different areas where logic-based languages are used for modeling information system features and

A. Dovier, E. Pontelli (Eds.): 25 Years of Logic Programming, LNCS 6125, pp. 183–203, 2010.

managing large datasets has benefited of the results of deductive databases (e.g. integration of advanced features in SQL standards, nonmonotonic reasoning, artificial intelligence and others).

The World Wide Web (WWW, or simply Web) is nowadays the most famous information system. Its success is witnessed by its enormous size and rate of growth. However, the success itself has given raise to a status of the WWW where more sophisticated techniques are urgently needed to properly handle such information overload. Recent years have seen a tremendous interest for Web technologies that can employ some form of logical reasoning. In particular, the ambitious plan for an evolution of the WWW, the so-called Semantic Web [10], has shown that it is of primary importance to find an appropriate interaction between the Web infrastructure, and solutions coming from the LP research area. Interest in the (Semantic) Web application context is testified by initiatives such as the ALPSWS series of international workshops on *Applications of Logic Programming to the (Semantic) Web and Web Services*[1] started in 2006 and traditionally co-located with the *International Conference on Logic Programming*, and the recent special issue of the TPLP journal [72].

We point out that several topics considered in this chapter have also been investigated in the areas of Non-Monotonic Reasoning (NMR) and Answer Set Programming (ASP) and, for more information, we refer readers to [46,12]. The connection between the areas of deductive databases and (Semantic) Web is strong as the Web can be modeled as a (huge) database and the research in both fields is mainly devoted to extend Datalog to have enough expressivity and ensure efficiency in querying and managing relational databases and the (Semantic) Web [15]. This chapter contains a reference selection of Italian contributions in the intersection of LP with databases (section 2) and the (Semantic) Web (section 3).

2 Deductive Databases and Logic Programming

In this section we will discuss some of the research aspects in deductive databases with a particular attention to some fields which have been of particular interest for the Italian deductive database community. In the next subsection we will present some basic definitions on Datalog [22,98]. Next, we will discuss the coupling of Prolog systems and database systems (subsection 2.2), and evaluation and optimization techniques (subsection 2.3). Subsequently, we will present some Datalog extensions regarding the possibility to express nondeterministic and aggregate queries (subsections 2.4 and 2.5). Finally, we will discuss active rules and their relation to deductive rules (subsection 2.6).

2.1 Datalog

A Datalog program is a logic program without function symbols. The restriction imposed by Datalog allows to have finite models which can be efficiently computed by means of a standard bottom-up evaluation. Extensions allowing

[1] `http://www.kr.tuwien.ac.at/events/alpsws2008/`

complex, finite objects have also been considered, but for the sake of simplicity, we restrict ourselves to only consider simple terms.

The semantics of a *positive* (i.e., negation-free) program P is given by the minimum model that coincides with the least fixpoint $T_P^\infty(\emptyset)$ of the *immediate transformation* T_P [68]. The semantics of a logic program with negation is given by the stable model semantics [42,90]. Stable models are said to be total (2-valued) if atoms can be either $true$ or $false$, with the standard order $false < true$, or partial (3-valued) if the truth value of atoms can be $true$, $false$ or undefined, with the order $false < undefined < true$. On the set of partial stable models it is possible to define an order relation so that they define a semi-lattice [90]. Minimal and stable model semantics have also been extended for programs with disjunctive heads [84]. It is worth observing that although different alternative semantics have been proposed so far (e.g. well-founded semantics, deterministic models, minimal founded semantics [41,93,49,35]), total stable model semantics has been widely accepted by the nonmonotonic reasoning community and for stratified programs these semantics coincide. In particular, general programs with negation may have 0, one or several stable models, *positive* (i.e., negation-free) standard programs have unique (total) stable models, corresponding to the minimum model, and *stratified* programs (i.e. programs where recursion does not "pass" through negated atoms) have a unique (total) stable model which is called *perfect* model.

Generally, the logic language Datalog is denoted by DATALOG$^\neg$, whereas restrictions allowing only stratified negation or positive rules only are denoted, respectively, by DATALOG$^{\neg s}$ and DATALOG; the extension allowing disjunctive heads is denoted by DATALOG$^{\neg\vee}$ [27]. Predicates are usually partitioned into *extensional* (or EDB) and *intensional* (or IDB) predicates. EDB predicates are associated with facts denoting the input databases[2], whereas IDB predicates are associated with rules denoting a set of (possibly recursive) views. The input database will be denoted by D, the program will be denoted by P and it is assumed that $|P| \ll |D|$ (the size of D is much greater than the size of P). It is also assumed that the rules of our programs are *safe* [98], i.e. variables appearing in the head or in negated literals in the body of rules are range restricted, i.e., they take values from the database.

The unique stable model of a stratified program P, applied to a database D, can be computed in polynomial time in the size of D (i.e., the number of symbols in D). For general programs we have that i) the existence of a stable model is not guaranteed, ii) finding a stable model is $\mathcal{NP}$-hard, and iii) deciding whether a program admits some stable model is $\mathcal{NP}$-complete. For programs with disjunctive heads the complexity is even higher (in the general case, in the second level of the polynomial hierarchy). The notion of *data complexity* is defined naturally by viewing the program P as a function computed on the database (which is thus viewed as the input variable). Another important notion is that of *genericity* which means that the database is unchanged if all constants are consistently renamed [3].

[2] For each tuple t belonging to a relation r of the input database there is a fact $r(t)$.

A *query* Q is a pair $\langle g, P \rangle$ where P is a program and g is a predicate symbol in P denoting the output relation; The *answer* of Q on D, denoted by $Q(D)$, is the set of relations on g denoted as $A_g = \{M(g) | M \text{ is a stable model of } P \cup D\}$. Two queries Q_1 and Q_2 are equivalent ($Q_1 \equiv Q_2$) if for each database D the answers of Q_1 and Q_2 on D are the same. The query is called *deterministic* or *non-deterministic* according to whether the mapping is single-valued or multi-valued.

Since it is assumed to deal with large databases, only tractable (i.e. polynomial time) queries are considered. Therefore, in the rest of this chapter we only consider stratified queries and tractable extensions. For queries using unstratified negation and/or disjunctive heads we address the reader to "nonmonotonic formalisms" which are discussed in [46,12].

2.2 Coupling Relational Databases with LP Systems

Integrating LP and relational databases has been recognized to be very promising since both LP and relational databases are related by their common ancestry of mathematical logic [38]. The combination of advanced query processing facilities, typical of expert systems, and efficient access techniques of relational database systems has been very promising. In particular, Prolog systems would greatly benefit from both the ability to store large amounts of information in secondary memory and the optimization techniques built into database systems.

The coupling of a Prolog front-end with a database back-end has been a very promising vehicle for developing database and knowledge-based applications and has received a lot of attention in the field's early years. In practice, systems linking Prolog and a relational database system simply tack on a software interface between a pre-existing Prolog implementation and a pre-existing relational database system. In other words, the two systems are loosely coupled. The interface allows Prolog to query the database when needed, either via the automatic translation of Prolog goals into SQL or else by directly embedding SQL statements into the Prolog code.

In designing the interface between a relational database and a Prolog interpreter, persistence and efficiency were the major concerns. Persistence is obtained by the capability of storing not only data but rules in the database as well. Thus, after a session is over and a new session starts, the user does not need to reassert the knowledge asserted in the past. Considering efficiency, the objective of minimizing the interaction between the two systems is achieved by means of an optimizing translation mechanism.

A method for loading into the memory-resident database of Prolog facts permanently stored in secondary storage was proposed in [23,22]. The rationale of the method is to save queries accessing the database by never repeating the same query. This is carried out by storing in the main memory, in a compact and efficient way, information about the past interaction with the database. An underlying assumption of this approach is the availability of large core memories on the machine running Prolog [47].

2.3 Query Evaluation and Optimization

In the computation of a query Q over a database D, two main approaches have been proposed in the literature: the *top-down* computation (used by Prolog) and the *bottom-up* computation (used by deductive database languages). The latter is based on the fixpoint operator T_P and on "database implementations" such as the naive algorithm which transforms rules into relational algebra expressions which are evaluated repeatedly until a fixpoint is reached; the semi-naive algorithm improves the naive algorithm avoiding to re-evaluate relational expressions on the same sets of facts [98]. With the top-down approach, only rules and atoms relevant to the query are considered, but termination and duplicated computation are problematic issues. The bottom-up strategy always terminates instead, but it may compute irrelevant atoms. Therefore, optimization techniques combining top-down and bottom-up strategies have been proposed to try to compute only atoms which may be "relevant" for the query in a bottom-up fashion. The key idea here consists of rewriting the rules with respect to the query in order to answer it without actually referring to irrelevant facts. The well-known *magic-set* technique is based on rewriting the rules in P (for a given query $Q = \langle g, P \rangle$) into a set P^α such that, let $Q = \langle g, P^\alpha \rangle$, Q and Q^α are query-equivalent, i.e. the sets of g-facts computed by Q and Q^α are the same [9,97]. General rewriting techniques can be applied to all queries, but their efficiency is limited, while specialized techniques can be very efficient, but have limited applicability. An interesting class of queries is the one known as chain queries, i.e. queries where bindings are propagated from arguments in the head to arguments in the tail of the rules, in a chain-like fashion. For these queries, which are rather frequent in practice, insisting on general optimization methods is not convenient, while specialized methods for subclasses thereof have been proposed, but do not fully exploit bindings. The *counting* method specialized for bound chain queries was proposed to further improve the efficiency of queries [91,53]. However, although proposed in the context of general queries, it preserves the original efficiency only for a subset of chain queries whose recursive rules are linear. The so-called *pushdown* method, exploiting the relationship between chain queries, context-free languages and pushdown automata, was later proposed [51,52]. It rewrites queries into a form that is more suitable for the bottom-up evaluation, i.e. translates a chain query into a factorized left-linear program implementing the pushdown automaton recognizing the language associated with the query. A nice property here is that it reduces to the counting method in all the cases where the latter method behaves efficiently and introduces a unified framework for the treatment of special cases, such as the factorization of right-, left-, mixed-linear programs, as well as the linearization of non-linear programs [97].

2.4 Choice and Non-determinism in Datalog

Stable model semantics introduces a sort of non-determinism in the sense that programs may have more than one "intended" model [42]. Non-determinism

offers a solution to overcome the limitations in expressive power of deterministic languages [4,5]. For instance, non-determinism can be used to capture the class of polynomial-time queries on unordered domains [5,50]. The problem with stable model semantics is that the expressive power can blow up without control, so that polynomial time resolution is no longer guaranteed. Thus, it is possible that polynomial time queries are computed in exponential time, that is, it is possible to get exponential time resolution. In order to guarantee polynomial time computability and the existence of stable models, nondeterministic constructs and semantics have been proposed.

Given a query $Q = \langle g, P \rangle$ and a database D, the *answer* to Q on D is a relation defined as follows:

1. under *non-deterministic semantics*: $M(g)$, for some stable model M for $P \cup D$ and $\emptyset$ if no stable model exists;
2. under *possibility semantics* with ground query goal: $M(g)$, if there exists a stable model M such that $M(g) \neq \emptyset$ and $\emptyset$ otherwise — thus, the answer to a query can be either "true" or "false";
3. under *certainty semantics*: $\bigcap M_i(g)$ for all stable models M_i.

Moreover, the mappings defined by queries are multi-valued under non-deterministic semantics and single-valued under possible and certain semantics, i.e., possible and certain semantics are deterministic. In practice, to answer a query under non-deterministic semantics it is sufficient to find any relation in $Q(D)$; this corresponds to determining any stable model. As discussed above, full negation under stable model semantics cannot be used in practical database languages because the complexity is not guaranteed to be polynomial also for queries expressing polynomial problems.

A controlled usage of stable model semantics has been proposed in [44,50,92], where the *choice* construct, first introduced in [60], has been given a stable model semantics. The *choice* construct is used to enforce functional constraints in rules. Thus, an atom of the form, $choice((X),(Y))$, where X and Y denote vectors of variables, in a rule r denotes that any consequence derived from r must respect the functional dependency $X \rightarrow Y$.

A fixpoint algorithm for Datalog programs with choice constructs (called Choice Fixpoint Procedure) has been proposed in [44,45], where it has also been shown that the time complexity of computing, nondeterministically, a stable model, and consequently a nondeterministic answer, is polynomial. This procedure has been extended to programs with stratified negation in [50] where it has also been shown that given a database D and a stratified program with choice P, the problem of deciding if there exists a stable model M (non-deterministic semantics) for $P \cup D$ is polynomial time. In the same paper it has been demonstrated that the class of nondeterministic polynomial problems is captured by Datalog with stratified negation and choice. Therefore, *choice* is a powerful don't-care form of non-determinism which allows one to express some problems for which domain ordering is needed but is not available [5,43].

2.5 Aggregates in Datalog

Early research on deductive databases strived to support a declarative high-level formulation of problem solution without surrendering the performance obtainable by careful programming in an imperative language. In this respect, an interesting challenge is posed by optimization problems, such as finding the minimum spanning tree in a graph or the knapsack problem, that are encountered in several applications.

Datalog, enriched with extrema (*least*/*most*) and *choice* constructs, can express and efficiently solve optimization problems requiring a greedy search strategy [40,54]. Moreover, many optimization problems can be solved efficiently using a *dynamic programming* technique that is based on the division of the problem into subproblems: the original problem is divided into simpler subproblems that are solved separately; their solutions are then used to solve the original problem. Therefore, Datalog extensions allowing to express classical problems whose efficient solutions are based on greedy and dynamic programming methods have been proposed as well [48].

These extensions are based on the definition of built-in aggregate predicates which enhance Datalog representational capabilities, making it possible to naturally express many well-known algorithms that have wide applicability. The extension of Datalog with classical aggregates (*least*, *most*, *count* and *sum*) has been investigated by considering two main aspects: the definition of suitable semantics for programs with aggregates and the efficiency of the evaluation.

The main novelty of the proposed approach is that only stratified aggregation and a semantics allowing to define linear orders on the input domain are considered. Moreover, the paper also considers in some cases unstratified negation to guarantee efficiency and termination. Another important novelty is that the paper introduces a new aggregate, called *summation*, which combined with *least* and *most* permits us to express and efficiently compute optimization problems such as dynamic programming and integer programming problems. More specifically, the global class of integer programming problems can be easily expressed in the proposed framework and extended programs can be efficiently computed by using a dynamic programming evaluation technique.

The possibility of transforming queries with *least* and *most* predicates into equivalent queries that can be computed more efficiently has been investigated in [36]. Recently there have been further proposals to extend the well-founded and stable model semantics with unstratified aggregates [16,80,95]. Moreover, as pointed out in [77], unstratified aggregates are not necessary if ordered domains and arithmetics are available.

2.6 Deductive and Active Databases

The field of active databases is based on logics and combines techniques from databases, expert systems and artificial intelligence. The main peculiarity of this technology is the support for automatic 'triggering' of rules in response to events. Automatic triggering of rules can be useful in different areas such as

integrity constraint maintenance, update of materialized views, knowledge bases and expert systems [100].

Active rules follow the so called *Event-Condition-Action* (ECA) paradigm; rules autonomously react to events occurring on the data, by evaluating a data dependent condition and executing a reaction whenever the condition is true. Active rules consist of three parts: *Event* (which causes the rule to be triggered), *Condition* (which is checked when the rule is triggered) and *Action* (which is executed when the rule is triggered and the condition is true). Thus, according to the semantics of a single active rule, the rule reacts to a given event, tests a condition, and performs a given action.

Understanding the behavior of active rules, especially in the case of rules which interact with one another, is very difficult, and often the actions performed are not the expected ones. The semantics of active rules are usually given in terms of execution models, specifying how and when rules will be applied, but execution models are not completely satisfactory since their behavior is not always clear and could result in nonterminating computations. Most commercial active rule systems operate at a relatively low-level of abstraction and are heavily influenced by implementation-dependent procedural features. A further problem of active databases is that, as shown in [81], most of the operational semantics proposed in the literature have very high complexity and expressivity (PSPACE or even higher complexity).

Different solutions using deductive database semantics to provide a clear semantics to active rules have been proposed. Here we recall the solution proposed in [61,11,33] where declarative semantics are associated to active rules, and in [101,75], where active rules are modeled by means of deductive rules with an attribute denoting the state of the computation. The advantage of associating a declarative semantics to active rules is that confluence and termination are guaranteed and complexity is much lower.

In some sense, active and deductive rules can be seen as opposite ends of a spectrum of database rule languages [99]. Deductive rules provide a high-level powerful framework for specifying intensional relations. In contrast, active rules are more low-level and often need explicit control on rule execution. The problem of providing a homogeneous framework for integrating, in a database environment, active rules, which allow the specification of actions to be executed whenever certain events take place, and deductive rules, which allow the specification of deductions in a logic programming style has been investigated in [79,61].

Since active rules are often used to make databases consistent, *active integrity constraints (AICs)*, an extension of integrity constraints for consistent database maintenance [21], have been recently proposed [17]. An active integrity constraint is a special constraint whose body contains a conjunction of literals which must be false and whose head contains a disjunction of update actions representing actions (insertions and deletions of tuples) to be performed if the constraint is not satisfied (that is its body is true). The AICs work in a domino-like manner as the satisfaction of one AIC may trigger the violation and therefore the activation of another one. The advantage of AICs is that they have declarative semantics

(i.e. they can be rewritten into logic rules), lower complexity than active rules and can be used to compute consistent answers, even if the source database is inconsistent [18]. An alternative semantics for AICs is proposed in [19], whereas its relationships to Revision Programming is investigated in [20].

3 From Databases to the (Semantic) Web

The Web has caused a revolution in how we represent, retrieve, and process information. Its growth has given us a universally accessible database but in the form of a largely unorganized collection of documents. This is changing, thanks to the simultaneous emergence of new ways of representing data: from within the Web community, the eXtensible Markup Language (XML)[3]; and from within the database community, *semistructured data*. The convergence of these two approaches has rendered them nearly identical, thus promoting a concerted effort to develop effective techniques for retrieving and processing both kinds of data [2]. In spite of the success of XML as data interchange format, it has turned out very soon that XML has severe limits in conveying data semantics.

The Semantic Web is an evolving extension of the Web in which the semantics of information and services on the Web is defined, making it possible for the Web to understand and satisfy the requests of people and machines to use the Web content [10]. It derives from W3C (World Wide Web Consortium) director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange. At its core, the Semantic Web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements of the Semantic Web are expressed as prospective future possibilities that are yet to be implemented or realized. The Semantic Web architecture is a stack of layers, on top of XML, each of which equipped with one or more mark-up languages, notably the Resource Description Framework (RDF)[4], the RDF Schema (RDFS)[5] and the Web Ontology Language (OWL)[6] all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain. The use of formal specifications, also called *ontologies*, fairly overcomes the aforementioned limits of XML.

In this section, we will survey solutions employing LP for querying the Web (subsection 3.1), for manipulating Web pages (subsection 3.2), for representing knowledge in the Semantic Web (subsection 3.3) and for learning Semantic Web ontologies and rules (subsection 3.4).

3.1 LP-Based Query Languages for the Web

The Web can be seen as a vast heterogeneous collection of databases, which must be queried in order to extract information. In fact, in many ways the Web is not similar to a database system: it has no uniform structure, no integrity

[3] http://www.w3.org/XML/
[4] http://www.w3.org/RDF/
[5] http://www.w3.org/TR/rdf-schema/
[6] http://www.w3.org/2004/OWL/

constraints, no support for transaction processing, no management capabilities, no standard query language, or data model. Perhaps the most popular data model for the Web is the labelled graph, where nodes represent Web pages (or internal components of pages) and arcs correspond to links. Labels on the arcs can be viewed as attribute names for the nodes. The lack of structure in Web pages has motivated the use of semistructured data techniques, which also facilitate the exchange of information between heterogeneous sources. Abiteboul [1] suggests the following features for a semistructured data query language: standard relational database operations (using an SQL viewpoint), navigational capabilities in the hypertext/Web style, information retrieval influenced search using patterns, temporal operations, and the ability to mix data and schema (type) elements together in queries. Many languages support regular path expressions over the graph for stating navigational queries along arcs. The inclusion of wild cards allows arbitrarily deep data and cyclic structures to be searched, although restrictions must be applied to prevent looping.

Queries can be posed to Web pages with XML or RDF content. XML is a notation for describing labelled ordered trees with references. Specifying a query language for XML has been an active area of research, much of it coordinated by the XML Query Working Group of the W3C. The suggested features for such a language are almost identical to those for querying semistructured data. It is hardly surprising that most proposals adopt models which view XML as an edge-labelled directed graph, and use semistructured data query languages. The main difference is that the elements in an XML document are sometimes ordered. The XPath language[7] is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. Conversely, XQuery[8] is a query and functional programming language that is designed to query collections of XML data. XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases. Therefore it finally supports the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases. XPath 2.0 is in fact a subset of XQuery 1.0.

RDF is an application of XML aimed at facilitating the interoperability of meta-data across heterogeneous hosts. With RDF, the most suitable approach is to focus on the underlying data model. Even though XQuery could be used to query RDF descriptions in their XML encoded form, a single RDF data model could not be correctly determined with a single query due to the fact that RDF allows several XML syntax encodings for the same data model. Conceived to address this issue, **Metalog** is a LP language where facts and rules are translated and stored as RDF statements [73,71]. Facts are treated as RDF triples, while rule syntax is supported with additional RDFS statements for LP elements such as head, body, if and variable. A query language for RDF, called SPARQL[9],

[7] `http://www.w3.org/TR/xpath20/`
[8] `http://www.w3.org/TR/xquery/`
[9] `http://www.w3.org/TR/rdf-sparql-query/`

has been recommended by the RDF Data Access Working Group of the W3C in 2008. Also it has been proved that SPARQL and non-recursive safe DATALOG$^{\neg}$ have equivalent expressive power, and hence, by classical results, SPARQL is equivalent from an expressive point of view to Relational Algebra [6]. A LP-based rule system for querying persistent RDFS data is suggested in [58] as an alternative to SPARQL engines.

3.2 LP for Web Computation

The ability to support the execution of logic and constraint programs on parallel and distributed architectures have prompted LP researchers to consider some natural generalization of these programming paradigms to suit the needs of some specific application areas among which the Web.

The concurrent constraint-based LP language **W-ACE** has explicit support for the Web computation [83]. Some of its novel ideas include representing Web pages as LP trees and the use of constraints to manipulate tree components and the relationship between trees. W-ACE also contains modal operators for reasoning about groups of pages, and composition operators very similar to those in LogicWeb [69].

In [82] the author studies the use of distributed logic programming models to provide a natural concurrent framework for Web programming. A concurrent logic-based framework (called **WEB-KLIC**) has already been developed and is currently publicly distributed as part of the ICOT Free Software Project[10]. A relevant component of this part of the project includes the design of constraint domains for representing HTML and XML documents. Also, a primary goal has been the improvement of its CGI facilities (i.e., for server-side computation).

3.3 LP for Knowledge Representation in the Semantic Web

The advent of the Semantic Web has given a tremendous impulse on research in Knowledge Representation (KR) due to the key role played by ontologies in the Semantic Web architecture. Indeed the design of OWL has been based on KR formalisms known as *Description Logics* (DLs) [7], more precisely on the $\mathcal{SH}$ family of the so-called very expressive DLs [56]. DLs are a family of decidable First Order Logic (FOL) fragments that allow for the specification of knowledge in terms of classes (*concepts*), binary relations between classes (*roles*), and instances (*individuals*). Complex concepts can be defined from atomic concepts and roles by means of constructors such as atomic negation ($\neg$), concept conjunction ($\sqcap$), value restriction ($\forall$), and limited existential restriction ($\exists$) - just to mention the basic ones. A DL KB can state both is-a relations between concepts (*axioms*) and instance-of relations between individuals (resp. couples of individuals) and concepts (resp. roles) (*assertions*). Concepts and axioms form the so-called TBox whereas individuals and assertions form the so-called ABox. An $\mathcal{SH}$ KB encompasses also a RBox, i.e. axioms defining hierarchies over roles.

[10] `http://www.jipdec.or.jp/icot/ARCHIVE/Museum/IFS/`

The semantics of DLs can be defined through a mapping to FOL. Thus, coherently with the *Open World Assumption* (OWA) that holds in FOL semantics, a DL KB represents all its models. The main reasoning task for a DL KB is the *consistency check* that is performed by applying decision procedures mostly based on tableau calculus. Summing up, when a DL-based ontology language is adopted, an ontology is nothing else than a TBox eventually coupled with a RBox. If the ontology is populated, it corresponds to a whole DL KB, i.e. encompassing also an ABox.

The Semantic Web architecture poses several challenges to KR like (i) the scalability of ontology reasoning, and (ii) the integration of rules and ontologies. It turns out that LP can help facing these challenges, as explained in the following subsections, though Italian research has focused more on the latter challenge.

DL reasoning with LP

A second round of standardization at W3C has very recently delivered OWL 2[11] which now includes several profiles (or fragments) that can be more simply and/or efficiently implemented than the former OWL proposal. E.g., the OWL 2 RL profile is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade the full expressivity of the language for efficiency, and RDFS applications that need some added expressivity from OWL 2. This is achieved by defining a syntactic subset of OWL 2 which is amenable to implementation using rule-based technologies such as LP. The design of OWL 2 RL has been inspired by Description Logic Programs [55] which are at the intersection of DLs and Datalog. Yet the influence of LP tradition on the implementation of DL systems is also testified by, e.g., KAON2[12] and DLog[13].

Contrary to most currently available DL reasoners, **KAON2** does not implement the tableaux calculus [57]. Rather, it implements novel algorithms which reduce an $\mathcal{SHIQ}$ KB to a disjunctive Datalog program. These algorithms allow applying well-known deductive database techniques, such as magic sets or join-order optimizations, to DL reasoning, thus making answering queries in KAON2 one or more orders of magnitude faster than in existing systems.

DLog is a DL ABox reasoner that uses resolution [70]. It performs query-driven execution whereby the terminological part of the DL KB is converted into a Prolog program using a specialisation of the PTTP Theorem Proving approach and the assertional facts are accessed dynamically from a database. DLog 2 will ensure scalability by specialising well-established LP techniques for parallel computation and efficiency by using an ad-hoc abstract machine.

Rule Systems combining LP and DLs

Rules are currently in the focus within the Semantic Web architecture, and consequently interest and activity in this area has grown rapidly over recent years. They would allow the integration, transformation and derivation of data

[11] `http://www.w3.org/TR/owl2-overview/`
[12] `http://kaon2.semanticweb.org/`
[13] `http://www.dlog-reasoner.org/`

from numerous sources in a distributed, scalable, and transparent manner. The rules landscape features design aspects of rule markup; engineering of engines, translators, and other tools; standardization efforts, such as the recent Rules Interchange Format (RIF)[14] activity at W3C; and applications. Rules complement and extend *ontologies* on the Semantic Web. They can be used in combination with ontologies, or as a means to specify ontologies. Rules are also frequently applied over ontologies, to draw inferences, express constraints, specify policies, react to events, discover new knowledge, transform data, etc. Rule markup languages enrich Web ontologies by supporting publishing rules on the Web, exchange rules between different systems and tools, share guidelines and policies, merge and maintain rulebases, and more.

The debate around a RIF is still ongoing. Because of the great variety in rule languages and rule engine technologies, this format will consist of a core language to be used along with a set of standard and non-standard extensions. These extensions need not all be combinable into a single unified language. As for the expressive power, two directions are followed: monotonic extensions towards full FOL and non-monotonic extensions based on the LP tradition, i.e. on *Clausal Logics* (CLs). In particular, the latter will most likely be the so-called *hybrid systems* that integrate DLs and (fragments of) CLs. These KR systems are constituted by two or more subsystems dealing with distinct portions of a single KB by performing specific reasoning procedures [34]. The motivation for investigating and developing such systems is to improve on two basic features of KR formalisms, namely *representational adequacy* and *deductive power*, by preserving the other crucial feature, i.e. *decidability*. Indeed DLs and CLs are FOL fragments incomparable as for the expressiveness [13] and the semantics [85] but combinable at different degrees of integration: tight, loose, full.

The semantic integration is *tight* when a model of the hybrid KB is defined as the union of two models, one for the DL part and one for the CL part, which share the same domain. In particular, combining DLs with CLs in a tight manner can easily yield to undecidability if the interaction scheme between the DL and the CL part of a hybrid KB does not fulfill the condition of *safeness*, i.e. does not solve the semantic mismatch between DLs and CLs [86]. E.g., the hybrid KR system CARIN is *unsafe* [63] because the interaction scheme is left unrestricted. Conversely, $\mathcal{AL}$**-log** [24] is a *safe* hybrid KR system that integrates Datalog with the DL $\mathcal{ALC}$ [94]. In particular, variables occurring in the body of rules may be constrained with $\mathcal{ALC}$ concept assertions to be used as "typing constraints". This makes rules applicable only to explicitly named objects. As in CARIN, query answering is decided using the constrained SLD-resolution which however in $\mathcal{AL}$-log is decidable and runs in single non-deterministic exponential time. The hybrid KR framework of $\mathcal{DL}$+**log** [87] allows for the *weakly-safe* integration of DATALOG$^{\neg\vee}$ with any DL. The condition of weak safeness allows to overcome the main representational limits of the approaches based on the DL-safeness condition, e.g. the possibility of expressing a *union of conjunctive queries* (UCQ), by keeping the integration scheme still decidable. Apart from the FOL semantics,

[14] `http://www.w3.org/2005/rules/`

$\mathcal{DL}$+log has a NM semantics obtained by extending the stable model semantics. According to it, DL-predicates are still interpreted under OWA, while Datalog predicates are interpreted under CWA. The problem statement of satisfiability for finite $\mathcal{DL}$+log KBs relies on the problem known as the *Boolean CQ/UCQ containment problem* in DLs. It is shown that the decidability of reasoning in $\mathcal{DL}$+log, thus of ground query answering, depends on the decidability of the Boolean CQ/UCQ containment problem in $\mathcal{DL}$.

The semantic integration is *loose* when the DL part and the CL part are separate components connected through a minimal interface for exchanging knowledge. An example of one such kind of coupling is the integration scheme for ASP and DLs illustrated in [28]. It derives from the previous work of the same authors on the extension of ASP with higher-order reasoning and external evaluations [29] which has been implemented into the system DLVHEX[15].

The semantic integration is *full* when there is no separation between vocabularies of the two parts of the hybrid KB. In [76], the authors introduce a so-called faithful integration scheme of LP with DLs using the logic of Minimal Knowledge and Negation as Failure (MKNF).

A complete picture of the computational properties of systems combining DL ontologies and Datalog rules can be found in [88]. An updated survey of the literature on hybrid DL-CL systems [26] is suggested for further reading.

3.4 LP for Learning Semantic Web Ontologies and Rules

The advent of the Semantic Web has also raised a knowledge acquisition bottleneck problem for ontologies and rules. Some promising solutions to this problem come from that Machine Learning approach known under the name of Inductive Logic Programming (ILP).

Rooted into LP, the methodological apparatus of ILP inherits the inferential mechanisms for induction from Concept Learning, the most prominent of which is *generalization* [78]. In Concept Learning, thus in ILP, generalization is traditionally viewed as search through a partially ordered space of inductive hypotheses [74]. According to this vision, an inductive hypothesis is a clausal theory and the induction of a single clause requires (i) structuring, (ii) searching and (iii) bounding the space of clauses. In order to achieve (i), a *generality order* is imposed on clauses for determining which one, between two clauses, is more general than the other. Since partial orders are considered, uncomparable pairs of clauses are admitted. Once structured, the space of hypotheses can be searched (ii) by means of refinement operators. A *refinement operator* is a function which computes a set of specializations or generalizations of a clause according to whether a top-down or a bottom-up search is performed. The two kinds of refinement operators have been therefore called *downward* and *upward*, respectively. The definition of refinement operators presupposes the investigation of the properties of the various orderings and is usually coupled with the specification of a *declarative bias* for bounding the space of clauses (iii). This

[15] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

concerns anything which constrains the search for theories, e.g. a *language bias* specifies syntactic constraints on the clauses in the search space.

A distinguishing feature of ILP with respect to other forms of Concept Learning is the use of background knowledge (BK), i.e. prior knowledge of the domain of interest, during the induction process. Therefore, an ILP algorithm generalizes from individual instances/observations in the presence of BK, finding valid hypotheses. *Validity* depends on the underlying *setting*. At present, there exist several settings in ILP that vary according to: (i) the *scope of induction* (prediction vs description) and (ii) the *representation of observations* (ground definite clauses vs ground unit clauses). *Prediction* aims at inducing hypotheses with discriminant power as required in tasks such as classification where observations encompass both positive and negative examples. *Description* is more suitable for finding regularities in a data set. This corresponds to learning from positive examples only. Aspect (ii) affects the notion of *coverage*, i.e. the condition under which a hypothesis explains an observation. In *learning from entailment*, hypotheses are clausal theories, observations are ground definite clauses, and a hypothesis covers an observation if the hypothesis logically entails the observation. In *learning from interpretations*, hypotheses are clausal theories, observations are Herbrand interpretations (ground unit clauses) and a hypothesis covers an observation if the observation is a model for the hypothesis.

ILP has been historically concerned with Concept Learning from examples and BK within the representation framework of Horn CL and with the aim of prediction. More recently ILP has considered the problems of learning in different FOL fragments such as DLs and hybrid DL-CL formalisms. This bunch of ILP research is relevant to the Semantic Web application domain.

Inducing DL Concept Descriptions

An ILP characterization of the problem has been proposed by [8,62]. Contributions from the Italian LP community are on the formal treatment of learning in DLs, e.g.: Supervised learning in $\mathcal{ALC}$ [30]; Unsupervised learning (concept formation) in $\mathcal{ALC}$ [32]; Supervised learning in OWL DL [31].

Inducing Hybrid DL-CL Rules

Only three ILP frameworks have been proposed that adopt a hybrid DL-CL representation for both hypotheses and background knowledge: [89] chooses CARIN-$\mathcal{ALN}$, [64] resorts to $\mathcal{AL}$-log, and [65] builds upon $\mathcal{SHIQ}$+log. They can be considered as attempts at accommodating ontologies in ILP by having ontologies as BK. Indeed both proposals extend previous work in ILP, notably the order of generalized subsumption [14], to hybrid DL-CL KR frameworks [66].

The framework proposed in [89] focuses on discriminant induction and adopts the ILP setting of learning from interpretations. Hypotheses are represented as CARIN-$\mathcal{ALN}$ non-recursive rules with a Horn literal in the head that plays the role of target concept. The coverage relation adapts the usual one in the ILP setting of learning from interpretations to the case of hybrid CARIN-$\mathcal{ALN}$ BK. Procedures for testing both the coverage relation and the generality relation are based on the existential entailment algorithm of CARIN. In [59], the author

studies the learnability of CARIN-$\mathcal{ALN}$ and provides a pre-processing method which enables traditional ILP systems to learn CARIN-$\mathcal{ALN}$ rules.

In [64], hypotheses are represented as $\mathcal{AL}$-log rules. As opposite to [89], this framework is general, meaning that it is valid whatever the scope of induction (prediction/description) is. Therefore the literal in the head of hypotheses represents a concept to be either discriminated from others or characterized. The generality order for one such hypothesis language can be checked with a decidable procedure based on constrained SLD-resolution. Coverage relations for both ILP settings of learning from interpretations and learning from entailment have been defined on the basis of query answering in $\mathcal{AL}$-log. As opposite to [89], the framework has been implemented into an ILP system that supports a variant of a very popular data mining task - frequent pattern discovery - where rich prior conceptual knowledge is taken into account during the discovery process in order to find patterns at multiple levels of description granularity [67].

The ILP framework presented in [65] represents hypotheses as $\mathcal{DL}$+log rules restricted to the DL $\mathcal{SHIQ}$ and positive Datalog. Analogously to [64], it encompasses both scopes of induction but, differently from [64], it assumes the ILP setting of learning from interpretations only. Both the coverage relation and the generality relation boil down to query answering in $\mathcal{DL}$+log. Refinement operators are defined to search the hypothesis space either top-down or bottom-up. Compared to [89] and [64], this framework shows an added value which can be summarized as follows. First, it relies on a more expressive DL, i.e. $\mathcal{SHIQ}$. Second, it allows for inducing definitions for new DL concepts, i.e. rules with a $\mathcal{SHIQ}$ literal in the head. Third, it adopts a more expressive integration scheme of DLs and CLs, i.e. the weakly-safe one.

References

1. Abiteboul, S.: Querying semi-structured data. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 1–18. Springer, Heidelberg (1996)
2. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, San Francisco (2000)
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
4. Abiteboul, S., Simon, E.: Fundamental Properties of Deterministic and Nondeterministic Extensions of Datalog. Theoretical Compututer Science 78(1), 137–158 (1991)
5. Abiteboul, S., Vianu, V.: Non-Determinism in Logic-Based Languages. Annals of Mathematics and Artificial Intelligence 3(2-4), 151–186 (1991)
6. Angles, R., Gutierrez, C.: The Expressive Power of SPARQL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008)
7. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications, 2nd edn. Cambridge University Press, Cambridge (2007)
8. Badea, L., Nienhuys-Cheng, S.-W.: A Refinement Operator for Description Logics. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 40–59. Springer, Heidelberg (2000)

9. Beeri, C., Ramakrishnan, R.: On the Power of Magic. Journal of Logic Programming 10(1-4), 255–299 (1991)
10. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
11. Bidoit, N., Maabout, S.: A Model Theoretic Approach to Update Rule Programs. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 173–187. Springer, Heidelberg (1996)
12. Bonatti, P., Calimeri, F., Leone, N., Ricca, F.: Answer Set Programming. In: Dovier, Pontelli [25], ch. 8, vol. 6125, pp. 159–178 (2010)
13. Borgida, A.: On the Relative Expressiveness of Description Logics and Predicate Logics. Artificial Intelligence 82(1-2), 353–367 (1996)
14. Buntine, W.: Generalized Subsumption and Its Applications to Induction and Redundancy. Artificial Intelligence 36(2), 149–176 (1988)
15. Calì, A., Gottlob, G., Lukasiewicz, T.: Tractable Query Answering over Ontologies with Datalog+/-. In: Description Logics (2009)
16. Calimeri, F., Faber, W., Leone, N., Perri, S.: Declarative and Computational Properties of Logic Programs with Aggregates. In: IJCAI, pp. 406–411 (2005)
17. Caroprese, L., Greco, S., Sirangelo, C., Zumpano, E.: Declarative Semantics of Production Rules for Integrity Maintenance. In: Etalle, S., Truszczyński, M. (eds.) ICLP 2006. LNCS, vol. 4079, pp. 26–40. Springer, Heidelberg (2006)
18. Caroprese, L., Greco, S., Zumpano, E.: Active Integrity Constraints for Database Consistency Maintenance. IEEE Transactions on Knowledge and Data Engineering 21(7), 1042–1058 (2009)
19. Caroprese, L., Truszczyński, M.: Declarative Semantics for Active Integrity Constraints. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 269–283. Springer, Heidelberg (2008)
20. Caroprese, L., Truszczyński, M.: Declarative Semantics for Revision Programming and Connections to Active Integrity Constraints. In: Hölldobler, S., Lutz, C., Wansing, H. (eds.) JELIA 2008. LNCS (LNAI), vol. 5293, pp. 100–112. Springer, Heidelberg (2008)
21. Ceri, S., Fraternali, P., Paraboschi, S., Tanca, L.: Automatic Generation of Production Rules for Integrity Maintenance. ACM Transactions on Database Systems 19(3), 367–422 (1994)
22. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (1990)
23. Ceri, S., Gottlob, G., Wiederhold, G.: Efficient Database Access from Prolog. IEEE Transaction on Software Engineering 15(2), 153–164 (1989)
24. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: $\mathcal{AL}$-log: Integrating Datalog and Description Logics. J. of Intelligent Information Systems 10(3), 227–252 (1998)
25. Dovier, A., Pontelli, E. (eds.): 25 Years of Logic Programming in Italy. LNCS, vol. 6125. Springer, Heidelberg (2010)
26. Drabent, W., Eiter, T., Ianni, G.B., Krennwallner, T., Lukasiewicz, T., Maluszynski, J.: Hybrid Reasoning with Rules and Ontologies. In: REWERSE, pp. 1–49 (2009)
27. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems 22(3), 364–418 (1997)
28. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. Artificial Intelligence 172(12-13), 1495–1539 (2008)

29. Eiter, T., Ianni, G.B., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: IJCAI, pp. 90–96 (2005)
30. Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-Intensive Induction of Terminologies from Metadata. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004)
31. Fanizzi, N., d'Amato, C., Esposito, F.: DL-FOIL Concept Learning in Description Logics. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 107–121. Springer, Heidelberg (2008)
32. Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Concept Formation in Expressive Description Logics. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 99–110. Springer, Heidelberg (2004)
33. Flesca, S., Greco, S.: Declarative semantics for active rules. Theory and Practice of Logic Programming 1(1), 43–69 (2001)
34. Frisch, A.M., Cohn, A.G.: Thoughts and Afterthoughts on the 1988 Workshop on Principles of Hybrid Reasoning. AI Magazine 11(5), 84–87 (1991)
35. Furfaro, F., Greco, G., Greco, S.: Minimal founded semantics for disjunctive logic programs and deductive databases. Theory and Practice of Logic Programming 4(1-2), 75–93 (2004)
36. Furfaro, F., Greco, S., Ganguly, S., Zaniolo, C.: Pushing extrema aggregates to optimize logic queries. Information Systems 27(5), 321–343 (2002)
37. Gallaire, H., Minker, J. (eds.): Logic and Data Bases. Plenum Press, New York (1978)
38. Gallaire, H., Minker, J., Nicolas, J.M.: Logic and databases: A deductive approach. ACM Computing Surveys 16(2), 153–185 (1984)
39. Gallaire, H., Nicolas, J.M., Minker, J. (eds.): Advances in Data Base Theory, vol. 2. Plenum Press, New York (1984)
40. Ganguly, S., Greco, S., Zaniolo, C.: Extrema Predicates in Deductive Databases. Journal of Computer and Systems Science 51(2), 244–259 (1995)
41. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. J. ACM 38(3), 620–650 (1991)
42. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP, pp. 1070–1080 (1988)
43. Giannotti, F., Greco, S., Saccà, D., Zaniolo, C.: Programming with Non-Determinism in Deductive Databases. Annals of Mathematics and Artificial Intelligence 19(1-2), 97–125 (1997)
44. Giannotti, F., Pedreschi, D., Saccà, D., Zaniolo, C.: Non-Determinism in Deductive Databases. In: Delobel, C., Masunaga, Y., Kifer, M. (eds.) DOOD 1991. LNCS, vol. 566, pp. 129–146. Springer, Heidelberg (1991)
45. Giannotti, F., Pedreschi, D., Zaniolo, C.: Semantics and Expressive Power of Nondeterministic Constructs in Deductive Databases. Journal of Computer and Systems Science 62(1), 15–42 (2001)
46. Giordano, L., Toni, F.: Knowledge representation and non-monotonic reasoning. In: Dovier, Pontelli [25], ch. 5, vol. 6125, pp. 86–110 (2010)
47. Gozzi, F., Lugli, M., Ceri, S.: An overview of PRIMO: a portable interface between PROLOG and relational databases. Information Systems 15(5), 543–553 (1990)
48. Greco, S.: Dynamic Programming in Datalog with Aggregates. IEEE Transactions on Knowledge and Data Engineering 11(2), 265–283 (1999)

49. Greco, S., Saccà, D.: Complexity and Expressive Power of Deterministic Semantics for DATALOG. Information and Computation 153(1), 81–98 (1999)
50. Greco, S., Saccà, D., Zaniolo, C.: Datalog Queries with Stratified Negation and Choice: from p to d^p. In: Y. Vardi, M., Gottlob, G. (eds.) ICDT 1995. LNCS, vol. 893, pp. 82–96. Springer, Heidelberg (1995)
51. Greco, S., Saccà, D., Zaniolo, C.: The PushDown Method to Optimize Chain Logic Programs. In: Fülöp, Z., Gecseg, F. (eds.) ICALP 1995. LNCS, vol. 944, pp. 523–534. Springer, Heidelberg (1995)
52. Greco, S., Saccà, D., Zaniolo, C.: Grammars and Automata to Optimize Chain Logic Queries. Int. Journal Foundations of Computer Science 10(3), 349 (1999)
53. Greco, S., Zaniolo, C.: Optimization of Linear Logic Programs Using Counting Methods. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 72–87. Springer, Heidelberg (1992)
54. Greco, S., Zaniolo, C.: Greedy algorithms in Datalog. Theory and Practice of Logic Programming 1(4), 381–407 (2001)
55. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: WWW, pp. 48–57 (2003)
56. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics 1(1), 7–26 (2003)
57. Hustadt, U., Motik, B., Sattler, U.: Deciding expressive description logics in the framework of resolution. Information and Computation 206(5), 579–601 (2008)
58. Ianni, G.B., Krennwallner, T., Martello, A., Polleres, A.: A Rule System for Querying Persistent RDFS Data. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 857–862. Springer, Heidelberg (2009)
59. Kietz, J.-U.: Learnability of Description Logic Programs. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 117–132. Springer, Heidelberg (2003)
60. Krishnamurthy, R., Naqvi, S.A.: Non-Deterministic Choice in Datalog. In: JCDKB, pp. 416–424 (1988)
61. Lausen, G., Ludäscher, B., May, W.: On Logical Foundations of Active Databases. In: Logics for Databases and Information Systems, pp. 389–422 (1998)
62. Lehmann, J., Hitzler, P.: Foundations of Refinement Operators for Description Logics. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 161–174. Springer, Heidelberg (2008)
63. Levy, A.Y., Rousset, M.-C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104, 165–209 (1998)
64. Lisi, F.A.: Building Rules on Top of Ontologies for the Semantic Web with Inductive Logic Programming. Theory and Practice of Logic Programming 8(03), 271–300 (2008)
65. Lisi, F.A., Esposito, F.: Foundations of Onto-Relational Learning. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 158–175. Springer, Heidelberg (2008)
66. Lisi, F.A., Esposito, F.: On Ontologies as Prior Conceptual Knowledge in Inductive Logic Programming. In: Knowledge Discovery Enhanced with Semantic and Social Information, pp. 3–18. Springer, Heidelberg (2009)
67. Lisi, F.A., Malerba, D.: Inducing Multi-Level Association Rules from Multiple Relations. Machine Learning 55, 175–210 (2004)

68. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
69. Loke, S.W., Davison, A.: LogicWeb: Enhancing the Web with Logic Programming. Journal of Logic Programming 36(3), 195–240 (1998)
70. Lukácsy, G., Szeredi, P.: Efficient Description Logic Reasoning in Prolog: The DLog system. Theory and Practice of Logic Programming 9(3), 343–414 (2009)
71. Marchiori, M.: Towards a people's web: Metalog. In: Web Intelligence, pp. 320–326. IEEE Computer Society Press, Los Alamitos (2004)
72. Marchiori, M.: Introduction to the Special Issue on Logic Programming and the Web. Theory and Practice of Logic Programming 8(3), 247–248 (2008)
73. Marchiori, M., Saarela, J.: Query + Metadata + Logic = Metalog. In: W3C Workshop on Query Languages (1998)
74. Mitchell, T.M.: Generalization as Search. Artificial Intelligence 18, 203–226 (1982)
75. Motakis, I., Zaniolo, C.: Temporal Aggregation in Active Database Rules. In: SIGMOD Conference, pp. 440–451 (1997)
76. Motik, B., Rosati, R.: A Faithful Integration of Description Logics with Logic Programming. In: IJCAI, pp. 477–482 (2007)
77. Mumick, I.S., Shmueli, O.: How Expressive is Statified Aggregation? Annals of Mathematics and Artificial Intelligence 15(3-4), 407–434 (1995)
78. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. Springer, Heidelberg (1997)
79. Palopoli, L., Torlone, R.: Generalized Production Rules as a Basis for Integrating Active and Deductive Databases. IEEE Transactions on Knowledge and Data Engineering 9(6), 848–862 (1997)
80. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. Theory and Practice of Logic Programming 7(3), 301–353 (2007)
81. Picouet, P., Vianu, V.: Semantics and Expressiveness Issues in Active Databases. Journal of Computer and Systems Science 57(3), 325–355 (1998)
82. Pontelli, E.: Concurrent Web-Programming in CLP(WEB). In: HICSS (2000)
83. Pontelli, E., Gupta, G.: W-ACE: A Logic Language for Intelligent Internet Programming. In: IEEE ICTAI, pp. 2–10 (1997)
84. Przymusinski, T.C.: Semantics of Disjunctive Logic Programs and Deductive Databases. In: Delobel, C., Masunaga, Y., Kifer, M. (eds.) DOOD 1991. LNCS, vol. 566, pp. 85–107. Springer, Heidelberg (1991)
85. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. Journal of Web Semantics 3(1), 61–73 (2005)
86. Rosati, R.: Semantic and Computational Advantages of the Safe Integration of Ontologies and Rules. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 50–64. Springer, Heidelberg (2005)
87. Rosati, R.: $\mathcal{DL}$+log: Tight Integration of Description Logics and Disjunctive Datalog. In: KR, pp. 68–78 (2006)
88. Rosati, R.: On Combining Description Logic Ontologies and Nonrecursive Datalog Rules. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 13–27. Springer, Heidelberg (2008)
89. Rouveirol, C., Ventos, V.: Towards Learning in CARIN-$\mathcal{ALN}$. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 191–208. Springer, Heidelberg (2000)
90. Saccà, D.: The Expressive Powers of Stable Models for Bound and Unbound DATALOG Queries. Journal of Computer System Sciences 54(3), 441–464 (1997)

91. Saccà, D., Zaniolo, C.: The Generalized Counting Method for Recursive Logic Queries. Theoretical Computer Science 62(1-2), 187–220 (1988)
92. Saccà, D., Zaniolo, C.: Stable Models and Non-Determinism in Logic Programs with Negation. In: PODS, pp. 205–217 (1990)
93. Saccà, D., Zaniolo, C.: Deterministic and Non-Deterministic Stable Models. Journal of Logic and Computation 7(5), 555–579 (1997)
94. Schmidt-Schauss, M., Smolka, G.: Attributive Concept Descriptions with Complements. Artificial Intelligence 48(1), 1–26 (1991)
95. Son, T.C., Pontelli, E., Elkabani, I.: An unfolding-based semantics for logic programming with aggregates. CoRR, abs/cs/0605038 (2006)
96. Tsur, S.: Deductive Databases in Action. In: PODS, pp. 142–153 (1991)
97. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. II. Computer Science Press (1989)
98. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press (1988)
99. Widom, J.: Deductive and Active Databases: Two Paradigms or Ends of a Spectrum? In: Rules in Database Systems, pp. 306–315 (1993)
100. Widom, J., Ceri, S. (eds.): Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, San Francisco (1996)
101. Zaniolo, C.: The Nonmonotonic Semantics of Active Rules in Deductive Databases. In: Bry, F., Ramamohanarao, K. (eds.) DOOD 1997. LNCS, vol. 1341, pp. 265–282. Springer, Heidelberg (1997)