REASONING ON THE SEMANTIC WEB: BEYOND ONTOLOGY LANGUAGES AND REASONERS

François Bry
University of Munich, Germany
http://www.pms.ifi.lmu.de/

Massimo Marchiori
University of Venice, Italy, and W3C
http://www.w3.org/People/Massimo/

Abstract

This article discusses forms of reasoning (beyond ontology reasoning), and reasoning languages (beyond ontology languages), that are needed for a full deployment of the Semantic Web. We first outline a motivating application scenario, then discuss the logic languages needed on the Semantic Web and related issues. The views reported about in this article underly and have been developed within the EU research project REWERSE (REasoning on the WEb with Rules and SEmantics, cf. http://rewerse.net).

1 Motivating Application Scenario: Business Rules for Car Rental

Consider *business rules* of a car rental company specifying 'price regulations for company customers'. In a simplified form, such business rules, inspired from [1], can be as follows:

- No more than 20 cars can be rented at once by a (company) customer at a reduced company price.
- Reduced company prices are not applicable on Class A cars.

Beside regulations for company customers, the car rental company might have specified other business rules expressing 'general car renting regulations':

- No more than one car can be rented at once by a person.
- Price reductions are not applicable to young drivers, i.e. drivers under 25 years.

Note that the above two sets of business rules, the 'price regulations for company customers' and the 'general car renting regulations' can well be at distinct web sites, reflecting that two different departments of the car rental company at different locations are responsible for these two sets of business rules.

A certain company customer of the car rental company might also well have its own 'car rental regulations', e.g.:

- Junior employees are at most 24 years old.
- Every employee is junior or senior employee.
- Senior employees rent only Class A cars.

Note that the above regulations are expressible in OWL-DL but not in RDF, as quantifiers, cardinalities, and negation are needed.

Consider now the *forms of reasoning* that might be needed in processing business rules like the above regulations:

- To draw the conclusion that an employee older than 25 years is a senior employee, *excluded middle*[1] is needed.
- To draw the conclusion that no junior employees can rent a car on the (reduced) company price, *refutation*[2] and *monotonic negation*[3] are needed.
- To check whether a regulation, say *"No more than 20 cars can be rented at once by a (company) customer at a reduced company price."*, is currently enforced[4] *non-monotonic negation*[5] is needed.
- To determine so-called 'gold customers', i.e. customers renting cars at least 5 times during the last 12 months, *constructive reasoning* suffices, i.e. excluded middle and refutation are *not* needed.

2 Reasoning Languages for the Semantic Web

The Semantic Web requires languages of different kinds:

1. Deductive languages of three kinds:

[1] At least one of A and $\neg A$ is true.

[2] If under the assumption A, a contradiction, i.e. B and $\neg B$ for some B, can be derived, then $\neg A$ is proven.

[3] The negation of classical logic and mathematics.

[4] For sure, such a regulation might will be violated from time to time!

[5] The negation used in concluding that flights not mentioned in a time table do not exist.

(a) *constructive rules* (also called *views*),
(b) *normative rules* (also called *integrity constraints*),
(c) *descriptive specifications* (also called *ontologies*),

2. and *Reactive rules.*

Consider the motivating scenario of the previous section. The definition of 'gold customers' requires a constructive rule or view. The regulation *"No more than 20 cars can be rented at once by a (company) customer at a reduced company price."* is a normative rule or integrity constraint. The three sets of business rules are descriptive specifications or ontologies. Reactive rules are needed in specifying workflows, e.g. how to process car rental orders or how to move cars to places where customers will collect them.

Let us discuss in more detail the kinds of languages the Semantic Web requires.

Constructive rules serve to specify new data from data already available. The name 'constructive rules' stresses that consequences from such rules can be drawn in constructive logic, i.e. without relying on excluded middle or refutation. Such rules are called 'views' in databases. Constructive rules typically involve data selection and grouping. Constructive rules are often, but not always, expressed as implications of the form `new-data` $\Leftarrow$ `query`.

Normative rules, called 'integrity constraints' in databases, express conditions that data must fulfill, e.g. customer numbers uniquely characterize customers, and that must be checked when data are updated. Data schemas, e.g. a DTD or an XML Schema, express normative rules. Normative rules can be expressed as 'denials' and evaluated like constructive rules. A denial is a rule of the form `false` $\Leftarrow$ `query` where the head `false`, or `error(...)`, etc., denotes a violation of a requirement `req` and the denial's body `query` expresses a negation of this requirement, i.e. `query` $\equiv \neg$`req`. E.g. the following denial expresses that customer numbers uniquely characterize customers: `error(Nb)` $\Leftarrow$ `customer(Name1, Nb)` $\wedge$ `customer(Name2, Nb)` $\wedge$ `Name1` $\neq$ `Name2`.

Descriptive specifications specify data types and relationships between data types without necessarily referring to actual data. They are used in software specifications, data schemas, and ontologies. They are often expressed in logics[6] corresponding to classical logic fragments with *restricted quantifications* of the forms $\forall x : s\ F[x]$ and $\exists x : s\ F[x]$ restricting the variable x to some sort, class, entity, etc. s. Such quantifications can be expressed in classical logic as $\forall x\ s(x) \Rightarrow F[x]$ and $\exists x\ s(x) \wedge F[x]$, resp. using a conveniently defined unary predicate symbol s.

Reactive rules specify how a data store can be modified depending on the current state of the store and, in some languages, on events. Reactive rules commonly have one of the forms `if condition then action` and `on event if condition then action`. Rules of the first kind are called *production rules* [2], rules of the second kind are called *ECA* (short for *Event-Condition-Action*) *rules*. In production and ECA rules, `condition` is an (atomic or compound) query to the data store similar to a body of a constructive or normative rule, and `action` is an atomic (i.e. single) or compound update of the data store (typically consisting of insertions, removal, and/or changes in a data item). In an ECA rule, `event` denotes an *event query*, i.e. a query to events received so far. An event query can be atomic, i.e. refer to a single event, or compound, i.e. refer to composite events. In the following, the condition of a production or ECA rule is called *standard query* so as to stress its similarity with the body of a constructive or normative rule.

Specifications are often not inherently 'normative' or 'descriptive specifications'. In many cases, the distinction between normative rules (integrity constraints) and descriptive specifications (ontologies) subtly depends on the use. Consider a system of rules expressing some regulation, e.g. under which conditions students are allowed to register for courses. In drawing conclusions from the regulation, or in verifying that it is consistent or non-redundant, the regulation is used as a descriptive specification – certain forms of reasoning such as excluded middle and refutation make sense and might even be indispensable. In verifying that student registrations to courses enforce the regulation, the regulation is used as integrity constraint – excluded middle and refutation do not make sense.

3 Negations on the Semantic Web

Two main kinds of negation have been considered in applications of automated reasoning: *non-monotonic* and *monotonic negation.*

Non-monotonic negation, cf. [3] for selected articles, is the negation (mostly) used in databases. It is used e.g. in concluding that flights not mentioned in a time table do not exist. It is called non-monotonic because adding new premises, e.g. adding a flight to a flight F time table, withdraw a conclusion, in the example mentioned $\neg F$ does not longer holds after the addition of flight F.

Monotonic negation is the negation used in classical logic and mathematics. With monotonic negation, all consequences still hold after adding premises. If a consequence $\neg F$ follows from a set of premises $\mathcal{S}$, then with monotonic negation $\neg F$ also follows from $\mathcal{S} \cup \{F\}$. Monotonic negation is (implicitly) used in excluded middle and in refutation reasoning, cf. Section 1.

Non-monotonic negation is the negation of choice for *constructive rules (views)* because data construc-

[6] E.g. sorted logics and description logics.

tions depends on both, available and non-available data. Since normative rules can be expressed as constructive rules (cf. Section 2), non-monotonic negation is also the negation of choice for normative rules. Non-monotonic negation is the negation of choice for reactive rules, too, for both 'event queries' (i.e. the `event` parts of ECA rules) and 'standard queries' (i.e. the `condition` parts of production or ECA rules).

Monotonic negation is the negation of choice for *descriptive specifications (ontologies)* because descriptive specifications do not refer to actual data, e.g. the flights listed in a time table, but instead to meta-level specifications, e.g. conditions flights must fulfill, the negation needed in descriptive specifications does not have to refer to the absence or non-availability of such data.

Recall, cf. Section 2, that the same rule can be used as a normative specification (integrity constraint) or descriptive specification (ontology). As a consequence, the choice of a negation semantics, monotonic or non-monotonic, does not necessarily depend on the *syntax* of negation, but instead on the *use* of the expression in which the negation occurs.

4 Declarative and Operational Semantics, Event Processing

It is desirable that reasoning languages for the Semantic Web have *declarative semantics* defined as 'Tarski-style model theories'. *Tarski-style models* [4], i.e., the models of classical logic, are expressed in terms of so-called 'valuation functions' that are defined recursively on a formula's structure. They make possible to evaluate a formula independently of other formulas. Therefore, they are easy to understand, and they do not require complex operational semantics. Note that most declarative semantics for non-monotonic negation that do not assume stratified, or stratifiable, rules, e.g. the stable [5] and well-founded [6] semantics, do *not* have Tarski-style model theories.

Production and ECA rules usually amount to *imperative programming*, hence they look inherently not amenable to declarative semantics. However,

- declarative semantics are possible and desirable for the 'standard query' and 'event query' languages used in production or ECA rules languages,
- a formal semantics amenable to reasoning on production and ECA rule programs is possible (and desirable!).

On the Web, forward chaining is well-suited only for well-defined and closed sets of Web sites. Queries referring directly or indirectly (through sub-queries triggered by constructive rules at queried Web sites) to a set of Web sites that cannot be recognized statically, i.e. before query evaluation, cannot consequently be evaluated by forward chaining. Indeed, with such queries, forward chaining would require to compute intermediate results from all possible Web sites. Thus, on the web, backward chaining is the reasoning paradigm of choice for constructive and normative rules.

Theory reasoning, a term coined after Mark Stickel's 'theory resolution' [7], denotes enhancing a general purpose reasoning method with special reasoners where convenient, e.g., reasoning on bank accounts with a basic arithmetic 'theory reasoner' instead of the Peano axioms of Arithmetic.

The operational semantics of a logic language is conveniently expressed with constructive and normative rules. Backtracking is useful for a fine tuning of proof construction in implementing logic languages. Backtracking is however undesirable as a programming concept for high-level logic languages like the logic languages needed on the Semantic Web because it destroys the language's declarativity. The operational paradigm(s) desirable for a Semantic Web logic language can be equivalently called 'backtracking-free logic programming' or 'set-oriented functional programming'. It is worth noting almost all of the query languages proposed for RDF are of this kind.

The operational semantics of a logic language or reasoner is usually and conveniently expressed in terms of inference rules of the form:

$$\frac{\texttt{Premise}_1 \ldots \texttt{Premise}_n}{\texttt{Conclusion}}$$

Inference rules can be seen as constructive rules in a meta-language specifying proofs for formulas of the object-level language. Thus, constructive rules are subjacent to (the procedural semantics of) *every* rule language and reasoners. This observation has led to successful uses of the run-time system [8] of Prolog or of the Prolog language itself [9] for implementing efficient theorem provers. Normative rules, too, are convenient in specifying the procedural semantics of rule languages and reasoners for expressing constraints on the proof, or search, space. Reactive rule can be convenient in implementing logic languages and reasoners.

Since constructive and reactive rule languages can be used in specifying and implementing logic languages and reasoners, some authors have claimed that a single language of such a kind would be sufficient for the Semantic Web. This amounts to claiming that only one single, e.g., imperative, programming language could be sufficient for developing software.

Event broadcasting is undesirable on the Web. Events can be exchanged between Web sites using a push, or a pull model. Pushed events can be sent as data streams, calling for streamed query evaluation methods. Evaluating event queries, e.g. the event parts of ECA rules, calls for event driven query evaluation methods.

On the Web, events can not be broadcasted, i.e. indiscriminately sent to all sites, because this would result in too high a traffic. Generally, events can be exchanged on the Web sites via either push, i.e. events are sent by the emitters to specific recipients, or pull methods, i.e. each site publishes the events it emits, together with the event's recipients, on a 'blackboard' which is repeatedly queried by the potential recipient sites. Such queries are called *continuous*. With the push model, events can be sent as 'data streams' [10]. Continuous queries [11, 12, 13, 14], data streams [10], and event queries [15, 16] require specific query evaluation methods.

5 Data and Data Processing

A Semantic Web reasoning or reactive language must

- be capable of accessing data everywhere on the Web
- be 'data versatile', i.e. capable of accessing data and meta-data in any common Web Semantic Web format – especially XML, RDF, Topic Maps, and OWL, as well as the formats of Semantic Web logic languages
- be capable of (some forms of) meta-level reasoning.

There has already been a number of pleas in favour of data versatile query languages, e.g. [17].

Meta-level reasoning poses interesting, but not impossible, challenges. Meta-level reasoning has bad reputation among Computational Logicians: however, conveniently (e.g. constructively) restricted, *cf.* [18], meta-level reasoning is semantically as safe, and practically as useful as higher-order functions in Functional Programming. Note that meta-level reasoning is already present, though in a limited form, on the Semantic Web: RDF Schema, the "RDF Vocabulary Description Language", is itself an RDF Vocabulary for describing terms in an RDF vocabulary.

6 Language Coherency, Inter-Operability, Engineering, and Rendering

Inter-operable languages of various kinds should be striven for. Inter-operability is sustained by the following forms of coherency: syntax coherency, rendering coherency, reasoning coherency, and explanation coherency.

Syntax coherency means that expressions from different languages with similar meanings are expressed similarly.

Rendering coherency means that expressions from different languages are (visually or verbally) rendered similarly, possibly using the same rendering methods or tools.

Reasoning coherency means that similar forms of reasoning applied on different languages, e.g. for deriving new data using constructive rules, for computing the closure of RDF specifications, or for checking normative rules, are performed using similar reasoners. Reasoning coherency is desirable both for programmers and language design, and implementation. An important aspect of reasoning coherency is to have a common semantics for non-monotonic negation in constructive, normative, and reactive rule languages.

Explanation coherency means that similar forms of reasoning are explained, by explanation tools, relying on similar explanation paradigms.

Reasoning languages for the Semantic Web should be referentially transparent, strongly closed, have Web formats, and modern type systems, i.e. type systems supporting abstract data types and offering static type checking, parametric polymorphism, and modules. The specification of abstract machines should be striven for because abstract machines are extremely useful for wide-spreading languages.

Referential transparency, i.e. when two occurrences of a same expression within a same declaration scope have the same meaning, is desirable because it is *the* trait of declarativity. *Closure*, i.e. when the data returned by a program are in the same format as the data accessed by programs in the same language, is also very important. *Strong closure* means that the data returned by a program can be further processed by this same program. Strong closure is desirable because it eases structuring programs in sub-programs. *Web formats*, especially XML formats such as RuleML formats, are desirable for rule languages because they eases inter-changing programs on the Web, e.g., for Web services applications. *Abstract data types* and *static type checking* are desirable for Semantic Web reasoning and reactive languages as they are for any other programming languages: *"Well typed programs do not go wrong."* [19]

Reasoning languages for the Semantic Web should have *visual* and *verbal renderings.* Declarative languages are especially well-suited to visual rendering and visual rendering is very appealing to potential users of logic languages for the Semantic Web, as the many systems for graphical rendering and/or visualization of business rules amply demonstrate. Programs used on the Web and Semantic Web should be *verbalizable*, i.e. the rules or formulas they consist of should be expressible in a controlled language [20, 21], i.e. in a non-ambiguous language resembling natural language. Rules (e.g. expressing policy specifications and trust) verbalized in a controlled language would considerably help wide-spreading the (verbal as well as non-verbal forms of the) languages they are expressed in.

Acknowledgments

The authors thank their colleagues from REWERSE for many fruitful discussions on the subject of this article.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (*cf.* `http://rewerse.net`).

References

[1] The Business Rules Group: Defining Business Rules – What Are They Really? `http://www.businessrulesgroup.org/first_paper/br01c0.htm` (2001)

[2] Brownston, L., Farrell, R., Kant, E., Martin, N.: Programming Expert Systems in OPS5: An Introduction to Rule-based Programming. Addison-Wesley (1985)

[3] Dix, J., Pereira, L.M., Przymusinski., T.C., eds.: Selected Papers from the Non-Monotonic Extensions of Logic Programming. LNCS 1216. Springer-Verlag (1996)

[4] Keisler, J.: Fundamentals of Model Theory. In: Handbook of Mathematical Logic. North-Holland (1989) 47–103

[5] Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Proc. Int. Conf. and Symp. Logic Programming. (1988)

[6] Gelder, A.V., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. Jour. ACM **38** (1991) 620–650

[7] Stickel, M.E.: Automated Deduction by Theory Resolution. Jour. of Automated Reasoning **1** (1985) 333–355

[8] Stickel, M.E.: A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Computer. Jour. of Automated Reasoning (1988)

[9] Manthey, R., Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog,. In: Proc. 9th Conf. on Automated Deduction. (1988)

[10] Bry, F., Coskun, F., Durmaz, S., Furche, T., Olteanu, D., Spannagel, M.: The XML Stream Query Processor SPEX. In: Proc. 21st Int. Conf. on Data Engineering (ICDE). (2005)

[11] Terry, D., Goldberg, D., Nichols, D., Oki, B.: Continuous Queries over Append-Only Databases. In: Proc. ACM SIGMOD Int. Conf. on Management of Data. (1992)

[12] Babu, S., Widom, J.: Continuous Queries over Data Streams. SIGMOD Record (2001)

[13] Nguyen, B., Abiteboul, S., Cobena, G., Preda., M.: Monitoring XML Data on the Web. In: Proc. ACM SIGMOD Intl. Conf. on Management of Data. (2001)

[14] Pandey, S., Krithi Ramamritham, a.S.C.: Monitoring the Dynamic Web to Respond to Continuous Queries. In: Proc. 12th Int. World Wide Web Conference. (2003)

[15] Bry, F., Pătrânjan, P.L.: Reactivity on the Web: Paradigms and Applications of the Language XChange. In: Proc. 20th Annual ACM Symp. Applied Computing (SAC). (2005)

[16] Bailey, J., Bry, F., Pătrânjan, P.L.: Composite Event Queries for Reactivity on the Web. In: Proc. 14th Int. World Wide Web Conference. (2005)

[17] Robie, J.: The Syntactic Web: Syntax and Semantic on the Web. In: Proc. XML Conf. and Exposition. (2001)

[18] Chen, W., Kifer, M., Warren, D.S.: HILOG: A Foundation for Higher-Order Logic Programming. Jour. of Logic Programming **15** (1993) 187–230

[19] Milner, R.: Fuly Abstract Models of Typed λ-Calculi. Theoretical Computer Science **4** (1977) 1–22

[20] Marchiori, M., Saarela, J.: Query + Metadata + Logic = Metalog. In: Proc. QL '98, The Query Languages Workshop. (1998) `http://www.w3.org/TandS/QL/QL98/`.

[21] Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English – Not Just Another Logic Specification Language. In: Proc. 8th Int. Workshop (LOPSTR). LNCS 1559, Springer-Verlag (1999)