Constraint Answer Set Programming*

Yuliya Lierler
University of Nebraska at Omaha

Introduction

Constraint answer set programming (CASP) is a novel, promising direction of research whose roots go back to propositional satisfiability (SAT). SAT solvers are efficient tools for solving boolean constraint satisfaction problems that arise in different areas of computer science, including software and hardware verification. Some constraints are more naturally expressed by non-boolean constructs. Satisfiability modulo theories (SMT) extends boolean satisfiability by the integration of non-boolean symbols defined by a background theory in another formalism, such as a constraint processing language. Answer set programming (ASP) extends computational methods of SAT in yet another way, inspired by ideas from knowledge representation, logic programming, and nonmonotonic reasoning. As a declarative programming paradigm, it provides a rich, simple modeling language that, among other features, incorporates recursive definitions. Answer set programming languages also use variables; software tools called grounders are used as front ends of answer set solvers to eliminate variables, whereas SAT-like procedures form their back-ends.

Constraint answer set programming draws on both of these extensions of SAT technology: it integrates answer set programming with constraint processing. This new area has already demonstrated promising results, including the development of the CASP solvers ACSOLVER [11] (Texas Tech University), CLINGCON[1] [7] (Potsdam University, Germany), EZCSP[2] [1] (KODAK), IDP[3] [15] (KU Leuven), MINGO[4] [10] (Aalto University, Finland). CASP is a new, powerful paradigm for declarative programming that provides new modeling features for answer set programming and also improves grounding and solving performance by delegating processing of constraints over large and possibly infinite domains to specialized systems. As a result CASP opens new horizons for declarative programming applications. The origins of this work go back to [2].

*We are greatful to Marcello Balduccini, Broes de Cat, Vladimir Lifschitz, and Peter Schüller for valuable comments.

[1] http://www.cs.uni-potsdam.de/clingcon/

[2] http://marcy.cjb.net/ezcsp/index.html

[3] http://dtai.cs.kuleuven.be/krr/software/idp

[4] http://research.ics.aalto.fi/software/asp/

Related Work: HEX-Programs [3] integrate logic programs under answer set semantics with external computation sources via *external atoms*. They were motivated by the need to interface ASP with external computation sources, for example, to allow the synergy of ASP and description logic computations within the context of the semantic web. CASP shares a lot in common with HEX-programs. System DLVHEX[5] [4] computes models of such programs. It allows defining plug-ins for inference on external atoms and as such can be used as a general framework for developing CASP solvers (but it does not provide any specific computational mechanism by default).

1 Logic Programs with Constraint Atoms

A *regular program* is a finite set of rules of the form

$$\begin{aligned} a_0 \leftarrow a_1, \ldots, a_l, &not\ a_{l+1}, \ldots, not\ a_m, \\ &not\ not\ a_{m+1}, \ldots, not\ not\ a_n, \end{aligned} \tag{1}$$

where a_0 is $\bot$ or an atom, and each a_i $(1 \leq i \leq n)$ is an atom. This is a special case of programs with nested expressions [9]. We refer the reader to the paper by Lifschitz et al.[9] for details on the definition of an answer set of a logic program. A *choice rule* construct $\{a\}$ [12] of the LPARSE language can be seen as an abbreviation for a rule $a \leftarrow not\ not\ a$ [5]. We adopt this abbreviation in the rest of the paper.

A constraint satisfaction problem (CSP) is defined as a triple $\langle X, D, C\rangle$, where X is a set of variables, D is a domain of values, and C is a set of constraints. Every constraint is a pair $\langle t, R\rangle$, where t is an n-tuple of variables and R is an n-ary relation on D. An *evaluation* of the variables is a function from the set of variables to the domain of values, $\nu : X \to D$. An evaluation ν *satisfies* a constraint $\langle (x_1, \ldots, x_n), R\rangle$ if $(v(x_1), \ldots, v(x_n)) \in R$. A *solution* is an evaluation that satisfies all constraints.

Consider an alphabet consisting of regular and constraint atoms, denoted by $\mathcal{A}$ and $\mathcal{C}$ respectively. By $\tilde{\mathcal{C}}$, we denote the set of all literals over $\mathcal{C}$. The constraint literals are identified with constraints via a function $\gamma : \tilde{\mathcal{C}} \to C$ so that for any literal l, $gamma(l)$ has a solution if and only if $gamma(\bar{l})$ does not have one (where $\bar{l}$ denotes a complement of l). For a set Y of constraint literals over $\mathcal{C}$, by $\gamma(Y)$ we denote a set of corresponding constraints, i.e., $\{\gamma(c) | c \in Y\}$. Furthermore, each variable in $\gamma(\tilde{\mathcal{C}})$ is associated with a domain. For a set M of literals, by M^+ and $M^{\mathcal{C}}$ we denote the set of positive literals in M and the set of constraint literals over $\mathcal{C}$ in M, respectively.

A *logic program with constraint atoms* is a regular logic program over an extended alphabet $\mathcal{A} \cup \mathcal{C}$ such that a_0 is $\bot$ or $a_0 \in \mathcal{A}$. Given a logic program with constraint atoms Π, by $\Pi^{\mathcal{C}}$ we denote Π extended with choice rules $\{c\}$ for each constraint atom c occurring in Π. We say that a consistent and complete set M of literals over atoms of Π is an answer set of Π if (i) M^+ is an answer set of $\Pi^{\mathcal{C}}$ and (ii) $\gamma(M^{\mathcal{C}})$ has a solution.

[5] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

For example, let Π be the program

$$\begin{array}{l} am \leftarrow \ X < 12 \\ lightOn \leftarrow \ switch, not\ am \\ \{switch\} \\ \bot \leftarrow not\ lightOn. \end{array} \tag{2}$$

Intuitively, this program states that (a) *light* is *on* only if an action of *switch* occurs during the *pm* hours and (b) *light* is *on* (according to the last rule in the program). Consider a domain of X to be integers from 0 till 24. It is easy to see that a set

$$\{switch, lightOn, \neg\ am, \neg X < 12\}$$

forms the only answer set of program (2).

2 On the relation of constraint answer set languages and solvers

In the introduction, we listed a number of CASP systems ACSOLVER, CLINGCON, EZCSP, IDP, MINGO that were developed in recent years. The core of the languages of these systems is captured by the definition of a logic program with constraint atoms presented here. Yet, relating these languages is not an easy task. One difficulty lies in the fact that these languages are introduced together with specific system archetecture in mind that rely on various ASP/CSP/CLP/SMT technology. The syntactic differences that stam from technological differences stand on the way of clear understanding of key features of the languages. Relating CASP systems formally is even more complex task. The variations in underlying technologies complicate clear articulation of their similarities and differences. For example, the CASP solver CLINGCON [7] is developed using an ASP solver CLASP [6] and a constraint solver GECODE [14]. The main building blocks of the CASP solver ACSOLVER [11] are the ASP system SMODELS [12] and SICSTUS PROLOG[6]. In addition, the CASP solvers adopt different communication schemes among their heterogeneous solving components. For instance, the system EZCSP relies on *blackbox* integration of ASP and CSP tools in order to compute the answer sets of an EZCSP program [1]. Systems ACSOLVER, CLINGCON, IDP promote tighter integration of multiple automated reasoning methods. The broad attention to CASP paradigms suggests a need for a principled and general study of methods to develop unifying terminology and formalisms suitable to capture variants of the languages and solvers. The work by Lierler [8] can be seen as a step in this direction. It presents a formal account that illustrates a precise relationship between the languages of ACSOLVER and CLINGCON as well as between the respective systems. Usually backtrack search procedures (Davis-Putnam-Logemann-Loveland (DPLL)-like procedures) that form a backbone of CASP computational methods are described in terms of pseudocode.

[6] `http://www.sics.se/isl/sicstuswww/site/index.html`

In [13], the authors proposed an alternative approach to describing DPLL-like algorithms. They introduced an abstract framework that captures what states of computation are, and what transitions between states are allowed. In this way, it defines a directed graph such that every execution of DPLL corresponds to a path in this graph. Some edges may correspond to a propagation steps, some to branching, some to backtracking. This approach allows us to model a DPLL-like algorithm by a mathematically simple and elegant object, graph, rather than a collection of pseudocode statements. An abstract framework of the sort served as the main tool for performing precise formal analyses relating such constraint answer set solvers as ACSOLVER and CLINGCON [8].

References

[1] Marcello Balduccini. Representing constraint satisfaction problems in answer set programming. In *Proceedings of ICLP'09 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'09)*, 2009.

[2] Sabrina Baselice, Piero A. Bonatti, and Michael Gelfond. Towards an integration of answer set and constraint solving. In Maurizio Gabbrielli and Gopal Gupta, editors, *ICLP*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2005.

[3] Thomas Eiter, Gerhard Brewka, Minh Dao-Tran, Michael Fink, Giovambattista Ianni, and Thomas Krennwallner. Combining Nonmonotonic Knowledge Bases with External Sources. In Silvio Ghilardi and Roberto Sebastiani, editors, *7th International Symposium on Frontiers of Combining Systems (FroCos 2009)*, volume 5749 of *LNAI*, pages 18–42. Springer, September 2009.

[4] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 90–96, 2005.

[5] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

[6] Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. MIT Press, 2007.

[7] Martin Gebser, Max Ostrowski, and Torsten Schaub. Constraint answer set solving. In *Proceedings of 25th International Conference on Logic Programming (ICLP)*, pages 235–249. Springer, 2009.

[8] Yuliya Lierler. On the relation of constraint answer set programming languages and algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*. MIT Press, 2012.

[9] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[10] Guohua Liu, Tomi Janhunen, and Ilkka Niemelae. Answer set programming via mixed integer programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 13th International Conference*, page 3242, 2012.

[11] Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 2008.

[12] Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.

[13] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 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, 2006.

[14] Christian Schulte and Peter J. Stuckey. Efficient constraint propagation engines. *Transactions on Programming Languages and Systems*, 2008.

[15] Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165, 2008.