Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

What's Hot in the SAT and ASP Competitions

Marijn J. H. Heule*
The University of Texas at Austin, United States
marijn@cs.utexas.edu

Torsten Schaub†
University of Potsdam, Germany
torsten@cs.uni-potsdam.de

Abstract

During the Vienna Summer of Logic, the first FLoC Olympic Games were organized, bringing together a dozen competitions related to logic. Here we present the highlights of the Satisfiability (SAT) and Answer Set Programming (ASP) competitions.

Introduction

The SAT Competitions, organized since 2002, have been the driving force of SAT solver development. The performance of contemporary SAT solvers is incomparable to those of a decade ago. As a consequence, SAT solvers are used as the core search engine in many utilities, including tools for hardware and software verification. In 2014, the SAT Competition was organized for the ninth time. As in earlier editions, SAT Competition 2014 consisted of three categories: industrial, hard-combinatorial, and random benchmarks. Similarly, the ASP Competition has a great impact on system development in ASP. Unlike the SAT Competition, its emphasis lies on modeling and tracks reflecting different language fragments with an increasing level of expressiveness.

What's Hot in SAT

The SAT Competition 2014 (SC14) was larger than any of its earlier editions: 79 participants from 14 countries submitting a total of 137 solvers. The number of participating solvers was restricted to 70 by limiting the number of solvers per (co-)author. A large timeout of 5,000 seconds was used for each solver/benchmark pair. SC14 consumed 400,000 hours of CPU time, which were executed in only five days on the Lonestar cluster at Texas Advanced Computing Center (TACC) at The University of Texas at Austin.

The big winner of SC14 was Armin Biere, whose tools *lingeling* (sequential) and *plingeling* (parallel) (Biere 2014) solved most industrial benchmarks and also performed well on hard-combinatorial instances. The crucial difference between *lingeling* and most other top solvers is *inprocessing* (Järvisalo, Heule, and Biere 2012): applying preprocessing techniques during search. Roughly speaking, *lingeling* and *plingeling* alternate each second between a conflict-driven clause learning (CDCL) engine and inprocessing.

Since SAT Competition 2013, it is mandatory for solvers participating in the unsatisfiability tracks to emit refutation proofs to ensure that the results are correct. A new proof format for refutations, called DRAT, introduced this year supports expressing all techniques used in state-of-the-art solvers compactly. Some solvers, such as *lingeling*, use techniques that cannot be expressed using resolution and cannot be expressed in the SAT Competition 2013 formats. One technique that cannot be expressed using resolution, but is used in some top solvers, is *bounded variable addition* (Manthey, Heule, and Biere 2013). The DRAT format facilitates expressing this (and other techniques) compactly, and DRAT proofs were efficiently checked during SC14 using the DRAT-trim tool (Wetzler, Heule, and Hunt 2014).

The results on random benchmarks were exciting this year as the (single core) solver *dimetheus* (Gableske 2014a) by Oliver Gableske outperformed all other solvers, even parallel ones (which ran on twelve cores), on random benchmarks with a huge margin. *dimetheus* consists of two engines, one based on Message Passing (MP) (Gableske 2014b) and one based on Stochastic Local Search (SLS) (Balint and Schöning 2012). The MP engine is used to initialize the SLS engine with an "almost satisfying" assignment. None of the other solvers that participated in SC14 used an MP engine, but the success of *dimetheus* might change that in future competitions.

There are several challenges regarding SAT solvers. First, just a few years ago, parallel versions of top-tier SAT solvers frequently performed worse than the sequential versions (measured in wall-clock time). The last two years, parallel SAT solver performance clearly improved, the gains on a 12-core machine are still modest. This holds for both CDCL and SLS solvers. Second, the unsatisfiability results of parallel SAT solvers cannot be validated efficiently, in contrast to checking the results of sequential SAT solvers. Bridging this gap would increase the trust in the results of parallel SAT solvers. Third, although we witnessed a huge performance boost of a local search solver on random benchmarks this year, SLS solvers still perform poorly on most industrial benchmarks. Achieving similar speed-ups by SLS solvers on industrial benchmarks will be a big challenge.

*Supported by DARPA contract number N66001-10-2-4087.

†Affiliated with Inria Rennes, France, and Simon Fraser University, Canada.

What's Hot in ASP

ASP is a multi-faceted Boolean constraint solving paradigm due to its strong roots in Knowledge Representation and Reasoning. First of all, ASP comes with a rich yet simple first-order modeling language that features object variables, classical and default negation, cardinality and weight constraints, (multi-criteria) optimization directives, and notably recursion. As a consequence, the ASP solving process consists of two phases, an initial grounding phase in which first-order problem representations are translated into a propositional format, followed by the actual solving phase, computing the (stable) models of the obtained propositional representation. While modern ASP grounders are Turing complete, since they allow for dealing with function symbols in an unrestricted way, ASP solvers allow for solving problems at the second level of the polynomial hierarchy (when optimizing problems even go up to Δ_3^P). The high expressiveness is motivated by modeling needs, where deterministic parts are (implicitly) addressed by the grounder, and the actual search is left to the solver.

This richness is also reflected by the series of international ASP Competitions that was started at two Dagstuhl meetings on ASP in 2002 and 2005 (Borchert et al. 2004) and led in 2007 to the official competition series (Gebser et al. 2007). Usually, each competition features two principal tracks: a model-and-solve as well as a systems track. The latter is similar to the approach pursued in SAT but problems are not expressed in a machine readable format but in a human readable one; problem instances are then processed by both a grounder and a solver. With the competition in 2013, the ASP community adopted a common language standard (Calimeri et al. 2012). This standard is supported by the grounders of the *gringo* 4 (and *clingo* 4) series, and (soon) the grounder of *dlv* (dlv). Unlike the systems track, the language of the model-and-solve track is left open in order to welcome non ASP systems as well. For instance, apart from genuine ASP systems, model expansion, Prolog, as well as planning systems participated in previous ASP competitions. The model-and-solve competition then consists in solving problems given in natural language along with some fixed instance format. At the occasion of this year's Vienna Summer of Logic an online variant was added (following the customary Prolog competitions hosted by the ICLP conference series). Teams of one to three members, using one computer, had 90 minutes to model and solve five problems. This competition was won by Mario Alviano, Carmine Dodaro, and Wolfgang Faber.

The ASP competitions are usually held biannually in odd years. The extraordinary competition during the Vienna Summer of Logic resulted in a reduced field of 16 participating ASP systems featuring only the above mentioned online variant along with the usual systems track. The competition consisted of two categories, allocating one or multiple processors to each system, respectively. To reflect the variety of ASP, each category was structured in four tracks:

Basic Decision Encodings: normal logic programs, simple arithmetic and comparison operators.

Advanced Decision Encodings: full language with queries, excepting optimization statements and non-HCF disjunction.[1]

Optimization Encodings: full language with optimization statements, excepting non-HCF disjunction.

Unrestricted Encodings: full language.

The language's expressiveness along with its associated complexity is meant to increase over the tracks. All but one track were won by the native ASP solver *clasp* (potassco); the (single processor) Advanced Decision track was won by the translation-based solver *lp2normal2+clasp* (lp2normal) using *clasp* with a dedicated preprocessor.

References

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *SAT 2012*, 16–29. Springer.

Biere, A. 2014. Yet another local search solver and lingeling and friends entering the sat competition. In *Proceedings of SAT competition 2014: Solver and Benchmark Descriptions*. 39–40.

Borchert, P.; Anger, C.; Schaub, T.; and Truszczyński, M. 2004. Towards systematic benchmarking in answer set programming: The Dagstuhl initiative. In *LPNMR 2004*, 3–7. Springer.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2012. ASP-Core-2: Input language format. Available at https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf.

dlv. http://www.dlvsystem.com.

Gableske, O. 2014a. Dimetheus. In *Proceedings of SAT competition 2014: Solver and Benchmark Descriptions*. 29–30.

Gableske, O. 2014b. An ising model inspired extension of the product-based MP framework for SAT. In *SAT 2014*, 367–383. Springer.

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

Järvisalo, M.; Heule, M. J. H.; and Biere, A. 2012. Inprocessing rules. In *Automated Reasoning*, 355–370. Springer.

lp2normal. http://research.ics.aalto.fi/software/asp/lp2normal.

Manthey, N.; Heule, M. J.; and Biere, A. 2013. Automated reencoding of Boolean formulas. In *Hardware and Software: Verification and Testing*, 102–117. Springer.

Potassco. http://potassco.sourceforge.net.

wasp. http://www.mat.unical.it/ricca/wasp.

Wetzler, N.; Heule, M. J. H.; and Hunt Jr., W. A. 2014. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT 2014*, 422–429. Springer.

[1] HCF stands for head-cycle-free; non-HCF disjunctive programs have an elevated complexity.