What should an ASP Solver output? A Multiple Position Paper *

Martin Brain[1], Wolfgang Faber[2], Marco Maratea[3], Axel Polleres[4], Torsten Schaub[5], and Roman Schindlauer[6,2]

[1] Department of Computer Science, University of Bath, United Kingdom
mjb@cs.bath.ac.uk
[2] Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy
faber@mat.unical.it
[3] DIST - University of Genova, Viale F. Causa 15, 16145, Genova, Italy
marco@dist.unige.it
[4] Digital Enterprise Research Institute, National University of Ireland, Galway
axel.polleres@deri.org
[5] Institut für Informatik, Univ. Potsdam, August-Bebel-Str. 89, D-14482 Potsdam, Germany, torsten@cs.uni-potsdam.de
[6] Institut für Informationssysteme 184/3, Technische Universität Wien, Favoritenstraße 9–11, A–1040 Vienna, Austria, roman@kr.tuwien.ac.at

Abstract. This position paper raises some issues regarding the output of solvers for Answer Set Programming and discusses experiences made in several different settings. The first set of issues was raised in the context of the first ASP system competition, which led to a first suggestion for a standardised yet miniature output format. We then turn to experiences made in related fields, like Satisfiability Checking, and finally adopt an application point of view by investigating interface issues both with simple tools and in the context of the Semantic Web and query answering.

1 Motivation

The development of solvers for Answer Set Programming (ASP;[1]) constitutes nowadays a major driving force of the field. This goes hand in hand with a growing range of applications along with more and more substantial collections of benchmark suites. The latter allow for a broad comparison among different ASP solvers. And benchmarking as such plays a major role for progressing ASP solver technology, as already experienced in many related areas, such as Automated Theorem Proving [2] or Satisfiability Checking [3]. Although many benchmarks stem from distinguished application areas, certain applications need dedicated formats due to sophisticated interactions with ASP solvers. This is an important

* This work was partially supported by the Austrian Science Fund (FWF) under grant P17212-N04, and by the European Commission through the IST Networks of Excellence REWERSE (IST-2003-506779).

issue when interfacing ASP solvers with other software modules in real-world applications.

This paper is one out of three position papers providing the basis for a discussion forum to be held on ASP languages at the occasion of the Workshop on *Software Engineering for Answer Set Programming* (SEA'07), co-located with the *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'07;[4]). While the two other papers offer perspectives on *input* and *intermediate languages*[7], we concentrate in what follows on issues related to the *output* of ASP solvers. We begin with a discussion on experiences made during the first ASP system competition. This is complemented in Section 3 by lessons learned in related fields, like Satisfiability Checking. Section 4 outlines the minimum requirements for an output format, based on end user experience. In Section 5 we discuss interface issues that arise in a particular area of application, namely the Semantic Web. Finally, in Section 6, we discuss requirements for the output of query answering, a major reasoning mode for ASP solvers.

2 Lessons from the First ASP System Competition

The first ASP system competition [5][8], held in conjunction with LPNMR'07, provided us with a first glance at issues arising from the distinct output formats of existing ASP solvers. For example, the original design of the underlying Asparagus platform [6][9] was based on *trust*, insofar as the output of a solver was never inspected.

This was changed when running the ASP system competition, for which the output of solvers had to conform to the following formats (in `typewriter` font):

SAT: `Answer Set: atom1 atom2 ... atomN`
: The output is one line containing the keywords '`Answer Set:`' and the names of the atoms in the answer set. Each atom's name is preceded by a single space. Spaces must not occur within atom names.

UNSAT: `No Answer Set`
: The output is one line containing the keywords '`No Answer Set`'.

The following list comments on some issues that arose during the competition; it also raises some further topics that might we worth considering in the future.

Result. The definition of the above basic output format was a big step forward in accessing the result of a solver's computation in a uniform way.

However, the distinction between satisfiable and unsatisfiable results quickly turned out to be insufficient. In future, we definitely need (at least) a third indicating string, say `UNKNOWN`, signalling that the solver terminated without

[7] That is, a language format for communication among grounders and solvers.
[8] `http://asparagus.cs.uni-potsdam.de/contest/`
[9] `http://asparagus.cs.uni-potsdam.de/`

finding a solution, although there might exist one. In fact, in the competition, we only had to deal with a single incomplete solver, whose output had then to be checked by hand. However, such an indicator also makes sense, for instance, in view of error handling (see below), where an encountered error should not lead to an indicative output (for instance, of **UNSAT**), simply because the wrapping script defaults to it.

Moreover, the competition only required the computation of a single answer set; hence, no format for producing all answer sets was put forward. On the other hand, printing a combinatorial elevated number of solutions is time consuming and presumably not necessarily desired in the context of a system competition. Also, it is unclear how the result could be verified in a reasonable amount of time (see below). Unlike this, however, it may be interesting to simply count the number of answer sets and have solvers report the number of answer sets they were able to compute within an allotted time. The number of solutions is actually relevant in some applications, like bio-informatics. On the other hand, it is unclear how such a result ought to be verified.

Certification. The second major advancement of the ASP competition was the verification of solutions. This worked, however, only for satisfiable instances, and it is yet an open problem how unsatisfiability should be certified.

The verification process builds upon the output of a *certificate*, given by an entire answer set or simply a subset of distinguished predicate instances representing a solution. (The latter was needed in the modelling track of the competition.) However, what should a solver output in case it treats an unsatisfiable instance? During the system competition, this issue was resolved pragmatically by trusting the majority of outcomes; and of course, whenever a single solver found a solution, we were able to check whether it was right.

Another major hurdle is given by the computational complexity underlying the verification problem. While it is easy for normal logic programs, it becomes significantly more difficult for disjunctive logic programs. But even in case of normal logic programs, we faced problem instances where the certificate became simply too large to be treated in a pragmatic fashion.

In a nutshell, the output of a certificate is essential for trusting the result of an ASP solver, however, there are still cases where it is yet unclear what constitutes a good certificate or at least a good approximation of it.

Performance. During the ASP competition, the performance of ASP solvers was measured externally by regarding the number of timeouts and, in case of ties, the run time of the respective solvers.

For a more fine-grained comparison among ASP solvers, one might be interested in some information that can only be gathered by the solvers itself. One such piece of information is the number of assignments, or to be more precise, the number of assignments due to propagation and the one due to heuristic choice operations, which could provide a much more detailed picture of the traversed search space. But apart from finding an agreement on the output format of such information, we first need solver developers to agree on collecting information

in the same way. For instance, systems like *smodels* or *dlv* report information about *choice points*, while having different definitions of what constitutes a choice point. Moreover, a system based on learning and back-jumping like *clasp* does not even (explicitly) flip assignments at choice points. Other solvers like *cmodels*, using SAT solvers as search engine, may not even have easy access to this type of information. So, this is an example where an agreement on an output format has to be preceded by a consideration of the underlying concepts.

Error handling. Asparagus controls the execution of each solver run by limiting it in time and space. An excess of either limit is detected and recorded by Asparagus.

In fact, some solvers, or to be more precise, their wrapping scripts, output **UNSAT** by default, even though they get interrupted by the run time system of Asparagus. Similarly, we had situations in which solvers returned within time and space limits, despite having encountered internal (e.g., input) errors. It becomes tricky when this happens with a solver whose wrapper reports **UNSAT** by default and which actually attempted to solve an unsatisfiable problem instance.

What is needed here is a systematic way of treating errors (or even termination) through appropriate signals. We need to define different error categories and how errors should be signalled (for instance, to *standard error* as opposed to *standard output*).

Moreover, it would be a great help, if solvers could receive termination signals, output some relevant information, and terminate by themselves.

Optimisation. The first edition of the ASP system competition dealt exclusively with decision problems, although more and more solvers allow for dealing with optimisation problems as well.

Although one may treat optimisation problems as decision problems, by asking whether a solution with optimal value of the objective function has been found, it is also of interest to regard the value of the best solution a solver came up with, even though it did not terminate within the allotted time. This is difficult because one needs an output from an externally terminated program (see above).

As with error handling, it makes sense to find a consensus of how to handle this problem in a uniform way.

3 Experiences from Related Areas

In this section, we will see how the issues that came up in the ASP Competition, and pointed out in Section 2, have been raised (assuming they are) in other research areas. It is interesting to note that also in other areas these issues showed up together with the definition and organization of Competitions/Evaluations.

SAT. Propositional Satisfiability (SAT) is one of the most studied problems in Artificial Intelligence and Computer Science. SAT Competitions[10] are organized

[10] `http://www.satcompetition.org/`

by years, with a great impact on the performance of SAT solvers. SAT output format already fixed the point about an **UNKNOWN** result by explicitly allowing it as a "valid" output, other than **SATISFIABLE** and **UNSATISFIABLE**. These words have to be put in a new line starting with "s" followed by a space, e.g., "s SATISFIABLE". Then, if a formula is satisfiable, a bunch of 0-terminated lines starting with "v" and representing a satisfying assignment has to be printed as certificate.

Moreover, in order to automatically check the correctness of solvers' answers, all solvers must also exit with an error code which characterizes its answer on the considered instance. This is done in addition to the automatic syntactic analysis of solvers' output. The error code must be:

- 10 for SATISFIABLE
- 20 for UNSATISFIABLE
- 0 for UNKNOWN
- any other error code for internal errors (considered as UNKNOWN)

The issue of certifying an unsatisfiable formula is not raised in the SAT Competitions (last year, SAT race[11]). Nonetheless, the need to cope with this problem is evident in the community and has opened the way to significant research efforts in this direction.

QSAT. Quantified SAT (QSAT) is the extension of SAT where variables are to be explicitly quantified, universally or existentially. QSAT is the prototypical PSPACE-complete problem. QBF Evaluations and Competitions[12] are organized since five years and have significantly contributed to this emerging research area. The output format requested to QBF solvers is very similar to the one for SAT solvers (the output must be 1, 0 or -1 instead of SATISFIABLE, UNSATISFIABLE and UNKNOWN, respectively). A main difference arises when certifying a formula: given the complexity of the problem, no compact certification is known. At the moment, QBF solvers output just a partial certificate of the input QBF's truth or falsity.

Beyond the already detailed explanation of the output format, the organizers of the QBF events have made available a "formal" description of such an output format, using a BNF grammar.[13] This document can be very useful for both competitors and organizers, in particular, when the "complexity" of the output increases, which is the direction a future output format is likely to follow for expressing a non-trivial form of certification.

PB. In Pseudo-Boolean (PB) (optimization) problems, solvers have to satisfy a set of on linear inequalities (with Boolean variables), while optimizing an objective function. The PB07 Evaluation[14] is the third event of the series. Given

[11] `http://fmv.jku.at/sat-race-2006/`
[12] `http://www.qbflib.org/`
[13] `http://www.qbflib.org/qdimacs.html`
[14] `http://www.cril.univ-artois.fr/PB07/`

the nature of the problem, solvers can output a new type of solution line, i.e., "s **OPTIMUM FOUND**", when they claim to have found the optimal value for the objective function. PB Evaluations introduced a nice idea related to the optimization of solutions: each solver is asked to output a line starting with 'o' each time it finds a solution with a better value of the objective function, even if it might not be optimal. This line should only contain the value of the objective function for the new solution. This enables an analysis of the way solvers progress toward the best solution. The utility of this information is (at least) twofold: given the simplicity, a graphical view on the progression toward the best solution can be provided, it is easy to (*i*) better understand a solver's behavior, and (*ii*) perform a deep analysis that can be used, for example, in the report of the competition.

Other series of Competitions/Evaluations can be interesting in the way they (try to) certify solutions, often related to the "complexity" of the problems.

In the Deterministic track of the International Planning Competition (IPC) competitions (the last being IPC-5[15]), the found plan is printed into a solution file and then checked by a plan validator made available by the IPC organizers. In the SMT Competitions[16] (SMT-COMP), a solver has to find solutions to formulas from decidable (quantifier-free) fragments of first-order logic, allowing for theories, like arithmetic, uninterpreted function, arrays, bit vectors, and their combinations. The SMT-COMP organizers ask for "suitable evidence" of the results, allowing for a "third-party proof checker publicly available, or a source code for it", and asking for an explicit option of the solver ('−−evidence') to dump the proof/model into a file because of the possibly huge size. Then "the verification is let to a Competition panel, separately to the main part of the competition" and "check is to be performed on small formulas". The CADE ATP System Competition (CASC) is related to first-order Automated Theorem Proving. Given the complexity of the problems, the organizers just "look for 'acceptable' proof/model".

Finally, the International Competition of Constraint Satisfaction Problems (CSP)[17] uses XML format, in this case to represent input instances. It could be fruitful to broaden the use of such a format: a motivating example for such a direction can be found in the next section.

4 An End User Perspective

From the point of view of an end user of answer set solvers, a standardised output format is highly desirable but raises two important questions: what output from a solver is needed and what is commonly done with this information? Output can be broken into three categories:

[15] `http://zeus.ing.unibs.it/ipc-5/`
[16] `http://www.csl.sri.com/users/demoura/smt-comp/`
[17] `http://www.cril.univ-artois.fr/CPAI06/`

1. Zero or more answer sets or a message saying that there aren't any answer sets.
2. Optionally an error message of some sort (most commonly out of time or out of memory).
3. Optionally some statistics.

Obviously as more sophisticated approaches to computing answer sets are developed, new types of information may be output (for example, problem specific analysis results of tuning parameters), but most applications current use only these three areas.

In turn, there are three common uses of this information (and thus three key requirements for the output format):

1. Answer sets are read by a human. Either to find out the answer to the initial problem, or to diagnose problems with the encoding. Thus a human readable format would seem to be a requirement.
2. Answer sets are ignored (or quickly checked), only the statistics are used. This is the common case in the development and benchmarking of solvers. Thus some way of quickly extracting the statistics would seem to be a good idea.
3. The output is parsed into another program[18] for further interpretation / application. Thus a format which is easy to parse would seem to be a requirement.

Additionally, there are practical arguments for keeping the output format as simple as possible (so more complex output formats can be layered over them with minimal overhead) and for minimising the amount of modification required for existing solvers.

Given these options, the simplest output standard seem to be roughly as follows:

- The success of the solver is given by it's system return code. 0 for 1 or more answer sets given, 1 for a program with no answer sets and any other return code constituting an error. This is in keeping with the POSIX standard, GNU/Linux implementations (a process killed due to signals, i.e. out of time, out of memory, etc. can be recognised from it's return code) and requires little to no extra implementation.
- Output is divided into lines, each prefixed by one of a number of codes. The actual codes aren't particularly important, but keeping to either the existing convention of human readable strings (i.e. `Answer Set`) or the SAT convention of single letters, seems sensible.

  `Answer Set :` Indicates the solver has found an answer set, which is given as a space separated list literals in the answer set. If any lines of this kind are present, the return value of the solver must be 0 and no lines starting `No Answer Sets` should be present.

[18] In this case, often the solver is being called from another program thus a standard calling convention and some standard option flags would be useful.

`No Answer Sets` Indicates the solver has shown that the program contains no answer sets. The line should contain nothing else. If any lines of this kind are present, the return value of the solver must be 1 and no lines starting `Answer Set :` should be present.

`Statistic :` Indicates a solver generated statistic, which should be given on the rest of the line, preferably in a form that could be easily parsed. As an appendix to the standard, a list of statistic names and how they are computed would make analysis easier.

`Comment :` A catch all field for other solver output intended for humans.

Any other line would be a considered an error message and the solver must return something other than 0 or 1.

5 Interfacing the Semantic Web

In recent years, several endeavours have been undertaken to deploy ASP in the area of the Semantic Web, as a powerful rule language to complement and extend the possibilities of established formalisms such as RDF and ontology languages. This development requires ASP solvers to interoperate with other software in a complex reasoning framework. Due to the heterogeneity of data in the domain of the Semantic Web, the most straightforward approach to such interfacing tasks is usually to use an XML-based language as data interchange format.

A prominent attempt to create a Web-suitable syntax for rule languages in general is the RuleML initiative [7], which aims at providing a Rule-Markup Language based on XML and/or RDF, in order to facilitate a common representation and exchange of rules. Also, the chosen format allows the possibility of annotating further information, as needed in the Semantic-Web context. However, it is not trivial to embed in a general framework the wide number of pre-existing rule-based formalisms, each of which provides its own variety of syntactic features. Thus, different classes of RuleML languages have been gradually introduced, in order to support constructs such as default negation or constraints.

One particular such branch of RuleML tailored to ASP and its extensions has been presented in [9]. There, the authors integrate a general construct into the framework of RuleML that can be used to express features such as built-in predicates, external atoms, or cardinality constraints. However, in this work the authors do not explicitly consider to encode the output of an ASP solver in RuleML. This can in fact be accomplished by using the notions of RuleML atoms, conjunction, disjunction, and negation. In general, RuleML does not impose specific semantics on its constructs; however, for the subset of operators needed to represent answer sets and our purposes, the intuition is rather straightforward. Atoms within the same answer set are connected by conjunction, while multiple answer sets are joined by disjunction.

For instance, a single atom $edge(a, b)$, i.e., a positive literal, is expressed in RuleML as follows:

```
<Atom>
  <Rel>edge</Rel>
  <Ind>a</Ind>
  <Ind>b</Ind>
</Atom>
```

The Tag `<Rel>` denotes the atom's predicate name, while `<Ind>` surrounds individual constants.[19] An atom is negated by embedding it into a `<Neg>` tag. A conjunction of atoms is expressed by an enclosing `<And>` tag, a disjunction by `<Or>`. Thus, the single answer set $\{edge(a, b), edge(a, c), color(a, blue)\}$ would be written as

```
<Assert>
  <And>
    <Atom>
      <Rel>edge</Rel>
      <Ind>a</Ind> <Ind>b</Ind>
    </Atom>
    <Atom>
      <Rel>edge</Rel>
      <Ind>a</Ind> <Ind>c</Ind>
    </Atom>
    <Atom>
      <Rel>color</Rel>
      <Ind>a</Ind> <Ind>blue</Ind>
    </Atom>
  </And>
</Assert>
```

The outermost `<Assert>` tag acts as a wrapper and denotes a declarative content.[20] A complete result by an ASP solver comprises several answer sets, joined by a disjunction:

```
<Assert>
  <Or>
    <And> ... </And>
    <And> ... </And>
  </Or>
</Assert>
```

An empty answer set corresponds to an empty `<And>` tag, while an empty `<Or>` clause denotes an empty result, i.e., no answer set.

[19] In the context of the Semantic Web, individual names might well contain special chars, for example URIs of resources. In XML we can simply encapsulate such strings within CDATA sections.

[20] `<Assert>` provides an attribute `mapClosure` to specify existential or universal closure within the assertion, but since ASP results are currently always ground, we can omit this information.

Currently, this output format is supported by the HEX-program solver dlvhex.[21] HEX-programs are an extension of ASP towards interoperability in the Semantic Web [8], providing a mechanism to exchange knowledge with external sources of information.

Other ongoing streams in the Semantic Web realm

Standard formats. Apart from RuleML, there are other ongoing efforts worthwhile to monitor in the context of how we could interface with the Semantic Web. First of all, the Rule Interchange Format (RIF) working group[22] is producing first results toward channeling various proposals, of which RuleML is only one into a real standard for exchanging rules. Emerging formats from this group will likely replace attempts like RuleML which were not governed by an official standardization body. Whereas RIF will likely be a good candidate for Web exchange of answer set programs, the exchange of results of the evaluation of rules though is not (yet) an explicit goal yet in RIF, but will likely arise as soon as people start to pick up these formats to a larger extent.

Standard formats. The Semantic Web and Web 2.0 ideas go towards piping results between different distributed applications. Such applications do not only require standard input and output formats, but moreover standard protocols and interfaces to be used. A good example for the definition of such normative interfaces and protocols is provided by W3C's Data access working group, who are in charge of defining SPARQL[23], a standard query language for RDF. However, they also went one step further, defining a protocol along with defining both the concrete message formats for sending a query and receiving the results to a SPARQL endpoint, ie. an online interface for a SPARQL-capable query engine. The definiton of such standard interfaces, makes smooth interplay of semantic Web interfaces possible which can be invoked via standard Web Service interfaces in the Web Services Definition Language (WSDL[24]).

When we think about defining defined standard input and output formats for ASP-solvers, we also might think about extending such interface definitions like the one defined for SPARQL toward standard Web service interfaces for ASP solvers in order to make them accessible within the service-oriented world [10].

6 Query Answering

While many answer set solvers currently focus on computing answer sets, there are applications that require query answering, among them Information Integration [11], Enterprise Information Systems [12], and Text Classification [13]. In this section, we will mainly discuss in which way the output requirements for

[21] `http://www.kr.tuwien.ac.at/research/dlvhex/`
[22] `http://www.w3.org/2005/rules/`
[23] `http://www.w3.org/TR/rdf-sparql-query/`
[24] `http://www.w3.org/TR/wsdl`

query answering differ from those for answer set generation. In particular, we shall argue that calculating query answers from a standard answer set output in an easy way is not feasible in all cases, thus giving rise to a native query answering output mode.

Given the fact that there may be any number of stable models, there is no unique way of defining the consequence relation which is used for answering queries. Traditionally, there are two major reasoning modes: *Brave* (also known as *credulous*) and *cautious* (also known as *skeptical*) reasoning. For brave reasoning, a formula follows from a program if it holds in one of the answer sets of the program, while for cautious reasoning a formula follows if it holds in all answer sets.

Query answering is then defined as the set of ground substitutions over variables in the query formula, such that the substituted formula follows (bravely or cautiously) from the program. For ground queries, this means that the answer is either the empty set (corresponding to "no") or a set containing the empty substitution (corresponding to "yes"). Usually, as for rules, queries are required to be *domain independent*, that is, the query answer must not depend on the domain chosen to interpret the program. In practice, queries are required to be *safe* (cf. [14]).

An important observation is that complex query formulas can be rewritten by means of additional fresh predicates and rules to programs with an atomic query, which does not contain constants (or function symbols). Ground queries therefore are reduced to a query containing a predicate of arity 0. Let this predicate be p, one can simulate brave reasoning by adding a constraint $\leftarrow \texttt{not}p$ to the program and checking whether this program has an answer set, answering with the empty substitution if it does. For cautious reasoning, one may add $\leftarrow p$ to the program and check whether this program has an answer set, answering with the empty substitution if it does not.

For nonground queries, such a simple simulation is not easily possible. For brave reasoning, one could compute the answer sets projected onto the query predicate by eliminating duplicates and extracting the substitutions from the resulting set. For cautious reasoning, things are not as easy; for example, if there is no answer set, the answer should comprise all possible substitutions over the Herbrand Universe. As a result, especially for cautious reasoning, just providing all answer sets does not appear like an acceptable solution.

Concerning the representation of the output for query answering, the query language SPARQL, which has already been mentioned in Section 5, also defines a format for query results[25]. As an example, two substitutions for variables `X` and `Y`, where one substitutes `a` for `X` and `b` for `Y`, and the other one substitutes `b` for `X` and `c` for `Y`, would be represented as follows:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
```

[25] `http://www.w3.org/TR/rdf-sparql-XMLres/`

```
    <variable name="X"/>
    <variable name="Y"/>
  </head>
  <results ordered="false" distinct="true">
    <result>
      <binding name="X">a</binding>
      <binding name="Y">b</binding>
    </result>
    <result>
      <binding name="X">b</binding>
      <binding name="Y">c</binding>
    </result>
  </results>
</sparql>
```

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Sutcliffe, G.: CASC-J3 — The 3rd IJCAR ATP System Competition. In Furbach, U., Shankar, N., eds.: Proceedings of IJCAR. Springer (2006) 572–573
3. Berre, D.L., Simon, L., eds.: In Berre, D.L., Simon, L., eds.: Special Volume on the SAT 2005 Competitions and Evaluations. Journal on Satisfiability, Boolean Modeling and Computation, IOS Press (2006)
4. Baral, C., Brewka, G., Schlipf, J., eds.: Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), Springer (2007) To appear.
5. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In [4] To appear.
6. Borchert, P., Anger, C., Schaub, T., Truszczyński, M.: Towards systematic benchmarking in answer set programming: The Dagstuhl initiative. In Lifschitz, V., Niemelä, I., eds.: Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04). Springer (2004) 3–7
7. Boley, H., Tabet, S., Wagner, G.: Design Rationale for RuleML: A Markup Language for Semantic Web Rules. In: Proceedings of the first Semantic Web Working Symposium (SWWS'01). (2001) 381–401. See also `http://www.ruleml.org`.
8. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: Proc. IJCAI 2005, Morgan Kaufmann (2005) 90–97
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A RuleML Syntax for Answer-Set Programming. In Polleres, A., Decker, S., Gupta, G., de Bruijn, J., eds.: Informal Proceedings of the Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS'06). (2006) 107–108
10. Papazoglou, M.P., Georgakopoulos, D.: Service Oriented Computing. Comm. ACM, vol. 46, no. 10, (2003) 25–28
11. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete

and Inconsistent Data In: Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005). (2005) 915–917
12. Ruffolo, M., Manna, M.: A Logic-Based Approach to Semantic Information Extraction In: ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration. (2006) 115–123
13. Cumbo, C., Iiritano, S., Rullo, P.: OLEX - A Reasoning-Based Text Classifier In: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Proceedings. (2004) 722–725
14. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases Addison-Wesley (1995)