Datalog Relaunched: Simulation Unification and Value Invention

François Bry[1], Tim Furche[2], Clemens Ley[2],
Bruno Marnette[2], Benedikt Linse[3], and Sebastian Schaffert[4]

[1] Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany
[2] Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
[3] Thomson Reuters, Landsberger Straße 191a, 80687 München, Germany
[4] Knowledge and Media Technologies, Salzburg Research
Jakob Haringer Str. 5/III, 5020 Salzburg, Austria

Abstract. For reasoning on the Web, Datalog is lacking data extraction and value invention. This article proposes to overcome these limitations with "simulation unification" and "RDFLog".
Simulation unification is a non-standard unification inspired from regular path queries. Like standard unification, it yields bindings for variables in both terms to unify. Unlike standard unification, it does not try to make the two terms identical but instead to embed the query into the data. Simulation unification is decidable. Without variables, it has polynomial complexity. With variables it is, like standard unification, NP-complete. We identify a number of interesting special cases of unification, e.g., in presence or absence of term injectivity. In particular, we show that simulation unification without term injectivity on tree data is linear and in presence of injectivity it is still polynomial even on unordered trees in contrast to the NP-complete unordered tree inclusion problem.
RDFLog is Datalog with arbitrary quantifier alternation: Blank nodes, i.e., existentially quantified variables, in rule heads may be governed by universally quantified variables, universally quantified variables by blank nodes. RDFLog's declarative semantics is defined in terms of RDF entailment; its sound and complete operational semantics, in terms of Skolemization, standard Datalog evaluation, and un-Skolemization. We show that RDFLog limited to $\forall^*\exists^*$ prefixes is (up to unique helper predicates) equivalent to RDFLog with full quantifier alternation. A lightweight implementation points to the efficiency of the approach.

Keywords: Datalog, Unification, Value Invention, Un-Skolemization, Query Language, Regular Path Queries, Semi-Structured Data, HTML, XML, XPath, RDF.

1 Introduction

Datalog is a fragment of the logic programming language Prolog [87] aiming at combining rule-based reasoning with relational databases. Datalog is declarative,

or "pure", in the sense that it includes none of the procedural features of Prolog such as a pre-defined evaluation order and language constructs such as the cut for modifying this order. The integration of Prolog-style rule-based reasoning and databases has been first advertised at the end of the 70es, beginning of the 80es of the 20th century in three workshops on "Logic and Databases" [62] and on "Advances in Database Theories" [64,65] – cf. also [63]. The name Datalog, a contraction of the then widespread expression "Database Prolog", has been coined by David Maier and David S. Warren for lecture notes [91]. Datalog provides first-order queries that, up to the syntax, correspond to SQL queries without negation. Datalog also provides definite rules that correspond to SQL views. Methods have been developed for a set-oriented evaluation of Datalog queries [40], i.e., an evaluation building upon relational algebra and therefore efficiently accessing large amounts of data on secondary storage. Sophisticated methods have been developed for implementing backward chaining relying on forward chaining [112,8,14,117,110,31] (cf. [127,41,42,111] for overviews) thus ensuring a terminating and set-oriented evaluation of recursive Datalog programs. Datalog and the aforementioned evaluation methods have two salient traits:

1. They are restricted to "flat terms",[5] i.e., terms containing no function symbols other than constants.
2. They are restricted to universal facts and rules, i.e., facts and rules where all the variables of which are universally quantified.

Datalog's restriction to flat terms is no stringent limitation if relational data are accessed. Indeed, even though non-first-normal-form relational databases have been considered in research, the focus of relational database technology is on first-normal-form databases that correspond to "flat" logic terms, i.e., logic terms containing no function symbols other than constants. If, however, instead of flat relational data, data on the Web are accessed, then the two aforementioned traits of Datalog must be overcome. Indeed, HTML and XML documents are semi-structured [1], i.e., can be formalized as labelled unranked trees or nested relations. Datalog's restriction to universally quantified variables must be overcome if Datalog is to be used for RDF querying and reasoning. Indeed, RDF graphs might contain so-called blank node, i.e., existentially quantified variables.

This article describes two approaches to adapt Datalog to semi-structured data (such as HTML and XML documents) and to RDF graphs respectively.

The first approach, called "simulation unification", is a non-standard form of unification tuned to data extraction from semi-structured data. Simulation unification is inspired from regular path queries [2,3]. Like standard unification, simulation unification determines bindings for variables in both terms to unify. Unlike standard unification, simulation unification does not make the two terms identical but instead searches for an embedding of the query into the data. Simulation unification is decidable, sound and complete, and has polynomial

[5] The article [117] is an exception: It describes an extension of the magic set method [8,14] to a restricted type of rules with nested terms.

data complexity. Without variables and some incompleteness query constructs, it has polynomial, on tree data even linear time complexity; with variables it is, like standard unification, NP-hard. Simulation unification is closely related to the fragment of XPath [46] with only forward axes, a restriction that does not affect the expressiveness of XPath [103].

The second approach, called RDFLog, is an extension of Datalog with arbitrary quantifier alternation. In RDFLog programs, blank nodes, i.e., existentially quantified variables in rule heads, may occur in the scope of all, some, or none of the universal variables of a rule. In other words, in RDFLog rules, existentially quantified variables may be governed by universally quantified variables, universally quantified variables by existentially quantified variables.

This articles is organized as follows. Section 2 discusses related work. Section 3 describes simulation unification. Section 4 is devoted to RDFLog. Section 5 suggest a notion of "rich unification" as a framework for adapting Datalog to various data types and a direction for further research.

2 Related Work

2.1 Wrapping

Since the end of the 90es of the 20th century, the extraction of data from the Web and from large Thesauri, i.e., from semi-structured documents such as HTML documents, has been investigated from several angles. One commonly distinguishes between text extraction, or text wrapping, and structure extraction, or structure wrapping. Text wrapping is the retrieval of portions of text regardless of the documents' structures. Text wrapping techniques returning so-called "bag of words" is one of the core functions of current search engines and is therefore well-mastered [88]. A more sophisticated form of text wrapping is targeted at so-called "factoids", or entities and relationships between entities, related to a query and extracted from unstructured text [4,70,68,128,129]. Structure wrapping, i.e., the retrieval of structured portions of text such as a section with its sub-sections from a (structured) document is another from of wrapping. Currently, structure wrapping is deployed on the limited scale of Web query and transformation languages such as XPath [46,17], XQuery [19] and XSLT [45]. XPath, which is part of both XQuery and XSLT, can be seen as the most used structure wrapping language.

The approaches to structure wrapping considered so far can be understood by formalizing semi-structured documents as node-labelled unranked trees, i.e., trees such that two nodes similarly labelled do not necessarily have the same number of children.[6] A structure-aware wrapper can be seen as a language for selecting nodes, or equivalently the sub-trees rooted at these nodes, from labelled unranked trees while possibly performing simple changes such as renaming labels and removing some-subtrees. More sophisticated structure reorganizations – such

[6] "Semi-structured" is used instead of "structured" for stressing this characteristic of Web documents as well as the lack of schema [35,1].

as transposing a table, adding sums to table rows or columns, or constructing from a bibliography by years a bibliography by authors, and more generally most forms of data aggregation – are not considered part of data extraction: They are out of the range of structure-aware wrappers.

XPath [46,17], which appeared in 1999 and follows the regular path approach introduced with [2], is the best-known and most used structure wrapper language. It is, however, far from being the only one. Others approach to structure wrapping are as follows: Regular path queries (with constraints) have been proposed in [3]; regular tree languages and (regular tree automata for their evaluation) have been proposed in [30] and further investigated amongst other in [102]; a Datalog-style language called WebOQL has been proposed in [5]; regular path queries with nesting (or RPN) have been proposed in [66,67]; exploiting the fact that regular tree languages coincide with tree languages expressible in monadic second order logic or MSO,[7] MSO is proposed in [66,67] as a reference language for investigations of the expressive power of structure-aware wrapper languages; monadic Datalog, the inspiration of Elog [12], the language of the commercial wrapper Lixto [68], has been proposed in [13,66,67]. More on Web wrapping and Web query languages can be found in the survey [7].

Common to the afore-mentioned proposals is that

1. they are designed for tree-shaped data,
2. the majority, in particular XPath, is designed for querying only for sets of nodes,
3. their query paradigm is navigational.

Being designed for tree-shaped data, they can neither exploit the hypertext links and references within an HTML or XML document, nor fully access structure of RDF graphs. This restriction might not be that significant if standard documents and Web pages are queried. If instead Semantic Web documents such as RDF graphs are queried, the restriction is more significant. Indeed, RDF graphs are almost never tree-shaped. The majority is designed for querying for sets of nodes, but not for sets of tuples of nodes. In logic terms, they are tuned to monadic, i.e., single answer-variable, queries. A navigational query paradigm is quite natural for monadic queries. [69,18,20] stress the drawbacks of navigational queries. Arguably, navigational queries in languages like XPath [46,17] that offer so called "reverse axes" compromise declarativity. Note that reverse axes do not increase XPath's expressive power [103], though it has been recently shown [89] that this does not hold for more expressive path languages containing a Kleene-star type construct (such as conditional XPath [92]).

The language UnQL [36] has introduced simulation as a means for query answering. This has been further investigated with the language XMAS [10]. UnQL and simulation unification differ from each other as follows. First, a query in UnQL can be processed by matching, or "half unification", of a query pattern containing variables with a data item containing no variables. In contrast,

[7] This is a classical result mentioned amongst other in [102].

simulation unification gives rise to unifying two query patterns both containing variables. Furthermore, variables in UnQL can only occur as leaves of query patterns while simulation unification gives rise to (constrained) variables at any depth of a query term.

Simulation unification is not limited to querying tree-shaped data but can instead accommodate graph-shaped queries against graph-shaped data, is not limited to monadic queries but instead can accommodate querying for tuples of nodes, and its paradigm is not navigational but instead, like logic queries, pattern-oriented.

2.2 Rules Languages for the Semantic Web

The considerable literature on Web and Semantic Web rule languages falls generally into four groups:

1. markup for rule languages,
2. implementation of description logics or of RDF-based reasoning in Datalog, in Logic Programming, or in Answer Set Programming,
3. Datalog or Logic Programming style rule languages for RDF,
4. "hybrid reasoning", i.e., integrations of description logic primitives, or built-ins, into Datalog or (Disjunctive) Logic Programming.

The prominent markup languages for rules are the Rule Markup Language RuleML [28,21,22] and the languages, called "dialects", of the Rule Interchange Format RIF [26,24,23,106,25,49,47]. The focus here is on the interchange on the Web relying on XML of rule programs of various kinds. Both have been developed to express various forms of reasoning, especially forward and backward chaining, and, as far as RIF is concerned, production rules. SWRL [77,78,79,76], a rule language integrating sublanguages of OWL [93,123,50,104,39,74,75,97] in RuleML is both, a markup language, and an integration of a description logic in Logic Programming.

The large number of implementations of description logics in Datalog or, more generally, in Logic Programming or Disjunctive Logic Programming, amongst others [15,99,71,11,80,81,82,97], reflects the diversity of description logics. An implementations of RDF/S rule-based reasoning in Answer Set Programming is described in [107].

RDF's syntax makes it rather natural a candidate for Datalog or Logic Programming rules. [85,27,33] are mostly devoted to syntax, markup and interoperability issues of RDF rule languages, [121,130,96,72,124,105,120,34] investigate various forms of rule-based reasoning with RDF/S data. See [61] for a survey on RDF query and rule languages.

Various forms of hybrid reasoning, i.e., integration of description logics into Datalog, Logic Programming, Disjunctive Logic Programming or Answer Set Programming are described amongst others in [78,76,114,113,125,48,79,98,115,55] [54,38,116,53,95,94,52,109,33].

Most of the rule languages considered in the afore-mentioned articles support neither existential variables nor blank nodes in rule heads [121,122,96,58,105].

Some support blank nodes in rule heads but only with limited quantifier alternations [130,72,120]. To the best of the authors' knowledge, existential variable in rule heads with unrestricted quantifier alternation has been first proposed in [34]. The second part of the present paper describes this approach.

3 Simulation Unification: Unification for Web Wrapping

Simulation unification has been developed as a technique for evaluating "query patterns", called in the following "query terms", that are both, in the style of logic queries and better adapted to Web querying. Like logic queries, the query terms might include several variables. Variables in a query term are logic variables, that is, all their occurrences must be consistently bound. In contrast to logic queries, query terms are incomplete specification of the data to retrieve. Query terms may contain constructs for expressing incompleteness in *breadth,* in *depth,* with respect to *order,* and with respect to *optional subterms:*

- Incompleteness in breath: While the query term $a[X, b]$ corresponds to a logic query $a(X, b)$, the query term $a[[X]]$ retrieves a-labelled nodes with *at least one* child (bound to the variable X).
- Incompleteness in depth: The query term $a[\mathsf{desc}\ b]$ retrieves a-labelled nodes with a b-labelled descendant node.
- Incompleteness with respect to order: The order of the matches for b and $\mathsf{desc}\ c$ is irrelevant in the queries $a\{b, \mathsf{desc}\ c\}$ and $a\{\{b, \mathsf{desc}\ c\}\}$.
- Incompleteness with respect to optional subterms: The query $a[b, \mathsf{optional}c[X]]$ binds the variable X to some value only if in the data retrieved the a-labeled node has a c-labelled child having itself a child node (bound to X).

Furthermore, references in query terms and in data are resolved during simulation unification allowing *graph-shaped queries graph-shaped data.* Additional constructs ease the expression of queries frequently needed on the Web. The comparison [29] with the XQuery programs from the XQuery Use Cases [43] demonstrates that these features, as well as a few others described in [119], considerably ease the expression of practical queries.

Definition 1 (Data Terms). Data terms *are expressions inductively defined as follows that satisfy Conditions 1 and 2 below:*

1. *If l is a label, then l is a (atomic) data term.*
2. *If id is an identifier and t is a data term neither ofthe form id_0: t_0 nor ofthe form $\uparrow id_0$, then id: t is a data term.*
3. *If id is an identifier, then $\uparrow id$ is a data term.*
4. *If l is a label and $t_1, \ldots, t_n$ are $n \geq 1$ data terms, then $l[t_1, \ldots, t_n]$ and $l\{t_1, \ldots, t_n\}$ are data terms.*

Condition 1: *For a given identifier id an* identifier definition *id*: t_0 *occurs at most once in a term.*
Condition 2: *For every* identifier reference $\uparrow id$ *occurring in a term t an identifier definition id*: t_0 *occurs in t.*

Definition 2 (Query Terms). Query terms *are expressions inductively defined as follows and satisfying Conditions 1 and 2 of Definition 1:*

1. *If l is a label and L is a label variable, then l, L, $l\{\{\}\}$, and $L\{\{\}\}$ are (atomic) query terms.*
2. *A term variable is a query term.*
3. *If id is an identifier and t is a query term neither ofthe form $id_0\colon t_0$ nor ofthe form $\uparrow id_0$, then $id\colon t$ is a query term.*
4. *If id is an identifier, then $\uparrow id$ is a query term.*
5. *If X is a variable and t a query term, then $X \rightsquigarrow t$ is a query term.*
6. *If X is a variable and t is a query term, then $X \rightsquigarrow$ desc t is a query term.*
7. *If l is a label, L a label variable and $t_1, \ldots, t_n$ are $n \geq 1$ query terms, then $l[t_1, \ldots, t_n]$, $L[t_1, \ldots, t_n]$, $l\{t_1, \ldots, t_n\}$, $L\{t_1, \ldots, t_n\}$, $l[[t_1, \ldots, t_n]]$, $L[[t_1, \ldots, t_n]]$, $l\{\{t_1, \ldots, t_n\}\}$, and $L\{\{t_1, \ldots, t_n\}\}$ are query terms.*

Query terms in which no variables occur are ground. *Query terms that are not of the form $\uparrow id$, are* strict. *The* leftmost label *of strict and ground query terms of the form l, $l\{\{\}\}$, $l\{t_1, \ldots, t_n\}$, and $l[t_1, \ldots, t_n]$ is l; the* leftmost label *of a strict and ground query term of the form $id : t$ is the leftmost label of t.*

Note that *desc* never occurs in a ground query term (for it is always coupled with a variable), data terms are (simple) query terms, in a query term, multiple occurrences of a same variable are possible. *Child subterms* and *subterms* of query terms are defined such that if $t = f[a, g\{Y \rightsquigarrow$ *desc* $b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}]$, then a and $g\{Y \rightsquigarrow$ *desc* $b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}$ are the only child subterms of t and e.g. a and X and $Y \rightsquigarrow$ *desc* $b\{X\}$ and $h\{a, X \rightsquigarrow k\{c\}\}$ and $X \rightsquigarrow k\{c\}$ and t itself are subterms of t. Note that f is not a subterm of t.

We allow in the following a query term desc t without leading $\rightsquigarrow$ as an abbreviation for $X \rightsquigarrow$ desc where X is a fresh variable.

Definition 3 (Variable Well-Formed Query Terms). *A term variable X* depends *on a term variable Y in a query term t if $X \rightsquigarrow t_1$ is a subterm of t and Y is a subterm of t_1. A query term t is* variable well-formed *if t contains no term variables $X_0, \ldots, X_n$ $(n \geq 1)$ such that 1. $X_0 = X_n$ and 2. for all $i = 1, \ldots, n$, X_i depends on X_{i-1} in t.*

Thus, $f\{X \rightsquigarrow g\{X\}\}$ and $f\{X \rightsquigarrow g\{Y\}, Y \rightsquigarrow h\{X\}\}$ are not variable well-formed. Variable well-formedness precludes queries specifying infinite answers. In the following, query terms are assumed to be variable well-formed.

The declarative semantics of simulation unification is based on graph simulation. A simulation of a graph G_1 in a graph G_2 is a mapping of the nodes of G_1 in the nodes of G_2 preserving the edges. The graphs considered are directed, ordered and rooted and their nodes are labelled.

Definition 4 (Simulation). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. Let $\sim$ be an equivalence relation on $V_1 \cup V_2$. A relation $\mathcal{S} \subseteq V_1 V_2$ is a* simulation *with respect to $\sim$ of G_1 in G_2 if:*

1. *If* $(v_1, v_2) \in \mathcal{S}$*, then* $v_1 \sim v_2$*.*
2. *If* $(v_1, v_2) \in \mathcal{S}$ *and* $(v_1, v_1') \in E_1$*, then there exists* $v_2' \in V_2$ *such that* $(v_1', v_2') \in \mathcal{S}$ *and* $(v_2, v_2') \in E_2$*.*

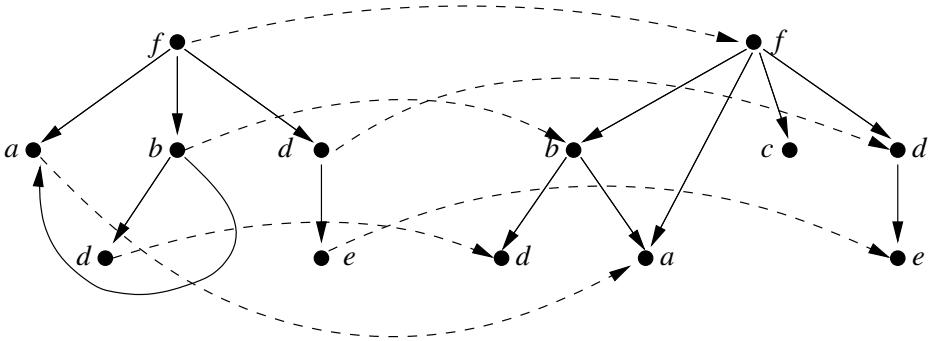
Let $\mathcal{S}$ *be simulation* $\mathcal{S}$ *of* $G_1 = (V_1, E_1)$ *in* $G_2 = (V_2, E_2)$*.* $\mathcal{S}$ *is* total*, if for each* $v_1 \in V_1$ *there exists at least one* $v_2 \in V_2$ *such that* $(v_1, v_2) \in \mathcal{S}$*. If* G_1 *has a root* r_1*,* G_2 *has a root* r_2 *and* $(r_1, r_2) \in \mathcal{S}$*, then* $\mathcal{S}$ *is a* rooted simulation*.* $\mathcal{S}$ *is* minimal*, if there are no simulations* $\mathcal{S}' \subseteq \mathcal{S}$ *of* G_1 *in* G_2 *such that* $\mathcal{S}' \neq \mathcal{S}$*.*

Note that every rooted simulation is total.

Definition 5 (Strict and Ground Query Term Simulation). $\preceq$ *is the relation on strict and ground query terms defined by* $t^1 \preceq t^2$ *if there exists a (minimal) rooted simulation with respect to label identity* $\mathcal{S}$ *of* t^1 *in* t^2 *such that:*

1. *if* $v_1 = l\{\}$ *occurs in* t^1 *and* $(v_1, v_2) \in \mathcal{S}$*, then* v_2 *has no children in* t^2*.*
2. *if* $v_1 = l[[t_1^1, \ldots, t_n^1]]$ $(n \geq 1)$ *occurs in* t^1*,* $(v_1, v_2) \in \mathcal{S}$ *and* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$*, then* $t_1^2, \ldots, t_m^2$ *occur in this indexing order as children of* v_2 *in the graph induced by* t^2*.*
3. *if* $v_1 = l[t_1^1, \ldots, t_n^1]$ $(n \geq 1)$ *occurs in* t^1*,* $(v_1, v_2) \in \mathcal{S}$ *and if* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$*, then* $t_1^2, \ldots, t_m^2$ *are pairwise distinct (i.e.* $m = n$*), they occur in this indexing order as children of* v_2 *in the graph induced by* t^2 *and* v_2 *has no other children than the* t_j^2 *in* t^2*.*
4. *if* $v_1 = l\{t_1^1, \ldots, t_n^1\}$ *occurs in* t^1*,* $(v_1, v_2) \in \mathcal{S}$ *and* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$*, then* $t_1^2, \ldots, t_m^2$ *are pairwise distinct (i.e.* $m = n$*) and* v_2 *has no other children than the* t_j^2 *in* t^2*.*
5. *if* $v_1 = l\{\{t_1^1, \ldots, t_n^1\}\}$ *occurs in* t^1*,* $(v_1, v_2) \in \mathcal{S}$ *and* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$*, then* $t_1^2, \ldots, t_m^2$ *are pairwise distinct (i.e.* $m = n$*).*

Fig. 1 A minimal simulation of the (graph induced by the) ground query term $t^q = f\{\mathrm{id}_1 : a, b[d\{\{\}\}, \uparrow \mathrm{id}_1], desce\}$ in the (graph induced by the) data term $t^{db} = f[b[d, \mathrm{id}_2 : a], \uparrow \mathrm{id}_2, c, d\{e\}]$.



By Definition 4, $\preceq$ is reflexive and transitive, i.e. it is a preorder on the set of data terms. $\preceq$ is not a partial order, for although $t_1 = f\{\{a\}\} \preceq t_2 = f\{\{a, a\}\}$ and $t_2 = f\{\{a, a\}\} \preceq t_1 = f\{\{a\}\}$ (both a of t_2 can be simulated by the same a of t_1), $t_1 = f\{\{a\}\} \neq t_2 = f\{\{a, a\}\}$.

Rooted simulation with respect to label equality is a first step towards a formalisation of answers to query terms: If there exists a rooted simulation of (the graph induced by) a data term t_1, considered as a query term, in (the graph induced by) a data term t_2, then t_2 is an answer to t_1. Ground instances of a query term (cf. Definition 6) gives rise to extend this notion of answers to query terms. An answer in a database D to a query term t^q is characterized by bindings for the variables in t^q such that the database term t resulting from applying these bindings to t^q is an element of D. Consider e.g. the query $t^q = f\{\{\ X \rightsquigarrow g\{\{b\}\}, X \rightsquigarrow g\{\{c\}\}\ \}\}$ against the database $D = \{f\{g\{a, b, c\}, g\{a, b, c\}, h\},\ f\{g\{b\}, g\{c\}\}\}$. The $\rightsquigarrow$ constructs in t^q yield the constraint $g\{\{b\}\} \preceq X \wedge g\{\{c\}\} \preceq X$. Matching t^q with the first data term in D yields the constraint $X \preceq g\{a, b, c\}$. Matching t^q with the second data term in D yields the constraint $X \preceq g\{b\} \wedge X \preceq g\{c\}$. $g\{b\} \preceq X \wedge g\{c\} \preceq X$ is not compatible with $X \preceq g\{b\} \wedge X \preceq g\{c\}$. Thus, the only possible value for X is $g\{a, b, c\}$, i.e. the only possible answer to t^q in D is $f\{g\{a, b, c\}, g\{a, b, c\}, h\}$.

Definition 6 (Ground Instances of Query Terms). *A* grounding substitution *is a function which assigns a label to each label variable and a database term to each term variable of a finite set of (label or term) variables. Let t^q be a query term, $V_1, \ldots, V_n$ be the (label or term) variables occurring in t^q and σ be a grounding substitution assigning v_i to V_i. The* ground instance *$t^q\sigma$ of t^q with respect to σ is the ground query term that can be constructed from t^q as follows:*

1. *Replace each subterm $X \rightsquigarrow t$ by X.*
2. *Replace each occurrence of V_i by v_i $(1 \leq i \leq n)$.*

Requiring in Definition 2 *desc* to occur to the right of $\rightsquigarrow$ makes it possible to characterize a ground instance of a query term by a grounding substitution. This is helpful for formalizing answers but not necessary for language implementations.

Not all ground instances of a query term are acceptable answers, for some instances might violate the conditions expressed by the $\rightsquigarrow$ and *desc* constructs.

Definition 7 (Allowed Instances). *The* constraint induced by a query term *t^q and a substitution σ is the conjunction of all inequalities $t\sigma \preceq X\sigma$ such that $X \rightsquigarrow t$ is a subterm of t^q not of the form desc t_0, and of all expressions $X\sigma \lhd t\sigma$ (read "$t\sigma$ subterm of $X\sigma$") such that $X \rightsquigarrow$ desc t is a subterm of t^q, if t^q has such subterms. If t^q has no such subterms, the constraint induced t^q and σ is the formula true. Let σ be a grounding substitution and $t^q\sigma$ a ground instance of t^q. $t^q\sigma$ is* allowed *if:*

1. *Each inequality $t_1 \preceq t_2$ in the constraint induced by t^q and σ is satisfied.*
2. *For each $t_1 \lhd t_2$ in the constraint induced by t^q and σ, t_2 is a subterm of t_1.*

	Query term	Data terms
T1	`a{ }`	$\preceq$ `a{ }`; `a[ ]` $\not\preceq$ `b{ }`
T2	`a[ ]`	$\preceq$ `a[ ]` $\not\preceq$ `b{ }`; `a{ }`
T3	`a{ b }`	$\preceq$ `a{ b }` $\not\preceq$ `a{ b, b }`
T4	`a{ b, b }`	$\preceq$ `a{ b, b }` $\not\preceq$ `a{ b }`; `a{ b, b, b }`
P1	`a{{ b }}`	$\preceq$ `a{ b }`; `a{ c, b, d }`; `a{ b, b }` $\not\preceq$ `a{ }`;
P2	`a[[ b, c ]]`	$\preceq$ `a[ b, c ]`; `a[ d, b, e, c ]` $\not\preceq$ `a[ c, b ]`; `a{ b, c }`
D1	`a{ desc b }`	$\preceq$ `a[ b ]`; `a[ c{ b, e } ]`; $\not\preceq$ `a{ d, c{ b } }`;
D2	`a{ desc b, desc c }`	$\preceq$ `a[ b, e[ c ] ]`; $\not\preceq$ `a{ b, c, d }`; `a{ e[b, c] }`;
D3	`a{{ desc b, desc c }}`	$\preceq$ `a[ b, e[ c ] ]`; `a{ b, c, d }`; $\not\preceq$ `a{ e[b, c] }`;

Table 2. Query terms and matching data (; separates different data terms)

Definition 8 (Answers). *Let t^q be a query term and D a database. An answer to t^q in D is a database term $t^{db} \in D$ such that there exists an allowed ground instance t of t^q satisfying $t \preceq t^{db}$.*

3.1 Examples of Simulation

Table 2 illustrates the simulation between (variable-free) query terms and data terms. For space reasons, we omit in query terms empty double braces and in data terms empty single braces, i.e., `c` reads `c{{ }}` in a query term and `c{ }` in a data term.

The first examples T1–T4 illustrate matching of ordered and unordered total query terms. Note, that unordered query terms match against ordered data terms (since the use of the curly braces indicates only that we do not care about the order). In total query terms both terms have exactly the same number of children in all cases. This is what sets partial query terms (P1–P2, I1–I2) apart from total query terms. Here, we may have additional query terms in the data that are ignored. The remaining examples of Table 2 illustrate the effect of `desc`. It allows matching at any depth (cf. D1–D3). Totality and injectivity are still enforced between the children of a matching data term (observe the difference between D2 and D3).

The effect of variables on term matching is illustrated in Table 4: Essentially, a variable matches any single term (or label, or position, or node, if so placed),

	Query term	Data terms	Bindings
V1	`a{ var X }`	$\preceq$ `a[ b ];` $\not\preceq$ `a{ }; a[ b, c ]`	$\{X/\texttt{b}\}$
V2	`a{{ var X }}`	$\preceq$ `a[ b, c ];` $\not\preceq$ `a{ };`	$\{X/\texttt{b}, X/\texttt{c}\}$
V3	`a{{ var X, var X }}`	$\preceq$ `a[ b, b ]; a{ c, b, b, d }` $\not\preceq$ `a{ b, c }; a{ b }`	$\{X/\texttt{b}_1, X/\texttt{b}_2\}$
V5	`a{ var X{ var X } }`	$\preceq$ `a[ b{ "b"} ];` $\not\preceq$ `a{ b, c };`	$\{X/\texttt{"b"}\}$
V6	`a{ var X ⇝c, var X }`	$\preceq$ `a[ c, c ];` $\not\preceq$ `a{ b, b };`	$\{X/\texttt{b}\}$
V7	`a{ desc var X }`	$\preceq$ `a[ c{ b, e[ f ] } ];` $\not\preceq$ `a{ d, c{ b } };`	$\{X/\texttt{c}\{\ldots\}, X/\texttt{b}, X/\texttt{e}[\ldots], X/\texttt{f}\}$

Table 4. Query terms containing variables and their bindings

but matches are recorded in the bindings of the query. If a variable occurs multiple times (V3), the matched query terms must be structurally equivalent. A variable may occur as a label (V5), in which case it is bound to the value of the label and can only match with other labels or character data (as the ‘‘b’’ in V5). A variable may occur in a term restriction before $\rightsquigarrow$ (V6), in which case the right hand query term restricts the matching bindings for X. Finally, it can be combined with `desc` yielding the expected result (V7).

3.2 Simulation Unification

Simulation unification is a non-deterministic algorithm for solving in the data term lattice $(\mathcal{T}_{db}/\equiv, \preceq)$ induced by the relation $\preceq$. Inequations of the form $t^q \preceq t^c$, where t^q is a query term, t^c is a so-called "construct term", i.e., a query term without $\rightsquigarrow$ and *desc* constructs.[8] (possibly a data term), and t^q and t^c are variable disjoint.[9] Thus, simulation unification computes substitutions σ such that $t^q\sigma$ and $t^c\sigma$ have instances $t^q\sigma\tau$ and $t^c\sigma\tau$ with $t^q\sigma\tau$ and $t^c\sigma\tau$ data terms and $t^q\sigma\tau \preceq t^c\sigma\tau$.

Simulation unification consists in repeated applications of Term Decomposition phases followed by a Consistency Verification phase to a formula C (for constraint store) consisting in disjunctions of conjunctions of inequations of the form $t^q \preceq t^c$ (with t^q query term and t^c construct term) and/or equations of the form $t^c_1 = t^c_2$ (with t^c_1 and t^c_2 construct terms). At the beginning C consists

[8] Simulation unification is in fact defined for more general construct terms in which grouping and aggregation construts à la $\mathcal{LDL}$ [101,44] might occur.

[9] Variable disjointness is achieved in deduction systems by the so-called "standardization apart."

in a single inequation $t^q \preceq t^c$. Both phases Term Decomposition and Consistency Verification consist in stepwise changes of the constraint store C. These changes are expressed in the following formalism inspired from [59]: A "simplification" $L \Leftrightarrow R$ replaces L by R. Trivially satisfied inequations or equations are replaced by the atomic formula *true*. Inconsistent conjunctions of inequations or equations are replaced by the atomic formula *false*. Memoing ensures that the recursive traversal cyclic query terms terminates. For space reasons, memoing is not discussed in the following, i.e., references in query terms are disregarded, and only the decomposition rules for terms of the form `a{}` or `{{}}` are given. See [118] for a full treatment.

Definition 9 (Term Decomposition Rules). *Let l (with or without indices) denote a label. Let t^1 and t^2 (with or without indices) denote query terms.*

– **Root Elimination:**

(1) $l \preceq l\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \mathit{true}$ *if* $m \geq 1$
$l \preceq l\{\{\}\} \Leftrightarrow \mathit{true}$

(2) $l\{t_1^1, \ldots, t_n^1\} \preceq l \Leftrightarrow \mathit{false}$ *if* $n \geq 1$
$l\{t_1^1, \ldots, t_n^1\} \preceq l\{\{\}\} \Leftrightarrow \mathit{false}$ *if* $n \geq 1$

(3) Let Π be the set of (total) functions $\{t_1^1, \ldots, t_n^1\} \to \{t_1^2, \ldots, t_m^2\}$*:*
$l\{t_1^1, \ldots, t_n^1\} \preceq l\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \bigvee_{\pi \in \Pi} \bigwedge_{1 \leq i \leq n} t_i^1 \preceq \pi(t_i^1)$
if $n \geq 1$ *and* $m \geq 1$

(4) $l_1\{t_1^1, \ldots, t_n^1\} \preceq l_2\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \mathit{false}$ *if* $l_1 \neq l_2$ *and* $n \geq 0$ *and* $m \geq 0$

– $\rightsquigarrow$ **Elimination:**

$$X \rightsquigarrow t^1 \preceq t^2 \quad \Leftrightarrow \quad t^1 \preceq t^2 \ \wedge \ t^1 \preceq X \ \wedge \ X \preceq t^2$$

– **Descendant Elimination:**

$$\mathit{desc}\ t^1 \preceq l_2\{t_1^2, \ldots, t_m^2\} \Leftrightarrow t^1 \preceq l_2\{t_1^2, \ldots, t_m^2\} \vee \bigvee_{1 \leq i \leq m} \mathit{desc}\ t^1 \preceq t_i^2 \quad \text{if } m \geq 0$$

Applying the $\rightsquigarrow$ and descendant elimination rules to a constraint store C in disjunctive normal form may yield a constraint store not in disjunctive normal form. Thus, the method has to repeatedly restore the disjunctive normal form of C.

In the following, $mgcu(t_1, \ldots, t_n)$ (with $t_1, \ldots, t_n$ query terms) returns a most general commutative-unifier of $t_1, \ldots, t_n$ (in the sense of [6]) expressed as either *false*, if t_1 and t_2 are not commutative-unifiable, or as *true* if t_1 and t_2 are commutative-unifiable and do not contain variables, or else as a conjunction of equations of the form $X = t$. Note that most general commutative-unifiers are only computed for construct terms (i.e., query terms without $\rightsquigarrow$ and *desc* constructs). Recall that commutative unification is decidable.

In the definition below, simulation unification is initialized with $X_0 \rightsquigarrow t^q \preceq t^c$, where X_0 is a variable occurring neither in t^q nor in t^c, instead of simply $t^q \preceq t^c$. The additional variable X_0 serves a complete specification of the answers returned. This is useful in proving the correctness of simulation unification but can sometimes be dispensed of in practice.

Definition 10 (Simulation Unification).

1. **Initialization:**
   $C := X_0 \rightsquigarrow t^q \preceq t^c$
   (with t^q query term, t^c construct term and t^q, t^c and X_0 variable disjoint).
2. **Term Decomposition:**
   Until C can no longer be modified, repeat performing one of:
   - *Apply a (applicable) Term Decomposition rule to C*
   - *Put C in disjunctive normal form*

   end-until
3. **Variable Binding:**
   Replace each $X \preceq t$ in C with $X = t$.
4. **Consistency Verification:**
   For each disjunct D of C and for each variable X occurring in D do:

   Replace in D the equations $X = t_1, \ldots, X = t_n$ by mgcu$(t_1, \ldots, t_n)$.

   end-for

Note that the constraint store C returned at the end of the Term Decomposition step is necessarily in disjunctive normal form. Indeed, if C is not in disjunctive normal form, then the halting condition of the until loop (Step 2 of Definition 10) is not satisfied.

For efficiency reasons it is preferable to intertwine the Term Decomposition and Consistency Verification phases instead of performing them one after another. The sequential processing of both phases in Definition 10 simplifies the proofs.

Proposition 1 (Correctness and Completeness). *Let t^q be a query term, t^c a construct term, i.e., a query term without $\rightsquigarrow$ and desc constructs, and X_0 a variable such that t^q, t^c and X_0 are variable disjoint. There exists a substitution τ such that $t^q\tau$ and $t^c\tau$ are database terms and $t^q\tau = t^c\tau$ if and only if a simulation unification initialized with $X_0 \rightsquigarrow t^q \preceq t^c$ returns a substitution σ such that*

- *For each variable X in t^q, $X\sigma$ is a subterm of $t^q\sigma$.*
- *$t^q\tau$ is an instance of $t^q\sigma$.*
- *$t^c\tau$ is an instance of $t^c\sigma$.*

The proof is given in [118].

3.3 Complexity of Simulation Unification

The complexity of standard unification is famously linear. The complexity picture for simulation unification is considerably more "complex": It is easy to see that full simulation unification is NP-complete. It is in NP, e.g., by translation to first-order logic [60]. It is NP-hard, e.g., by reduction from subgraph isomorphism [90] or 3SAT. In presence of incomplete term specifications and variables this is not surprising and in line with similarly expressive XML query languages such as XPath 2 (see [16]).

In this section, we thus introduce a family of restrictions to the query terms defined in Definition 2 and briefly summarize the complexity of simulation unification for these languages.

We denote with $\mathrm{SU}^{\mathsf{tree};-\mathsf{bi},-\mathsf{di}}_{-\mathsf{var},-\mathsf{inj},-\mathsf{u}}$ the simulation unification problem over query terms without variables ($-\mathsf{var}$), term injectivity ($-\mathsf{inj}$), and unordered terms ($-\mathsf{u}$), where query terms may only be tree-shaped (i.e., no references) and may be neither incomplete in depth ($-\mathsf{di}$) nor breadth ($-\mathsf{bi}$). Note, that all these restrictions only apply to query terms, not to data terms. For $-\mathsf{var}$, we can also use $-\mathsf{tvar}$ to only disallow term variables, such that label variables can still be used. For $-\mathsf{u}$ we can also use $-\mathsf{o}$ to disallow ordered terms. We can drop any of these restrictions, e.g., $\mathrm{SU}^{-\mathsf{bi}}_{-\mathsf{var}}$ denotes the simulation unification problem over the sub-language of query terms without variables and with no breadth-incomplete terms.

	Fragment	Complexity
1	SU	NP-complete
2	$\mathrm{SU}_{-\mathsf{var}}$	NP-complete
3	$\mathrm{SU}^{\mathsf{tree}}_{-\mathsf{tvar}}$	NP-complete
4	$\mathrm{SU}^{\mathsf{tree}}_{-\mathsf{var},-\mathsf{inj}}$	$q \cdot d^2$, $q \cdot d$ if *data terms* are trees or CIGs [60]
5	$\mathrm{SU}^{\mathsf{tree}}_{-\mathsf{var}}$	$q \cdot d \cdot \frac{(q+d)^{1.5} \cdot q \cdot d}{log(q+d)}$
6	$\mathrm{SU}^{\mathsf{tree}}_{-\mathsf{var},-\mathsf{o}}$	$q \cdot d \cdot \frac{(q+d)^{1.5} \cdot q \cdot d}{log(q+d)}$
7	$\mathrm{SU}^{\mathsf{tree}}_{-\mathsf{var},-\mathsf{u}}$	$q \cdot d^2$
8	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di}}_{-\mathsf{var}}$	$q^{1.5} \cdot d$
9	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di}}_{-\mathsf{var},-\mathsf{o}}$	$q^{1.5} \cdot d$
10	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di}}_{-\mathsf{var},-\mathsf{u}}$	$q \cdot d$
11	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di},-\mathsf{bi}}_{-\mathsf{var}}$	$q + d$
12	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di},-\mathsf{bi}}_{-\mathsf{var},-\mathsf{o}}$	$q + d$
13	$\mathrm{SU}^{\mathsf{tree};-\mathsf{di},-\mathsf{bi}}_{-\mathsf{var},-\mathsf{u}}$	$q + d$

Table 5. Complexity of simulation unification. (q size of query term; d size of data term (number of nodes, see [60])

Table 5 summarizes the complexity results for simulation unification over the most interesting classes of restricted query terms.

The first three lines recall NP-complete fragments: (1) simulation unification over unrestricted query terms is NP-complete. (2) It remains so even if we allow no variables, but query terms may be graph-shaped. (3) It also remains NP-complete if we allow only tree shaped query terms, but any form of variables (including only label variables).

Proof (Sketch). It is easy to see that (1) (and thus (2) and (3)) is in NP, cf. [60] for a reduction to first-order logic.

NP-hardness for (1) and (2) can be shown by reduction from subgraph isomorphism: Let P, G be arbitrary graphs. Then we can test if P is isomorphic to a subgraph in G by the following construction (let $\lambda \neq mu$ be arbitrary labels): For each node of P, we construct a *breadth-incomplete* query term q_i with label λ containing a reference to the query term of each adjacent node. Let $q = \mu\{\{q_1, q_2, \ldots\}\}$. For each node of G, we construct a breadth-complete data term d_i with label λ and references to each query term of adjacent nodes. Let $d = \mu\{d_1, d_2, \ldots\}$. Then, q simulates in d, if and only if P is a subgraph of G. For instance, for the graph P_1 with edges $(1,2), (2,3), (1,4)$ and the graph G_1 with edges $(1,2), (2,3), (2,4), (1,4), (1,5), (4,5)$ the terms are $\mu\{\{\lambda\{\{\lambda\{\{\lambda\{\{\}\}\ \}\}, \lambda\{\{\}\}\ \}\}$ and $\mu\{\lambda\{\lambda\{\}, \uparrow 4\}, 4@\lambda\{\uparrow 5\}, 5@\lambda\{\}\}$.

NP-hardness for (3) is shown in [90] by reduction from Clique.

Fragment 4 of Table 5 highlights a major result of [60]: If we relax the injectivity requirement, i.e., that the $t_1^2, \ldots t_m^2$ must be pairwise distinct in Definition 5, simulation unification becomes polynomial for query terms without of references and variables. In fact, there is a large class of graph-shaped data terms, called CIGs in [60], that includes all trees and forests on which simulation unification has linear data complexity in this case.

Fragments 5-7 consider the effect of injectivity in case of tree shaped query terms without variables. Fragment 5 is the general case, fragment 6 if we allow no ordered query terms (no []), fragment 7 if we allow no unordered query terms (no {}). The complexity of fragments 5 and 6 has been an open issue as stated in [90] and is first shown here. It turns out that all three fragments have polynomial complexity.

Proof. The polynomial complexity for Fragment 7 follows from equivalence of this fragment to a fragment of navigational XPath, see [60], for whose complexity see [16]. For instance, $a[[\mathsf{desc}\ b, c[\mathsf{desc}\ e]\ ]]$ is equivalent to

```
/a[./*[descendant-or-self::b]/following-sibling::c[./*[1]
      [descendant-or-self::e][not(following-sibling::*)]]
```

The complexity for fragment 6 can be shown by reduction to maximum matching for bipartite graphs (or a non-linear assignment problem): We consider bottom-up each sub-term c in the given query q. For each c, M_c denotes the set of nodes in the data term that match with c. We start with the leaf terms and set M_c to the set of nodes with the appropriate label and arity ($|M_c| \leq d$). For inner query terms, if c is incomplete, let $c = \lambda\{\{s_1, \ldots, s_n\}\}$. If c is complete

let $c = \lambda\{s_1, \ldots, s_n\}$. For each node n in the data term d, that has the appropriate label and arity for c, for consider all s_i and let $M^n_{s_i}$ be the restriction of M_c to children of n. Thus the $M^n_{s_i}$ are the possible matches for each s_i under the assumption that c matches with n. From the $M^n_{s_i}$ we construct a bipartite graph in the following way: $G = (P, C; E)$ where P is all the s_i, $C = \cup_i M^n_{s_i}$, and $E = \{(s_i, c) : c \in M^n_{s_i}\}$, i.e., there is an edge from s_i to a node in C, if that node is a possible match for s_i.

For this bipartite graph, we compute the *maximum matching* S. A subset M of E is called a *matching* if every vertex of G coincides with at most one edge from M. A matching is called maximal if it cannot be enlarged by any edge of the graph. A matching is called maximum, if it has maximal cardinality among all matchings for that graph.

If S has cardinality n, n is a match for c and is added to M_c. If $M_q \neq$ after all query terms have been processed, then q simulates in d.

Computing the maximum matching can be done in $O(\frac{(q+d)^{1.5} \cdot q \cdot d}{log(q+d)})$, cf.[37]. Thus, the whole algorithm takes $q \cdot d \cdot \frac{(q+d)^{1.5} \cdot q \cdot d}{log(q+d)}$.

For fragment 5 finally, we can use the same algorithm as for 6, but if c is ordered we walk over the s_i and the children of n at the same time and compare matches in order, which can be done in $q + n$ time. Thus, it has the same complexity as fragment 6.

This result is indeed surprising, as unordered tree inclusion is NP-complete [84]. The difference between our case and unordered tree inclusion is that simulation unification does not consider a query term such as $a\{\{\mathsf{desc}\ b, \mathsf{desc}\ c\}\}$ to simulate with $a\{d\{b, c\}\ \}$ due to the injectivity requirement in Definition 5. For unordered tree inclusion the graph with edges $(a, b), (a, c)$ is included in the graph $(a, d), (d, b), (d, c)$, as unordered tree inclusion only preserves the ancestor relationships.

Fragments 8-10 are the cases, where we also restrict the use of desc. All three cases following immediately from the complexity of ordered, resp. unordered *path* inclusion problems shown in [83].

Finally, fragments 11-13 are the cases, where we consider only complete terms (without either desc, [[]] or {{}}). In this case, simulation unification is the same problem as tree isomorphism for node-labeled trees, for which well-known linear algorithms exist.

To summarize, there are two surprising results when considering simulation unification over restricted fragments of the full query terms defined in Definition 2:

- Even in presence of incomplete terms and for terms mixing unordered and ordered specifications, simulation unification has linear data complexity on tree and CIG data and quadratic data complexity on arbitrary graphs, if we *ignore the injectivity requirement.*
- In presence of the injectivity requirement, simulation unification still remains polynomial in contrast to closely related problems such as unordered tree inclusion.

Though most of the results above have been proven in [60] and [90], the latter case has been stated as an open issue even in [90]. In this paper, we close this remaining gap in the complexity picture for simulation unification with a positive result: Simulation unification remains polynomial as long as we do not use (multiple occurrences of) variables or graph-shaped query terms, even in presence of unordered terms. It is especially, interesting that thanks to the particular variant of injectivity chosen for query terms, simulation unification remains polynomial, as this requirement has been made for practical reasons: An intuition—so far often confirmed, admittedly on unsystematic observations— is that injectivity eases in practice the programming of queries.

4 RDFLog: Datalog with Value Invention

RDFLog extends Datalog to support two distinguishing features of RDF: blank nodes and the logical core [100] of the RDFS vocabulary. In RDFLog, Blank nodes can be specified by existentially quantified variables in rule heads. RDFLog allows *unrestricted quantifier alternation* between existential and universal quantifiers in a rule. The following examples illustrate the benefit of unrestricted quantifier alternation.
(1) "Someone knows each professor" can be represented in RDFLog as

$$\exists stu \forall prof\ ((prof, \mathsf{rdf{:}type}, \mathsf{uni{:}professor}) \rightarrow (stu, \mathsf{uni{:}knows}, prof)) \qquad (1)$$

We call such rules $\exists\forall$ rules. Some approaches such as [130] are limited to rules of this form.
(2) "Each lecture must be "practiced" by another course (such as a tutorial or practice lab) without knowing more about that course". This statement can not be expressed by $\exists\forall$ rules. In RDFLog it can be represented as

$$\forall lec \exists crs \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \rightarrow (crs, \mathsf{uni{:}practices}, lec)\big) \qquad (2)$$

Such rules are referred to as $\forall\exists$ rules. Recent proposals for rule extensions to SPARQL are limited to this form, if they consider blank nodes in rule heads at all. Indeed, with SPARQL's `CONSTRUCT` patterns a fresh blank node is constructed for each binding of the universal variables (cf. Section 10.2.1 in [108]).
(3) "For each lecture there is a course that "practices" that lecture and is attended by all students attending the lecture". This is represented in RDFLog as

$$\forall lec \exists crs \forall stu \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \wedge (stu, \mathsf{uni{:}attends}, lec) \rightarrow (crs, \mathsf{uni{:}practices}, lec) \wedge (stu, \mathsf{uni{:}attends}, crs)\big) \qquad (3)$$

To the authors' knowledge, RDFLog is the first RDF query language that supports rules of this third kind. RDFLog furthermore is a closed RDF query language, i.e., the answer to an RDFLog program is again an RDF graph, and RDFLog can express the logical core ρdf of the RDFS semantics [100].

In [86] it is suggested to extend Logic Programming—called Computational Logic—with existential quantifications in rule's heads, that is, with the very extension RDFLog provides. However, in this book, no method is described for the processing such rules.

The proofs of all results of this section on RDFLog and value invention can be found in [32].

4.1 Preliminaries

Definition 11 (RDF Graph [73]). *An* RDF vocabulary V *consists of two disjoint sets called* URIs U *and* literals L. *The* blank nodes B *is a set disjoint from* U *and* L. *An* RDF graph *is a set of RDF triples where an* RDF triple *is an element of* $(\mathsf{U} \cup \mathsf{B}) \times \mathsf{U} \times (\mathsf{U} \cup \mathsf{L} \cup \mathsf{B})$. *If* $t = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ *is an RDF triple then* s *is the* subject, p *is the* predicate, *and* o *is the* object *of* t.

The set L of literals consists of three subsets, *plain literals*, *typed literals* and *literals with language tags*. In this work we consider only plain literals (and thus drop IL, the interpretation function for typed literals, see Section 1.3 in [73], in the following definitions).

Definition 12 (RDF Interpretation [73]). *An* interpretation I *of an RDF vocabulary* $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ *is a tuple* $(\mathsf{IR}, \mathsf{LV}, \mathsf{IP}, \mathsf{IEXT}, \mathsf{IS})$ *where* IR *is a non-empty set of* resources *such that* $\mathsf{L} \subseteq \mathsf{LV} \subseteq \mathsf{IR}$, IP *is a set of* properties *and* $\mathsf{IEXT} : \mathsf{IP} \to 2^{\mathsf{IR} \times \mathsf{IR}}$, *and* $\mathsf{IS} : \mathsf{U} \to \mathsf{IR} \cup \mathsf{IP}$ *are mappings.*

IR and IP are not necessarily disjoint since a same URI can be used both as a resource and as a property. RDF interpretations are used to assign a truth value to an RDF graph. RDF assigns a special meaning to a predefined vocabulary, called RDFS vocabulary. It is, e.g., required that $\mathsf{IEXT}(\mathsf{IP}(\mathsf{rdfs} : \mathsf{subPropertyOf}))$ is transitive and reflexive. The formulation of theses constraints on RDF interpretation makes use of a notion of a *class* which is omitted in the definition above for simplicity. The logical core of RDFS, denoted as *ρdf*, has been identified in [100]. An RDF interpretation I is a *ρdf interpretation* if I satisfied the constraints specified in Definition 3 of [100].

The semantics of RDF is completed by the notion of entailment: An RDF graph g *RDF-entails* (*ρdf-entails*) an RDF graph h if for all RDF (*ρdf*) interpretations I, $\mathsf{I}(\mathsf{h}) = \text{true}$ if $\mathsf{I}(\mathsf{g}) = \text{true}$ [73].

The following uses formulas, terms, structures, Herbrand structures, satisfaction $\models$, models, entailment $\models$, logic and Datalog programs, and the *immediate consequence operator* T_P of a logic program P. Infinite formulas [10] are also used: if Φ is a recursively enumerable set of formulas, then $\bigwedge(\Phi)$ is a formula; if $\bar{x} = x_1, x_2, \ldots$ is a recursively enumerable sequence of variables and if φ is a formula, then $\exists \bar{x}(\varphi)$ is a formula. We write $\varphi(\bar{x})$ to indicate that the free variables of a formula φ are amongst $\bar{x} = x_1, \ldots, x_n$.

[10] Of a limited form: infinite conjunctions all variables of which are existentially quantified.

We show that the semantics of RDF can be defined in terms of standard logic. In particular, RDF graphs can be translated to formulas so that logical entailment coincides with RDF entailment. For any RDF vocabulary $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ we define the alphabet $\Sigma_\mathsf{V} = \mathsf{U} \cup \mathsf{L} \cup \{T\}$ where U and L are constant symbols and T is an arbitrary ternary relation symbol.

Definition 13 (Canonical Formula of an RDF Graph). *Let* $\mathsf{g} = \{\mathsf{t}_1, \ldots, \mathsf{t}_n\}$ *be an RDF graph over* V*. The* canonical formula *of* g *is the formula* $\varphi_\mathsf{g} := \exists \bar{x} \, (\psi_1(\bar{x}) \wedge \ldots \wedge \psi_n(\bar{x}))$ *over* Σ_V *and variables from* B *where* $\psi_i = T(\mathsf{s}, \mathsf{p}, \mathsf{o})$ *if* $\mathsf{t}_i = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ *and* $\bar{x}$ *is the set of blank nodes occurring in* g*.*

It is easy to see that *ρdf* [100] corresponds to a finite set of Datalog rules $\Phi^{\rho df}$.

Proposition 2. *Let* g, h *be RDF graphs and* φ_g, φ_h *their canonical formulas. Then* g *RDF-entails* h *iff* $\varphi_\mathsf{g} \models \varphi_\mathsf{h}$ *and* g *ρdf-entails* h *iff* $\varphi_\mathsf{g} \wedge \Phi^{\rho df} \models \varphi_\mathsf{h}$*.*

4.2 RDFLog Syntax

Definition 14 (Syntax of RDFLog Programs). *Let* $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ *be an RDF vocabulary and Var a set of variables. An* RDFLog atom *over* V *is an atom* $T(t_1, t_2, t_3)$ *where* $t_1, t_2 \in (\mathsf{U} \cup Var)$ *and* $t_3 \in (\mathsf{U} \cup \mathsf{L} \cup Var)$*. An* RDFLog rule *over* V *is a formula*

$$\forall \bar{x}_1 \exists \bar{y}_1 \ldots \forall \bar{x}_n \exists \bar{y}_n \, (body(\bar{x}) \rightarrow head(\bar{x}, \bar{y}))$$

over Σ_V *and Var where* $\bar{x} = \bar{x}_1, \ldots, \bar{x}_n$ *and* $\bar{y} = \bar{y}_1, \ldots, \bar{y}_n$ *are finite sequences from Var and* $body(\bar{x})$ *and* $head(\bar{x}, \bar{y})$ *are finite conjunctions of RDFLog atoms. In addition we require that RDFLog rules are* range restricted*: if* $x \in Var(head)$ *is universal or there is an existential* $y \in Var(head)$ *such that* y *is in the scope of* x*, then* $x \in Var(body)$*. An* RDFLog program *over* V *is a finite set of RDFLog rules over* V*.*

Any finite RDF graph $\mathsf{g} = \{t_1, \ldots, t_n\}$ with blank nodes $\bar{x}$ can be encoded into the RDFLog rule $\exists \bar{x} \, (true \rightarrow t_1 \wedge \ldots \wedge t_n)$ where *true* denotes the empty conjunction. As it makes the notation simpler, we always assume that the input RDF graph is encoded into such a rule in the RDFLog program. As there is only one predicate symbol (T) in an RDFLog program, it can be omitted.

4.3 Declarative Semantics

The following RDFLog program shows that it is problematic to define the semantics of an RDF query language in terms of models. Let the *canonical structure* A_g of an RDF graph g be the structure over the domain of URIs, literals and blank nodes where (t_1, t_2, t_3) is true in A_g iff (t_1, t_2, t_3) is an RDF triple in g. As (2) is a fact in P and (1) is a rule in P, any canonical structure of an RDF graph that is a model of P must contain the triple ('Logic', uni:located_in, _:b) for some blank node _:b. Since this triple contains a literal in the subject position, it is not an

RDF triple. Thus, that P has no model which is the canonical structures of an RDF graph. Even if, as with SPARQL, literals in subject position are allowed, one can similarly argue with blank nodes in predicate position.

$$
\begin{aligned}
P = \big\{ & \forall sem \exists rm \forall stu \big((stu, \mathsf{uni{:}attends}, sem) \\
& \quad \rightarrow (sem, \mathsf{uni{:}located\_in}, rm) \wedge (stu, \mathsf{uni{:}knows}, rm) \big), && (1) \\
& \quad true \rightarrow (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \text{'Logic'}) \wedge (\mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF}) \big\} && (2)
\end{aligned}
$$

$$
\begin{aligned}
[\![P]\!] \ni \big\{ & (\mathsf{\_{:}b3}, \mathsf{uni{:}located\_in}, \mathsf{\_{:}b1}), (\mathsf{uni{:}julie}, \mathsf{uni{:}knows}, \mathsf{\_{:}b1}), \\
& (\mathsf{uni{:}RDF}, \mathsf{uni{:}located\_in}, \mathsf{\_{:}b2}), (\mathsf{uni{:}john}, \mathsf{uni{:}knows}, \mathsf{\_{:}b2}), \\
& (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \text{'Logic'}), (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b3}), \\
& (\mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF}) \big\}
\end{aligned}
$$

The difficulty is overcome by defining the semantics of RDFLog in terms of RDF entailment. More precisely, the semantics of an RDFLog program P is defined as the set of all RDF graphs g that entail exactly the same RDF graphs as P (and satisfying in particular $P \models g$).

Definition 15 (Denotational Semantics of RDFLog). *Let P be an RDFLog program and* RDF *the set of RDF graphs. The* denotational semantics $[\![P]\!]$ *of P is the set* $[\![P]\!] := \{\mathsf{g} \in \mathsf{RDF} \mid \forall \mathsf{h} \in \mathsf{RDF}\,(P \models \varphi_\mathsf{h} \textit{ iff } \varphi_\mathsf{g} \models \varphi_\mathsf{h})\}$ *where φ_g and φ_h are the canonical formulas of* g *and* h *respectively.*

Observe that the semantics of an RDFLog program is an infinite set of possibly infinite RDF graphs. As we formalized RDF graphs as formulas, we have to consider the special kind of infinite formulas defined above. Nonetheless it is immediate from the definition that the RDF graphs in $[\![P]\!]$ form an equivalence class under RDF entailment. Therefore any element of $[\![P]\!]$ characterizes the infinite set $[\![P]\!]$. In the next section we show how such a representative can be computed.

Observe that $\Phi^{\rho df}$ encoded in RDFLog. Therefore it is up to the programmer to enclose $\Phi^{\rho df}$ into P if the semantics of P is supposed to be aware of the *ρdf* vocabulary.

subsectionOperational Semantics

RDFLog operational semantics consists in (1) Skolemization, (2) standard Datalog evaluation, (3) un-Skolemization, and (4) normalization. Normalisation discards intermediary triples that may contain blank nodes in predicate position (see [126] for cases where this is useful), the final answer of an RDFLog program never contains such triples.

Definition 16 (Skolemisation). *Let Σ and Γ be disjoint alphabets, $\varphi = \forall \bar{x} \exists y(\psi)$ a formula over $\Sigma \cup \Gamma$ and $f \in \Gamma$. A* Γ-Skolemisation step *s_f maps φ to $s_f(\varphi) := \forall \bar{x} \psi\{y \leftarrow f(\bar{x})\}$. A* Γ-Skolemisation *s is a composition $s_{f_1} \circ \ldots \circ s_{f_n}$ of Γ-Skolemisation steps such that f_i does not occur in $s_{f_{i+1}} \circ \ldots \circ s_{f_n}(\varphi)$ and $s(\varphi)$ contains no existential variables. The definition of a Skolemisation is extended to sets in the usual way.*

The Skolemised of an RDFLog program P is equivalent to a range restricted logic program $s(P)$. Any logic programming engine can compute the minimal Herbrand model $M_{s(P)}$ of $s(P)$.

We define $\varphi_{M_{S(P)}}$ to be the conjunction of all ground atoms that are true in $M_{s(P)}$. However, $\varphi_{M_{S(P)}}$ might not be the canonical formula of an element of $[\![P]\!]$ for two reasons. First, the example shows that $\varphi_{M_{S(P)}}$ might contain atoms with skolem terms, such as $(\mathsf{uni{:}RDF}, \mathsf{uni{:}located\_in}, s_{rm}(\mathsf{uni{:}RDF}))$, which are not entailed by P. Second, $\varphi_{M_{S(P)}}$ can contain atoms that contain literals in subject or predicate position and blank nodes in predicate position. In the example the atom $(\text{`Logic'}, \mathsf{uni{:}located\_in}, s_{rm}(\text{`Logic'}))$ contains the literal 'Logic' in subject position.

The first problem is solved by "undoing" the Skolemisaton, i.e., replacing each Skolem term in $\varphi_{M_{S(P)}}$ by a fresh, distinct blank node. We call this operation *UnSkolemisaton*.

Definition 17 (Unskolemisation). *Let Σ and Γ be disjoint alphabets and φ a ground, possibly infinite, and quantifier free formula over $\Sigma \cup \Gamma$. Let $\bar{t}$ be the sequence of all ground terms $f(\bar{u})$ where f is in Γ and $\bar{u}$ is a sequence of terms over $\Sigma \cup \Gamma$. Then the* Γ-Unskolemisation *u maps φ to $u(\varphi) := \exists \bar{x} \left(\varphi\{\bar{t} \leftarrow \bar{x}\}\right)$. where $\bar{x}$ is a sequence of fresh variables.*

To solve the second problem, we remove all triples with literals or blank nodes in predicate position (no RDF graph may contain such a triple or any triple entailed by it). In addition we remove each triple t that contains a literal l in object position and add two triples t_1 and t_2 where t_1 is obtained from t by replacing an occurrence of a literal l in subject position by a fresh blank node b_l and t_2 is obtained from t by replacing all occurrences of l by b_l.

This is necessary to preserve information about the identity of domain elements that are denoted by blank nodes. For example observe that the RDF graph $\{(\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b}), (\mathsf{\_{:}b}, \mathsf{uni{:}located\_in}, s_{rm}(\text{`Logic'}))\}$ follows from the RDFLog program P above. To maintain this information we need to insert the triple $(\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b3})$ into $[\![P]\!]$. We formalise this step by defining the normalisation operator.

Definition 18 (Normalisation Operator). *Let φ be a formula of the form $\exists \bar{x} \left(a_1(\bar{x}) \wedge \ldots \wedge a_n(\bar{x})\right)$ where each $a_i(\bar{x}) = T(t_1, t_2, t_3)$ for some $t_1, t_2, t_3 \in (\mathsf{U} \cup \mathsf{B} \cup \mathsf{L})$. Let $\mathsf{L}' \subseteq \mathsf{L}$ be the set of literals that occur in the first argument of an atom in φ. We define $\mu : \mathsf{U} \cup \mathsf{B} \cup \mathsf{L} \to \mathsf{U} \cup \mathsf{B} \cup \mathsf{L}$ to be the injection such that $\mu(t) = b$ for some fresh blank node b (not in φ) if $t \in \mathsf{L}'$ and $\mu(t) = t$ otherwise. Then $\Pi(\varphi) = \{\Pi(a_1(\bar{x})), \ldots \Pi(a_n(\bar{x}))\}$ and*

$$\Pi(T(t_1, t_2, t_3)) = \begin{cases} \top & \text{if } t_2 \in \mathsf{B} \cup \mathsf{L} \\ (\mu(t_1), t_2, t_3) \wedge (\mu(t_1), t_2, \mu(t_3)) & \text{otherwise} \end{cases}$$

The normalisation operator ensures that, though intermediary triples may contain blank nodes in predicate position (see [126] for examples where this is useful), the final answer of an RDFLog program never contains such triples.

Definition 19 (Operational Semantics of RDFLog). *Let P be an RDFLog program over Σ, s a Γ-Skolemisation for P, and u an Γ-Unskolemisation. Then the* operational semantics *of P is $[P] := \Pi\left(u(\varphi_{M_{s(P)}})\right)$ where $\varphi_{M_{S(P)}}$ is as defined above: the conjunction of all ground atoms that are true in the minimal Herbrand model of $s(P)$.*

4.4 Properties and Experimental Evaluation

Even though we do not require that elements of the denotational semantics $[\![P]\!]$ of an RDFLog program P are models of P it holds that $u(\varphi_{M_{s(P)}})$ has a canonical structure that is not only a model of P but even a universal model [56,57]. Thus if we allow literals in subject position and blank nodes in subject or predicate position, we can omit Π from the operational semantics and compute a model of P.

To formulate this more precisely, we define an *extended Herbrand structure* A over alphabet Σ and variables *Var* as a structure (D, Rel, Fun) where D is the set of (possibly non-ground) terms over Σ and *Var*, and every function f^A is defined by $f^A(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. We extend the definition of Unskolemisation from formulas to extended Herbrand structures: if u is an Unskolemisation that replaces $\bar{t}$ by $\bar{x}$ then $u(M)$ is the extended Herbrand structure obtained from M by renaming the domain elements $\bar{t}$ by $\bar{x}$.

Lemma 1. *Let P be an RDFLog program, $A_P = u(M_{s(P)})$ and $\varphi_P = u\left(\varphi_{M_{s(P)}}\right)$. Then $A_P \models P$ and $P \models \varphi_P$.*

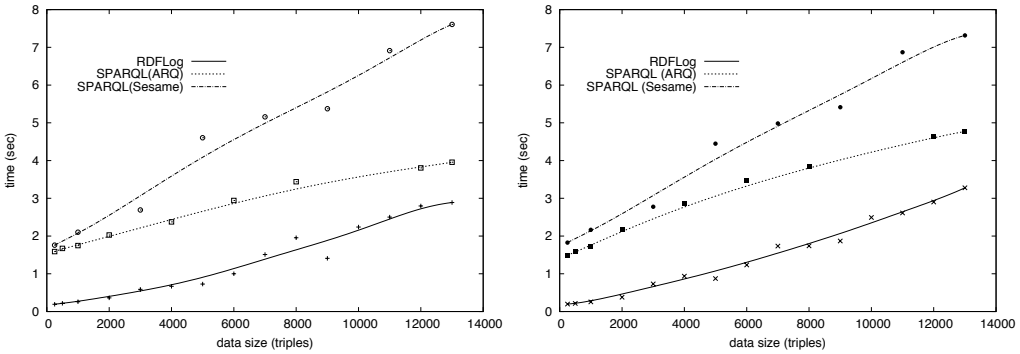
Intuitively, $A_P \models P$ means that φ_P captures all the information in P and $P \models \varphi_P$ means that it does not assert anything that is not asserted by P. From these two key observations, we can prove that the operational semantics of RDFLog is both sound and complete with respect to the denotational semantics.

Theorem 1. *Let P be an RDFLog program. Then $[P] \in [\![P]\!]$.*

The reduction of RDFLog to standard logic programs allows for a direct implementation of RDFLog on top of any logic programming or database engine that supports value invention and recursion. In the following, we we compare experimentally the performance of a very simple prototype based on that principle with two of the more common SPARQL implementations. Our implementation of RDFLog uses a combination of Perl pre- and post-filters for Skolemisation, Unskolemisation, and normalisation of RDFLog programs and XSB Prolog to evaluate the Skolemised programs.

We compare our implementation with the ARQ SPARQL processor of Jena (Version 2.1) and the SPARQL engine provided by the Sesame RDF Framework. For Sesame, we choose the main-memory store as it is "by far the fastest type of repository that can be used" according to Sesame's authors. With this store, Sesame becomes a main-memory, ad-hoc query engine just like RDFLog and ARQ. As common for ad-hoc queries we measure overall execution time including both loading of the RDF data and execution of the SPARQL or RDFLog query.

Fig. 2 Performance comparison on rule 1 (left) and on rule 2 (right)



In the experiments we evaluate three different queries against an RDF graph consisting of Wikipedia data. The experiments have been carried out on a Intel Pentium M Dual-Core with 1.86 GHz, 1 MB cache and 2 GB main memory. For each setting, the running time is averaged over 25 runs. We compare the following rules:

- *Rule 1:* $\forall x \forall y\,((x, \mathsf{wiki{:}internalLink}, y) \rightarrow (x, \mathsf{test{:}connected}, y))$
- *Rule 2:* $\forall x \forall y \exists z\,((x, \mathsf{wiki{:}internalLink}, y) \rightarrow (x, \mathsf{test{:}connected}, z))$

Figure 2 compares the performance of RDFLog with that of ARQ and Sesame for rule 1 and rule 2 (we omit rule 3 as it is not expressible in SPARQL). Despite its light-weight, ad-hoc implementation, RDFLog outperforms ARQ and Sesame in this setting. The figures show moreover that also for ARQ and Sesame, blank node construction does not bear any significant additional computational effort.

5 Conclusion

Datalog has proven a useful vehicle for research and advanced database systems. However, to remain such it must adapt to the ever more dominant Web. To that end, we describe two approaches for addressing two of the most glaring deficiencies of Datalog: simulation unification, for an easy access to semi-structured Web data, and RDFLog, for arbitrary quantifier alternation in rule heads needed for constructing RDF graphs. Both approaches pose new challenges to Datalog evaluation and analysis, but we show that in both cases polynomial core languages—at the cost of mild restrictions—can be identified.

Acknowledgements

The research leading to these results has received funding from the European Commission and the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE no. 506779 and the European

Research Council under the European Communitys Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 246858—DIADEM.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.
3. S. Abiteboul and V. Vianu. Regular Path Queries with Constraints. In *PODS*, pages 122–133, 1997.
4. D. E. Appelt. Introduction to Information Extraction. *AI Commun.*, 12(3):161–172, 1999.
5. G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. In *ICDE*, pages 24–33. IEEE Computer Society, 1998.
6. F. Baader. *Unification*, chapter Unification in Commutative Theories, pages 417–435. Academic Press, 1989.
7. J. Bailey, F. Bry, T. Furche, and S. Schaffert. Web and Semantic Web Query Languages: A Survey. In *Reasoning Web, First International Summer School 2005*, volume 3564 of *LNCS*. Springer-Verlag, 2005.
8. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *PODS*, pages 1–15. ACM, 1986.
9. P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors. *Reasoning Web, Second International Summer School 2006, Tutorial Lectures*, volume 4126 of *Lecture Notes in Computer Science*. Springer, 2006.
10. C. Baru, B. Ludäscher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. Features and Requirements for an XML View Denition Language: Lessons from XML Information Mediation. In *QL98*. W3C, 1998.
11. N. Bassiliades and I. P. Vlahavas. R-DEVICE: A Deductive RDF Rule Language. In G. Antoniou and H. Boley, editors, *RuleML*, volume 3323 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2004.
12. R. Baumgartner, S. Flesca, and G. Gottlob. The Elog Web Extraction Language. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR*, volume 2250 of *Lecture Notes in Computer Science*, pages 548–560. Springer, 2001.
13. R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *VLDB*, pages 119–128. Morgan Kaufmann, 2001.
14. C. Beeri and R. Ramakrishnan. On the Power of Magic. In *PODS*, pages 269–284. ACM, 1987.
15. K. V. Belleghem, M. Denecker, and D. D. Schreye. A Strong Correspondence between Description Logics and Open Logic Programming. In *ICLP*, pages 346–360, 1997.
16. M. Benedikt and C. Koch. Xpath leashed. 2007.
17. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernàndez, M. Kay, J. Robie, and J. Siméon, editors. *XML Path Language (XPath) Version 2.0*. Recommendation. W3C, 2007.
18. A. Berlea and H. Seidl. fxt – A Transformation Language for XML Documents. *J. of Computing and Information Technology (CIT), Special Issue on Domain-Specific Languages*, 2001.

19. S. Boag, D. Chamberlin, M. F. Fernaàndez, J. Robie, and J. Siméon, editors. *XQuery 1.0: An XML Query Language*. Recommendation. W3C, 2007.
20. H. Boley. Relationships Between Logic Programming and XML. In *Proc. 14th Workshop Logische Programmierung*, Würzburg, Jan 2000.
21. H. Boley. The RuleML Family of Web Rule Languages. In J. J. Alferes, J. Bailey, W. May, and U. Schwertel, editors, *PPSWR*, volume 4187 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2006.
22. H. Boley. Are Your Rules Online? Four Web Rule Essentials. In A. Paschke and Y. Biletskiy, editors, *RuleML*, volume 4824 of *Lecture Notes in Computer Science*, pages 7–24. Springer, 2007.
23. H. Boley, G. Halmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds, editors. *RIF Core Dialect*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
24. H. Boley and M. Kifer, editors. *RIF Basic Logic Dialect*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
25. H. Boley and M. Kifer, editors. *RIF Framework for Logic Dialects*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
26. H. Boley, M. Kifer, P.-L. Patranjan, and A. Polleres. Rule Interchange on the Web. In G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P.-L. Patranjan, and R. Tolksdorf, editors, *Reasoning Web*, volume 4636 of *Lecture Notes in Computer Science*, pages 269–309. Springer, 2007.
27. H. Boley, J. Mei, M. Sintek, and G. Wagner. RDF/RuleML Interoperability. In *Rule Languages for Interoperability*, 2005.
28. H. Boley, S. Tabet, and G. Wagner. Design Rationale for RuleML: A Markup Language for Semantic Web Rules. In I. F. Cruz, S. Decker, J. Euzenat, and D. L. McGuinness, editors, *SWWS*, pages 381–401, 2001.
29. O. Bolzer, F. Bry, T. Furche, S. Kraus, and S. Schaffert. Development of Use Cases, Part I. Technical Report PMS-FB-2005-23, Institute for Informatics, University of Munich, 2005.
30. A. Brügemann-Klein and D. Wood. Regular Tree Languages over Non-ranked Alphabets. Unpublished manuscript, 1998.
31. F. Bry. Query Evaluation in Deductive Databases: Bottom-Up and Top-Down Reconciled. *Data Knowledge Engineering*, 5:289–312, 1990.
32. F. Bry, T. Furche, C. Ley, B. Linse, and B. Marnette. RDFLog: It's like Datalog for RDF. Technical Report PMS-FB-2008-1, Institute for Informatics, University of Munich, 2005.
33. F. Bry, T. Furche, C. Ley, B. Linse, and B. Marnette. RDFLog: It's like Datalog for RDF. In *Workshop on (Constraint) Logic Programming (WLP 2008)*, 2008.
34. F. Bry, T. Furche, and B. Linse. The perfect match: Rpl and rdf rule languages. In A. Polleres and T. Swift, editors, *RR*, volume 5837 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2009.
35. P. Buneman. Tutorial Semistructured Data. In *PODS*, pages 117–121, 1997.
36. P. Buneman, M. F. Fernandez, and D. Suciu. Unql: A query language and algebra for semistructured data based on structural recursion. *VLDB J.*, 9(1):76–110, 2000.
37. R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, Society for Industrial and Applied Mathematics, 2009.
38. D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. View-Based Query Answering over Description Logic Ontologies. In G. Brewka and J. Lang, editors, *KR*, pages 242–251. AAAI Press, 2008.

39. J. J. Carroll and J. D. Roo, editors. *OWL Web Ontology Language Test Cases*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
40. S. Ceri, G. Gottlob, and L. Lavazza. Translation and Optimization of Logic Queries: The Algebraic Approach. In W. W. Chu, G. Gardarin, S. Ohsuga, and Y. Kambayashi, editors, *VLDB*, pages 395–402. Morgan Kaufmann, 1986.
41. S. Ceri, G. Gottlob, and L. Tanca. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989.
42. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.
43. D. Chamberlin, P. Fankhauser, M. Marchiori, and J. Robie, editors. *XML Query Use Cases*. W3C Working Group Note. World Wide Web Consortium (W3C), 2007.
44. D. Chimenti, R. Gamboa, R. Krishnamurthy, S. A. Naqvi, S. Tsur, and C. Zaniolo. The ldl system prototype. *IEEE Trans. Knowl. Data Eng.*, 2(1):76–90, 1990.
45. J. Clark, editor. *XSL Transformations (XSLT) Version 1.0*. Recommendation. W3C, 1999.
46. J. Clark and S. DeRose, editors. *XML Path Language (XPath) Version 1.0*. Recommendation. W3C, 1999.
47. J. de Bruijn, editor. *RIF RDF and OWL Compatibility*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
48. J. de Bruijn, T. Eiter, A. Polleres, and H. Tompits. On Representational Issues About Combinations of Classical Theories with Nonmonotonic Rules. In J. Lang, F. Lin, and J. Wang, editors, *KSEM*, volume 4092 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2006.
49. C. de Sainte Marie, G. Halmark, and A. Paschke, editors. *RIF Production Rule Dialect*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
50. M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
51. A. Deutsch, editor. *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, 2004.
52. W. Drabent, T. Eiter, G. Ianni, T. Krennwallner, T. Lukasiewicz, and J. Maluszynski. Hybrid Reasoning with Rules and Ontologies. In F. Bry and J. Maluszynski, editors, *REWERSE*, volume 5500 of *Lecture Notes in Computer Science*, pages 1–49. Springer, 2009.
53. T. Eiter, G. Ianni, T. Krennwallner, and A. Polleres. Rules and Ontologies for the Semantic Web. In C. Baroglio, P. A. Bonatti, J. Maluszynski, M. Marchiori, A. Polleres, and S. Schaffert, editors, *Reasoning Web*, volume 5224 of *Lecture Notes in Computer Science*, pages 1–53. Springer, 2008.
54. T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, and H. Tompits. Reasoning with Rules and Ontologies. In Barahona et al. [9], pages 93–127.
55. T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, and K. Wang. Forgetting in Managing Rules and Ontologies. In *Web Intelligence*, pages 411–419. IEEE Computer Society, 2006.
56. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *ICDT*, volume 2572 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2003.
57. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

58. D. Fensel, K. P. Sycara, and J. Mylopoulos, editors. *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.
59. T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.
60. T. Furche. *Implementation of Web Query Language Reconsidered: Beyond Tree and Single-Language Algebras at (Almost) No Cost*. Dissertation/doctoral thesis, Ludwig-Maxmilians University Munich, 2008.
61. T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. Rdf querying: Language constructs and evaluation methods compared. In Barahona et al. [9], pages 1–52.
62. H. Gallaire and J. Minker, editors. *Logic and Data Bases, Symposium on Logic and Data Bases*, Advances in Data Base Theory. Plenum Press, 1978.
63. H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and Databases: A Deductive Approach. *ACM Comput. Surv.*, 16(2):153–185, 1984.
64. H. Gallaire, J.-M. Nicolas, and J. Minker, editors. *Advances in Data Base Theory, Vol. 1, Based on the Proceedings of the Workshop on Formal Bases for Data Bases, December 12-14, 1979, Centre d'Études et de Recherches de l'École Nationale Supérieure de l'Aéronautique et de l'Espace de Toulouse (CERT), France*, Advances in Data Base Theory. Plenum Press, 1981.
65. H. Gallaire, J.-M. Nicolas, and J. Minker, editors. *Advances in Data Base Theory, Vol. 2, Based on the Proceedings of the Workshop on Logical Data Bases, December 14-17, 1982, Centre d'études et de recherches de Toulouse, France*, Advances in Data Base Theory. Plenum Press, 1984.
66. G. Gottlob and C. Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. In L. Popa, editor, *PODS*, pages 17–28. ACM, 2002.
67. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web information extraction. *J. ACM*, 51(1):74–113, 2004.
68. G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto Data Extraction Project - Back and Forth between Theory and Practice. In Deutsch [51], pages 1–12.
69. G. Grahne and L. V. S. Lakshmanan. On the Difference between Navigating Semi-structured Data and Querying It. In *Workshop on Database Programming Languages*, pages 271–296, 1999.
70. R. Grishman. Information Extraction. In *The Oxford Handbook of Computational Linguistics*, pages 545–559. Oxford University Press, 2003.
71. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.
72. C. Gutiérrez, C. A. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In Deutsch [51], pages 95–106.
73. P. Hayes, editor. *RDF Semantics*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
74. J. Heflin, editor. *OWL Web Ontology Language Use Cases and Requirements*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
75. M. Hori, J. Euzenat, and P. F. Patel-Schneider, editors. *OWL Web Ontology Language XML Presentation Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.

76. I. Horrocks. OWL Rules, OK? In *Rule Languages for Interoperability*, 2005.
77. I. Horrocks, J. Angele, S. Decker, M. Kifer, B. N. Grosof, and G. Wagner. Where Are the Rules? *IEEE Intelligent Systems*, 18(5):76–83, 2003.
78. I. Horrocks and P. F. Patel-Schneider. A proposal for an Owl rules language. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW*, pages 723–731. ACM, 2004.
79. I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL rules: A proposal and prototype implementation. *J. Web Sem.*, 3(1):23–40, 2005.
80. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ-Description Logic to Disjunctive Datalog Programs. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *KR*, pages 152–162. AAAI Press, 2004.
81. U. Hustadt, B. Motik, and U. Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.
82. G. Ianni, T. Krennwallner, A. Martello, and A. Polleres. A Rule System for Querying Persistent RDFS Data. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. P. B. Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 857–862. Springer, 2009.
83. P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, Faculty of Science, Department of Computer Science, 1992.
84. P. Kilpelainen and H. Mannila. Ordered and Unordered Tree Inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995.
85. G. Klyne. Representring Facts and Rules in RDF – Bridging Cconventional predicate representation and RDF. http://www.ninebynine.org/RDFNotes/RDFFactsAndRules.html, 2001.
86. R. Kowalski. Computational logic and human life: How to be artificially intelligent. Preprint, Department of Computing, Imperial College London, 2010. `http://www.doc.ic.ac.uk/~rak/papers/newbook.pdf`, to be published by Cambridge University Press.
87. R. A. Kowalski. The Early Years of Logic Programming. *Commun. ACM*, 31(1):38–43, 1988.
88. A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond – The Science of Search Engine Ranking*. Princetoon University Press, 2006.
89. C. Ley and M. Benedikt. How big must complete xml query languages be? In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 183–200, New York, NY, USA, 2009. ACM.
90. B. Linse. *Data Integration on the (Semantic) Web with Rules and Rich Unification*. PhD thesis, Ludwig-Maximilians-Universität München, 2010.
91. D. Maier. Communication during the Workshop Datalog 2.0, 2010.
92. M. Marx. Conditional xpath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.
93. D. L. McGuinness and F. van Harmelen, editors. *OWL Web Ontology Language Overview*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
94. G. Meditskos and N. Bassiliades. A Rule-Based Object-Oriented OWL Reasoner. *IEEE Trans. Knowl. Data Eng.*, 20(3):397–410, 2008.
95. G. Meditskos and N. Bassiliades. Combining a DL Reasoner and a Rule Engine for Improving Entailment-Based OWL Reasoning. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *ISWC*, volume 5318 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2008.

96. Z. Miklós, G. Neumann, U. Zdun, and M. Sintek. Querying semantic web resources using triple views. In Fensel et al. [58], pages 517–532.
97. B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, editors. *OWL 2 Web Ontology Language Profiles*. W3C Recommendation. World Wide Web Consortium (W3C), 2009.
98. B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006.
99. B. Motik and R. Volz. Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases. In F. Bry, C. Lutz, U. Sattler, and M. Schoop, editors, *KRDB*, volume 79 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
100. S. Muñoz, J. Pérez, and C. Gutiérrez. Minimal Deductive Systems for RDF. In E. Franconi, M. Kifer, and W. May, editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2007.
101. S. A. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, 1989.
102. F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.
103. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking Forward. In A. B. Chaudhri, R. Unland, C. Djeraba, and W. Lindner, editors, *EDBT Workshops*, volume 2490 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2002.
104. P. F. Patel-Schneider, P. Hayes, and I. Horrocks, editors. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
105. A. Polleres. From SPARQL to rules (and back). In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 787–796. ACM, 2007.
106. A. Polleres, H. Boley, and M. Kifer, editors. *RIF Datatypes and Built-Ins 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), 2010.
107. A. Polleres and R. Schindlauer. DLVHEX-SPARQL: A SPARQL Compliant Query Engine Based on DLVHEX. In A. Polleres, D. Pearce, S. Heymans, and E. Ruckhaus, editors, *ALPSWS*, volume 287 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
108. E. Prud'hommeaux and A. Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium (W3C), 2008.
109. J. Pührer, S. Heymans, and T. Eiter. Dealing with Inconsistency When Combining Ontologies and Rules Using DL-Programs. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2010.
110. R. Ramakrishnan. Magic Templates: A Spellbinding Approach to Logic Programs. In *ICLP/SLP*, pages 140–159, 1988.
111. R. Ramakrishnan and J. D. Ullman. A survey of deductive database systems. *J. Log. Program.*, 23(2):125–149, 1995.
112. J. Rohmer, R. Lescoeur, and J.-M. Kerisit. The Alexander Method - A Technique for The Processing of Recursive Axioms in Deductive Databases. *New Generation Comput.*, 4(3):273–285, 1986.

113. R. Rosati. On the decidability and complexity of integrating ontologies and rules. *J. Web Sem.*, 3(1):61–73, 2005.
114. R. Rosati. Semantic and Computational Advantages of the Safe Integration of Ontologies and Rules. In F. Fages and S. Soliman, editors, *PPSWR*, volume 3703 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2005.
115. R. Rosati. Integrating Ontologies and Rules: Semantic and Computational Issues. In Barahona et al. [9], pages 128–151.
116. R. Rosati. On Combining Description Logic Ontologies and Nonrecursive Datalog Rules. In D. Calvanese and G. Lausen, editors, *RR*, volume 5341 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2008.
117. D. Saccà and C. Zaniolo. Implementation of recursive queries for a data language based on pure horn logic. In *ICLP*, pages 104–135, 1987.
118. S. Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. Dissertation/Ph.D. thesis, Institute of Computer Science, LMU, Munich, 2004. PhD Thesis, Institute for Informatics, University of Munich, 2004.
119. S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Extreme Markup Languages*, 2004.
120. S. Schenk and S. Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, editors, *WWW*, pages 585–594. ACM, 2008.
121. M. Sintek and S. Decker. Triple - an rdf query, inference, and transformation language. In *INAP*, pages 47–56, 2001.
122. M. Sintek and S. Decker. Triple - a query, inference, and transformation language for the semantic web. In I. Horrocks and J. A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 2002.
123. M. K. Smith, C. Welty, and D. L. McGuinness, editors. *OWL Web Ontology Language Guide*. W3C Recommendation. World Wide Web Consortium (W3C), 2004.
124. T. Swift. Deduction in ontologies via asp. In V. Lifschitz and I. Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 275–288. Springer, 2004.
125. H. J. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *ISWC*, volume 3729 of *Lecture Notes in Computer Science*, pages 668–684. Springer, 2005.
126. H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *J. Web Sem.*, 3(2-3):79–115, 2005.
127. J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.
128. Y. Wilks and C. Brewster. Natural Language Processing as a Foundation of the Semantic Web. *Foundations and Trends in Web Science*, 1(3-4):199–327, 2009.
129. Y. Wilks and C. Brewster. *Natural Language Processing as a Foundation of the Semantic Web*. Now Publishers Inc., 2009.
130. G. Yang and M. Kifer. Reasoning about anonymous resources and meta statements on the semantic web. *J. Data Semantics*, 1:69–97, 2003.