Systems, Engineering Environments, and Competitions

Yuliya Lierler
University of Nebraska at Ohama
ylierler@unomaha.edu

Marco Maratea
University of Genoa, Italy
marco@dibris.unige.it

Francesco Ricca
University of Calabria, Italy
ricca@mat.unical.it

Abstract

The goal of this paper is threefold. First, we trace the history of the development of answer set solvers, by accounting for more than a dozen of them. Second, we discuss development tools and environments that facilitate the use of answer set programming technology in practical applications. Last, we present the evolution of the answer set programming competitions, prime venues for tracking advances in answer set solving technology.

Introduction

Answer set programming (ASP) is a prominent knowledge representation paradigm that found numerous successful industrial and scientific applications including product configuration, decision support systems for space shuttle flight controllers, large-scale biological networks repairs, team building and scheduling (see the next article for more details). The success story of ASP is largely due to its modeling language and the availability of efficient and effective answer set programming tools that encompass grounders, solvers, and engineering environments. Syntactically, simple answer set programs (or ASP programs) look like Prolog logic programs. Yet, solutions to such programs are represented in ASP by sets of atoms called answer sets, and not by substitutions, as in Prolog. An answer set system typically consists of two tools, a grounder and a solver, and is used to compute answer sets. Since 2007, the series of ASP Competitions has promoted the collection of challenging benchmarks as well as supply researchers with a uniform platform for tracking the progress in the development of ASP solving technologies. More recent introduction of programming environments eased the development of ASP programs and the implementation of software systems based on ASP. In this paper, we present a brief survey of (i) existing answer set grounders and solvers, (ii) engineering tools and environments that support production of ASP-based applications, and (iii) ASP Competitions. Our goal is to provide an interested reader with an outlook on existing ASP technologies together with sufficient literature pointers rather than in depth explanation of research and engineering ideas behind these technologies.

ASP Grounders and Solvers

In ASP, solutions to logic programs are represented by sets of atoms called answer sets (stable models) (Gelfond and Lifschitz 1988). Answer set solvers, such as SMODELS (Simons, Niemelä, and Soininen 2002), SMODELS$_{cc}$ (Ward and Schlipf 2004), and DLV (Leone et al. 2006), to name some of the first implementations, compute answer sets of a given propositional logic program. Conceptually, most answer set solvers have a lot in common with satisfiability solvers (or SAT solvers), systems that compute satisfying assignments for propositional formulas in clausal normal form. Tools called *grounders* complement answer set solvers. A grounder is a software system that takes a logic program with variables as its input and produces a propositional program as its output so that the resulting propositional program has the same answer sets as the input program. Propositional programs are crucial in devising efficient solving procedures, yet it is the logic programming language with variables that facilitates modeling and effective problem solving in ASP.

There are three main grounders available for ASP practitioners: LPARSE (Syrjänen 2001), DLV-grounder (Leone et al. 2006), and GRINGO (Gebser, Schaub, and Thiele 2007). Grounders LPARSE and GRINGO are stand alone tools that are commonly used as front-ends for distinct answer set solvers. System DLV encapsulates both a grounder and a solver. However, calling the system with an option *-instantiate* produces propositional (ground) program for the given input and exits the computation without accessing the solving procedure of the system.

"Native" answer set solvers such as SMODELS, SMODELS$_{cc}$, DLV are based on specialized search procedures in spirit of the classic backtrack-search Davis-Putnam-Logemann-Loveland (DPLL) algorithm. The DPLL algorithm and its modifications are at the core of majority of modern SAT solvers. This algorithm consists of performing three basic operations: decision, unit propagate, and backtrack. Unit propagate operation is based on a simple inference rule in propositional logic that given a formula F in clausal normal form allows to utilize knowledge about unit clauses occurring in F or being inferred so far by the DPLL procedure in order to conclude new inferences. Native answer set solvers replace unit propagate of DPLL by specialized operations based on inference rules suitable

in the context of logic programs. For example, SMODELS implements five propagators called *Unit Propagate, All Rules Canceled, Backchain True, Backchain False, and Unfounded* (for details on these propagators see, for instance, (Lierler and Truszczynski 2011)). In DLV the basic chronological backtrack-search was improved introducing backjumping and look-back heuristics (Maratea et al. 2008). Solver SMODELS$_{cc}$ extends the algorithm of SMODELS by conflict-driven backjumping and clause learning. Clause learning is an advanced solving technique that originated in SAT and proved to be powerful. The distinguishing feature about the answer set solver DLV is its ability to handle *disjunctive* answer set programs. In rules of such programs a disjunction of atoms in place of a single atom is allowed in the heads. The problem of deciding whether a disjunctive program has an answer set is Σ_2^P-complete. The other systems capable of dealing with such programs are GNT, CMODELS, CLASPD, and WASP. Brochenin et al. (2014) surveys the key features of disjunctive answer set solvers.

Answer sets of a "tight" logic program are in a one-to-one correspondence with models of completion, a propositional logic formula proposed by Clark (1977). This observation immediately leads to an idea that answer sets of a tight logic program can be found by running a SAT solver on clausified program's completion. Tightness is a simple syntactic condition that many interesting ASP applications satisfy. An inception of a SAT-based answer set solver CMODELS (Giunchiglia, Lierler, and Maratea 2006) is due to this fact. It starts its computation by forming completion of an input program. Then CMODELS calls a SAT solver for enumerating models of program's completion. Lin and Zhao (2004) proposed a concept of loop formula so that given a program, extending its completion by its loop formulas results in a propositional formula, whose models are in a one-to-one correspondence with answer sets. In general case, the number of loop formulas can be exponentially larger than the size of a program. Nevertheless, solvers ASSAT (Lin and Zhao 2004) and CMODELS found means to utilize the concept of a loop formula in order to compute answer sets of a program. This computation typically requires multiple interactions with a SAT solver. Loop formulas are related to so called unfounded sets, which is the basis behind *Unfounded* propagator often employed in answer set solvers. Both ASSAT and CMODELS take advantage of conflict-driven backjumping and clause learning available in SAT technology that they rely on.

Answer set solver CLASP (Gebser, Kaufmann, and Schaub 2012a) borrows the ideas from both native and "loop formula"-based solvers. Just as CMODELS or ASSAT, it starts its computation by forming the clausified completion of an input program. Next it implements a search procedure that relies on a unit propagator stemming from SAT on the program's completion and an *Unfounded* propagator stemming from native answer set solvers. System CLASP implements conflict-driven backjumping and clause learning. The PC(ID)/answer set solver MINISAT(ID) (Wittocx, Mariën, and Denecker 2008) and WASP (Alviano et al. 2015) share a lot in common with the design of CLASP. Lierler and Truszczynski (2011) present a study that draws parallels between several answer set solvers.

System LP2SAT (Janhunen 2006) represents a family of "translation-based" solvers. This family relies on a translation of propositional logic programs into logic formulas so that models of the resulting formula are in one-to-one correspondence with the answer sets of the input program. This translation may add auxiliary atoms in the process and may include the normalization of aggregates as well as the encoding of level mappings for non-tight problem instances. The latter can be expressed in different terms including acyclicity checking. Pseudo-Boolean and SAT formulations resulted in a variety of systems, such as LP2ACYCASP+CLASP, LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, and LP2ACYCSAT+GLUCOSE. Systems LP2DIFFZ3 and LP2DIFF+YICES utilize satisfiability modulo theory solvers (Nieuwenhuis, Oliveras, and Tinelli 2006) via a translation from logic programs to difference logic. Among other alternatives, solver LP2MIP relies on a translation into a Mixed Integer Programming problem, and runs CPLEX as back-end, while LP2NORMAL+CLASP normalizes aggregates (of small to medium size) and uses CLASP as back-end ASP solver.

ASP systems have been also extended to exploit multi-core/multi-processor machines by introducing parallel evaluation methods. In particular, parallel techniques for the instantiation of programs were proposed as extensions of the LPARSE (Pontelli, Balduccini, and Bermudez 2003) and DLV (Perri, Ricca, and Sirianni 2013) grounders. Recent approaches for extending the algorithm of CLASP include that of Gebser, Kaufmann, and Schaub (2012b).

Automated algorithm selection techniques have been employed in ASP for obtaining solvers performing well across a wide heterogeneous set of inputs. The idea is to leverage on a number of efficient implementations (or heuristically-different variants of these) and applying machine learning techniques for learning from a training set how to choose the "best" solver for an input program. System CLASPFOLIO (Gebser et al. 2011) combines variants of CLASP, and is a representative of portfolio solving in ASP. System MEASP (Maratea, Pulina, and Ricca 2014), instead, implements a multi-engine portfolio ASP solver, by combining several solvers. The adoption of the ASP-CORE-2 standard input language (Calimeri et al. 2013) allowed the application of algorithm selection techniques also to the grounding step.

Constraint answer set programming is a recent direction of research that attempts to combine advances in answer set programming with these in constraint processing. This new area has already demonstrated promising results, including the development of the solvers ACSOLVER, CLINGCON, EZCSP, IDP, and MINGO. Lierler (2014) surveys the key features of constraint answer set programming languages and systems. This direction of research is inspired by the advances in the related field of satisfiability modulo theories.

Engineering Environments

The availability of efficient solvers makes ASP a valuable tool for many computationally-intensive real-world applications. Effective large-scale software engineering requires infrastructure that includes advanced editors, debuggers, etc.

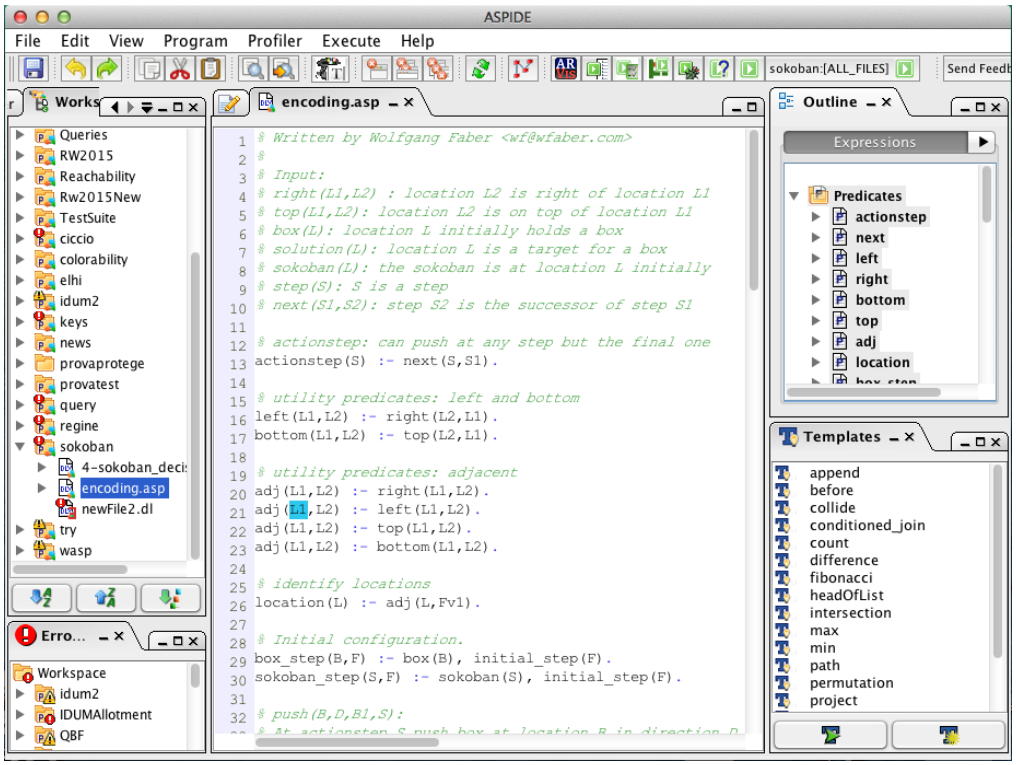

Figure 1: The user interface of *ASPIDE*.

These tools are usually collected in Integrated Development Environments (IDE) that ease the accomplishment of various programming tasks by both novice and skilled software developers. The development of Application Programming Interfaces (API) is also essential for allowing ASP-based solutions within large software frameworks common in modern highly technological world. The following subsections present an overview of both the IDEs for writing ASP programs, and available APIs for building full-fledged software systems based on ASP.

Development Environments for ASP

Several tools have been proposed in the last few years that aim at solving specific tasks arising during the development of ASP programs, including specialized editors, debuggers, testing tools, and visual programming tools. The IDEs that collect several tools in the same framework are also now available. SEA LION (Busoniu et al. 2013) is the first environment offering debugging for programs with variables. It also features unique tools for model-based engineering (using ER diagrams to model domains of answer set programs), testing via annotations, and bi-directional visualization of interpretations. The *ASPIDE* IDE (Febbraro, Reale, and Ricca 2011) is a comprehensive framework that integrates several tools for advanced program composition and execution. To provide an overview of insides of ASP IDEs we briefly outline key features of *ASPIDE*.

A snapshot of the user interface of *ASPIDE* is reported in Figure 1. Logic programs are organized in projects collected in a workspace (displayed in the left panel in Figure 1). The main editor for ASP programs (central frame in Figure 1) offers code line numbering, find/replace, undo/redo, copy/paste, coloring of keywords, dynamic highlighting of predicate names, variables, strings, and comments. The editor is able to complete (on request) predicate names (learned while reading from the files belonging to the same project), as well as variable names (suggested by taking into account the rule one is currently writing). Programs can be modified in an assisted way, for instance, by considering bindings of variables, or by applying custom re-writings (that can be user-defined). Syntax errors and some syntactic conditions (for instance, safety) are checked while writing and promptly outlined. *ASPIDE* suggests quick fixes that can be applied (on request) by automatically changing the affected part of code. Common programming patterns (such as guessing with disjunctive rules, and specific constraints) are available as *code templates* that are expanded as rules (again on request). An outline view (left frame in Figure 1) graphically represents program elements for quick access to the corresponding definition. Users accustomed to graphic programming environments can draw logic programs by exploiting a QBE-like tool for building logic rules (Febbraro, Reale, and Ricca 2010). The user can switch from the text editor to the visual one (and vice versa) thanks to a reverse-engineering mechanism from text to graphical format. The execution of ASP programs is fully customizable trough a number of shortcuts, including toolbar buttons, and drop down menus for a quick execution of files. The results are

presented to the user in a view combining tabular representation of predicates and a tree-like representation of answer sets. *ASPIDE* supports test-driven software development in the style of JUnit (see more details in (Febbraro et al. 2011)).

Program development is enhanced in *ASPIDE* by additional tools such as: the *dependency graph visualizer*, designed to inspect predicate dependencies and browsing the program; the *debugger* to find bugs (Dodaro et al. 2015) the DLV profiler, the ARVis comparator of answer sets, the answer set visualizer IDPDraw, and the *data source plugin* that simplifies the connection to external DBMSs via JDBC. Notably, *ASPIDE* is an extensible environment that can be extended by users providing new plugins supporting (i) new input formats, (ii) new program rewritings, and even (iii) customizing the visualization/output format of solver results.[1]

Building Full-fledged Applications with ASP

IDEs for ASP provide clear advantages for logic programmers, but are not enough to enable assisted development of full-fledged industry-level applications (Grasso et al. 2011; Ricca et al. 2012). ASP is not a full general-purpose language. Thus, ASP programs are eventually embedded in software components developed in imperative/object-oriented programming languages.

The development of APIs, which offer methods for interacting with an ASP system from an embedding program, is a necessary step in accommodating the use of ASP-based solutions within large software systems. Among the first proposals was the DLV Wrapper, a library that allows to embed ASP programs and control the execution of the DLV system from a Java program, and the ONTODLV API, a richer API that allows to embed ontologies and reasoning modules developed using the ONTODLP language (Ricca et al. 2009). More recently, the Potassco group from the University of Potsdam supported the embedding of ASP in Python and Lua programs using GRINGO and CLASP. These interfaces provide a finer grained access to grounder and solver functionality, and also allow incremental solving.

In APIs, however, the burden of the integration between ASP and Java is still in the hands of the programmer, who must take care of the (often repetitive and) time-consuming development of scaffolding code that executes the ASP system and gets data back and forth from logic-based to imperative representations.

These observations inspired the development of a hybrid language, called $\mathcal{JASP}$ (Febbraro et al. 2012), that transparently supports a bilateral interaction between ASP and Java. $\mathcal{JASP}$ introduces minimal syntax extensions both to Java and ASP. Its specifications are both easy to learn by programmers and easy to integrate with other existing Java technologies. The programmer can simply embed ASP code in a Java program without caring about the interaction with the underlying ASP system. An "ASP program" can access Java variables, and the answer sets, resulting from the execution of the ASP code, are automatically stored in Java objects, possibly populating Java collections, in a transparent way. A distinctive feature of $\mathcal{JASP}$ is the clean separation between the two integrated programming paradigms interacting through a standard Object Relational Mapping (ORM) interface. $\mathcal{JASP}$ supports both (i) a default mapping strategy, which fits the most common programmers' requirements, and (ii) custom ORM strategies, which can be specified according to the Java Persistence API (JPA) to perfectly suit enterprise application development standards. The framework also encompasses an implementation of $\mathcal{JASP}$ as a plug-in for the Eclipse platform, called JDLV.

Another hybrid language combining Java and ASP was proposed by Oetsch, Pührer, and Tompits (2011), which employs a radically different strategy for the interaction with Java. For instance, Java methods including constructors can be called by exploiting special atoms in ASP rules.

The ASP Competition Series

ASP Competitions are the events of the ASP community, where ASP solvers are evaluated for efficiency. Since 2007, they take place biennially and are affiliated with the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR); the exception was the fifth event that took place in 2014, affiliated with the 30th International Conference on Logic Programming (ICLP).

In this section we present the history and evolution, in terms of format, solvers participation, and winners, of the ASP Competition series, by first summarizing the editions up to the fourth edition, and then focusing on the 2014 and 2015 events.

ASP Competitions up to 2013

In September 2002, participants of the Dagstuhl Seminar on Nonmonotonic Reasoning, Answer Set Programming and Constraints (Brewka et al. 2002) agreed that standardization was one of the key issues for the development of ASP. This led to the the initiative to establish an infrastructure for benchmarking ASP solvers, as already in use in related research fields of SAT and Constraint Programming. The first informal competition took place during that workshop in Dagstuhl, featuring five systems, namely DLV, SMODELS, ASSAT, CMODELS, and ASPPS. The second informal edition took place in 2005, during another Dagstuhl meeting. Since then, the ASP Competitions have been established as reference events for the community.

The First ASP Competition (Gebser et al. 2007) was organized in Potsdam with two main goals. The first goal was to collect benchmarks. It was achieved through a *call for benchmarks* to members of the community. The second goal was to set up a fair competition environment. In the competition, only decision problems were considered. There were three categories of benchmarks involved:

- MGS (Modeling, Grounding, Solving), where benchmarks were specified by a problem statement, a set of instances, and the names of the predicates and their arguments to be used by programmers to encode solutions;

[1] *ASPIDE* is written in Java and is available for all the major operating systems, including Linux, Mac OS and Windows from `http://www.mat.unical.it/ricca/aspide`.

- SCORE (Solver, Core language), where benchmarks consisted of ground normal and disjunctive programs in the format common to DLV and LPARSE; and
- SLPARSE (Solver, Lparse language), where benchmarks consisted of ground programs in LPARSE format with aggregates.

Ten ASP solvers participated, with several new solvers compared to the first informal events, namely ASPER, CLASP, NOMORE, GNT, LP2SAT, and PBMODELS, thus establishing the advent of CDCL solvers, and solvers based on eager translation-based approaches to ASP solving. Solvers were ranked in terms of number of solved instances: DLV won the MGS and SCORE categories, while CLASP was the best solver on the SLPARSE category.

The Second ASP Competition (Denecker et al. 2009) was organized by K.U. Leuven. Differently from the precursor event, it was a *Model&Solve* team competition: A number of well-specified benchmarks (collected, again, through a *call for benchmarks*, and divided into categories based on complexity) had to be modeled by the participant teams and solved with a system of their choice. Moreover, optimization problems were introduced in the runnings. Sixteen solvers entered the competition: among others, IDP and approaches based on compilation into SMT participated for the first time. The POTASSCO team (Potsdam University) won the overall competition, and performed best on both decision and optimization problems.

In the 2011 and 2013 editions, the format consisted of two different tracks: a Model&Solve and a System track. The former was the continuation of the Second ASP Competition tradition, while the later was in spirit of the First ASP Competition that aimed at fostering language standardization and at allowing participants compete on given encodings under fixed conditions. Both tracks featured a selected suite of domains, chosen again by means of an open *call for benchmarks* stage, and organized in classes based on complexity.

The Third ASP Competition (Calimeri, Ianni, and Ricca 2014) was organized by the University of Calabria. Eleven systems participated in the System track, among them the first portfolio answer set solver CLASPFOLIO, and a number of translation-based solvers. Six teams entered the Model&Solve track, including the FAST DOWNWARD team from the planning community. Winners were determined with a scoring computed by the number of solved instances and the CPU time, plus the *quality* of the solution in case of optimization problems: CLASPD won the System track, while the POTASSCO team won the Model&Solve track. The portfolio solver CLASPFOLIO was the best system on the NP class, that included NP-complete problems and any problem in NP not known to be polynomially solvable.

The Fourth ASP Competition (Alviano et al. 2013) was jointly organized by TU Vienna and the University of Calabria. The design of the event was similar to the previous edition, with some important changes. The competition introduced the standard input language ASP-CORE-2.0 (Calimeri et al. 2013) for the System track (an evolution of the ASP-CORE language proposed in 2011); exceptions were made and problem encodings in legacy formats were still admitted. Also a System track for parallel systems was introduced. Sixteen solvers entered the System track: most of these solvers participated in the earlier editions, with the notable exception of the WASP solver. Seven teams entered the Model&Solve track. About the results: CLASPD and its parallel version CLASPD-MT (Potsdam University) won the System track, while the POTASSCO team was the winner of the Model&Solve track.

The Fifth and Sixth ASP Competitions

The Fifth and the Sixth editions of the ASP Competition series introduced significant modifications to the trend. We first outline the main changes, and then we speak of the two events separately.

The Model&Solve track was no longer an integral part of the events. Rather, it was organized as an (informal) on-site event. The reasons for this choice follow. First, it requires a substantial amount of work to organize and, even more, to participate in such a track. In addition, the participation from neighboring research communities was rather limited, probably due to the presence of competitions in the related research communities, and the non-negligible effort of participating. The first on-site event, called ASP Modeling Competition 2014, saw five participating teams. Each team was formed by three researchers and was allocated a fixed amount of time for solving few problems.

The Fifth Answer Set Programming Competition (Calimeri et al. 2015) broke the usual timeline of the competition series, in order to join the Olympic Games at the Vienna Summer of Logic, in affiliation with the 30th International Conference on Logic Programming (ICLP). It was jointly organized by the Aalto University, the University of Calabria, and the University of Genoa. This event did not feature a *call for benchmarks* and mostly relied on 2013 benchmarks. It was mainly conceived as a re-run of the System track of the previous event: participants to the 2013 event were invited to submit new versions of their solvers but also new solvers were welcome. Several significant design changes and improvements in the competition settings were introduced, i.e., (i) benchmark classes (called *tracks* in this edition) were defined based on the presence of language constructs (e.g., aggregates, choice rules, presence of queries) in problems encoding rather than on a complexity basis, in order to both push the adoption of the new standard, and allow participation also to solvers which may have not included all constructs, (ii) novel encodings for almost all problems were proposed, to overcome some observed limitations of 2013 encodings, and (iii) a simplified scoring schema for decision problems, based on solved instances only, and a scoring schema for optimization problems solely based on the solvers's ranking on solution quality, were employed. Sixteen solvers entered the competition. Answer set solver CLASP was the winner on the single-processor category, while its multi-threaded version CLASP-MT won the multi-threaded category. Interestingly, the solver LP2NORMAL+CLASP, which normalizes aggregates and then resorts to CLASP, was the best solver in an intermediate track, allowing for the full ASP-CORE-2 lan-

guage, except optimization statements and non-Head-Cycle-Free disjunction.

The Sixth ASP Competition (Gebser, Maratea, and Ricca 2015) has been jointly organized by the same institutions of the previous event. Its design maintained some choices of the last event, e.g., tracks based on language features, the scoring schemes, and the adherence to the ASP-CORE-2 standard language. It also presented some novelties, for instance (i) a *call for benchmarks* stage focused on obtaining new benchmarks arising from applications of practical impact, and/or being ASP focused, i.e. whose encodings are non-tight, and (ii) a benchmarks selection stage was introduced to classify instances according to their expected hardness. Moreover, a "marathon" track was added, where the best performing systems are given more time for solving hard instances. Thirteen solvers entered the competition. The winner of the regular track was the multi-engine solver ME-ASP, while the winner of the marathon track was WASP.

Conclusion

Answer set programing is a thriving research field featuring dozens of solvers and applications. Engineering environments for ASP facilitate the adoption of the technology by a broad spectrum of users. Quest for the *ideal* settings of the ASP Competitions attests the ever-changing fast-paste life of the field that strives at advancing answer set programming.

Acknowledgments

This work was partially supported by MIUR under PON project SI-LAB BA2KNOW Business Analitycs to Know, and by Regione Calabria, programme POR Calabria FESR 2007-2013, projects ITravel PLUS and KnowRex: Un sistema per il riconoscimento e lestrazione di conoscenza.

References

Alviano, M.; Calimeri, F.; Charwat, G.; Dao-Tran, M.; Dodaro, C.; Ianni, G.; Krennwallner, T.; Kronegger, M.; Oetsch, J.; Pfandler, A.; Pührer, J.; Redl, C.; Ricca, F.; Schneider, P.; Schwengerer, M.; Spendier, L. K.; Wallner, J. P.; and Xiao, G. 2013. The fourth answer set programming competition: Preliminary report. In *Proc. of LPNMR 2013*, volume 8148 of *LNCS*, 42–53. Springer.

Alviano, M.; Dodaro, C.; Faber, W.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *Proc. of LPNMR 2015*, volume 9345 of *LNCS*, 40–54. Springer.

Brewka, G.; Niemelä, I.; Schaub, T.; and Truszczyński, M. 2002. Dagstuhl Seminar Nr. 0238, Nonmonotonic Reasoning, Answer Set Programming and Constraints, System Competition.

Brochenin, R.; Lierler, Y.; and Maratea, M. 2014. Abstract disjunctive answer set solvers. In *Proc. of ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 165–170. IOS Press.

Busoniu, P.; Oetsch, J.; Pührer, J.; Skocovsky, P.; and Tompits, H. 2013. Sealion: An eclipse-based IDE for answer-set programming with advanced debugging support. *Theory and Practice of Logic Programming* 13(4-5):657–673.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2013. ASP-Core-2 input language format. `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`.

Calimeri, F.; Gebser, M.; Maratea, M.; and Ricca, F. 2015. Design and results of the fifth answer set programming competition. *Artificial Intelligence* in press, available online `http://www.sciencedirect.com/science/article/pii/S0004370215001447`.

Calimeri, F.; Ianni, G.; and Ricca, F. 2014. The third open answer set programming competition. *Theory and Practice of Logic Programming* 14(1):117–135.

Clark, K. L. 1977. Negation as failure. In *Logic and Data Bases*, 293–322.

Denecker, M.; Vennekens, J.; Bond, S.; Gebser, M.; and Truszczynski, M. 2009. The second answer set programming competition. In *Proc. of LPNMR 2009*, volume 5753 of *LNCS*, 637–654. Springer.

Dodaro, C.; Gasteiger, P.; Musitsch, B.; Ricca, F.; and Shchekotykhin, K. 2015. Interactive debugging of non-ground ASP programs. In *Proc. of LPNMR 2015*, volume 9345 of *LNCS*, 279–293. Springer.

Febbraro, O.; Leone, N.; Reale, K.; and Ricca, F. 2011. Unit testing in ASPIDE. In *Proc. of INAP/WLP 2011*, volume 7773 of *LNCS*, 345–364. Springer.

Febbraro, O.; Leone, N.; Grasso, G.; and Ricca, F. 2012. JASP: A framework for integrating answer set programming with Java. In *Proc. of KR 2012*. AAAI Press.

Febbraro, O.; Reale, K.; and Ricca, F. 2010. A visual interface for drawing ASP programs. In *Proc. of CILC 2010*, volume 598 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Febbraro, O.; Reale, K.; and Ricca, F. 2011. ASPIDE: integrated development environment for answer set programming. In *Proc. of LPNMR 2011*, volume 6645 of *LNCS*, 317–330. Springer.

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczynski, M. 2007. The first answer set programming system competition. In *Proc. of LPNMR 2007*, volume 4483 of *LNCS*, 3–17. Springer.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M. T.; and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *Proc. of LPNMR 2011*, volume 6645 of *LNCS*, 352–357. Springer.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012a. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence* 187:52–89.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012b. Multi-threaded ASP solving with clasp. *Theory and Practice of Logic Programming* 12(4-5):525–545.

Gebser, M.; Maratea, M.; and Ricca, F. 2015. The design of the sixth answer set programming competition – report –. In *Proc. of LPNMR 2015*, volume 9345 of *LNCS*. Springer.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In *Proc. of LPNMR 2007*, volume 4483 of *LNCS*, 266–271. Springer.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of ICLP 1988*, 1070–1080. MIT Press.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning* 36(4):345–377.

Grasso, G.; Leone, N.; Manna, M.; and Ricca, F. 2011. ASP at work: Spin-off and applications of the DLV system. In *LP-KR-NR Essays Dedicated to Michael Gelfond*, volume 6565 of *LNCS*, 432–451. Springer.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics* 16(1-2):35–86.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Lierler, Y., and Truszczynski, M. 2011. Transition systems for model generators - A unifying approach. *Theory and Practice of Logic Programming* 11(4-5):629–646.

Lierler, Y. 2014. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence* 207:1–22.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Maratea, M.; Ricca, F.; Faber, W.; and Leone, N. 2008. Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *Journal of Algorithms* 63(1-3):70–89.

Maratea, M.; Pulina, L.; and Ricca, F. 2014. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming* 14(6):841–868.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to DPLL(T). *Journal of the ACM* 53(6):937–977.

Oetsch, J.; Pührer, J.; and Tompits, H. 2011. Extending object-oriented languages by declarative specifications of complex objects using answer-set programming. *CoRR* abs/1112.0922.

Perri, S.; Ricca, F.; and Sirianni, M. 2013. Parallel instantiation of ASP programs: techniques and experiments. *Thoery and Practice of Logic Programming* 13(2):253–278.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Nonmonotonic reasoning on beowulf platforms. In *Proc. of PADL 2003*, volume 2562 of *LNCS*, 37–57. Springer.

Ricca, F.; Gallucci, L.; Schindlauer, R.; Dell'Armi, T.; Grasso, G.; and Leone, N. 2009. OntoDLV: An ASP-based system for enterprise ontologies. *Journal of Logic and Computation* 19(4):643–670.

Ricca, F.; Grasso, G.; Alviano, M.; Manna, M.; Lio, V.; Iiritano, S.; and Leone, N. 2012. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practive of Logic Programming* 12(3):361–381.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Syrjänen, T. 2001. Omega-restricted logic programs. In *Proc. of LPNMR 2009*, volume 2173 of *LNCS*, 267–279. Springer.

Ward, J., and Schlipf, J. S. 2004. Answer set programming with clause learning. In *Proc. of LPNMR 2004*, volume 2923 of *LNCS*, 302–313. Springer.

Wittocx, J.; Mariën, M.; and Denecker, M. 2008. The IDP system: a model expansion system for an extension of classical logic. In *Proc. of LASH 2008*, 153–165.