October, 2018

SMT-based Constraint Answer Set Solver EZSMT+ for Non-tight Programs

Da Shen, *University of Nebraska at Omaha*
Yuliya Lierler

SMT-based Constraint Answer Set Solver EZSMT$^+$ for Non-tight Programs

Da Shen and Yuliya Lierler
Department of Computer Science, University of Nebraska at Omaha
dashen@unomaha.edu, ylierler@unomaha.edu

Abstract

Constraint answer set programming integrates answer set programming with constraint processing. System EZSMT$^+$ is a constraint answer set programming tool that utilizes satisfiability modulo theory solvers for search. The truly unique feature of EZSMT$^+$ is its capability to process linear as well as nonlinear constraints simultaneously containing integer and real variables.

Introduction

Answer set programming (ASP) is a declarative programming paradigm for solving difficult combinatorial search problems (Brewka, Eiter, and Truszczyński 2011). Constraint answer set programming (CASP) is a recent development, which integrates ASP with constraint processing. Often, this integration allows one to tackle a challenge posed by *grounding bottleneck*. Originally, systems that processed CASP programs relied on combining algorithms/solvers employed in ASP and constraint processing (Gebser, Ostrowski, and Schaub 2009; Balduccini and Lierler 2017). Lee and Meng; Susman and Lierler; Lierler and Susman (2013; 2016; 2017) proposed an alternative approach that utilizes satisfiability modulo theory (SMT) solvers (Barrett and Tinelli 2014) in design of CASP systems.

System EZSMT (Susman and Lierler 2016) is a representative of an SMT-based approach for tackling CASP programs. Yet, it has several limitations. For instance, it is unable to process a large class of logic programs called *non-tight* (Fages 1994). This restriction does not allow users, for example, to express succinctly transitive closure. System EZSMT is also unable to enumerate multiple solutions to a problem. In this work, we extend EZSMT so that the described limitations are eliminated. We call the new system EZSMT$^+$.

Preliminaries

CASP extends the language of logic programs by specialized atoms that are linked to "(primitive) constraints" (Marriott and Stuckey 1998, Section 1.1). Intuitively, to establish whether a set X of atoms forms an answer set – a solution – of a CASP program two steps are required: (1) verifying that X is an answer set of a CASP program understood as a traditional logic program and (2) inspecting whether X has a solution by forming a constraint satisfaction problem based on the occurrence of special atoms in X. We now make these ideas precise.

Logic Programs. A *vocabulary* is a set of propositional symbols also called atoms. A *literal* is an atom a or its negation $\neg a$. A *(propositional) logic program* over vocabulary σ is a set of *rules* of the form

$$a \leftarrow b_1, \ldots, b_\ell,\ not\ b_{\ell+1}, \ldots,\ not\ b_m,\ not\ \ not\ b_{m+1}, \ldots,\ not\ \ not\ b_n \quad (1)$$

where a is an atom in σ or $\bot$, and each b_i, where $1 \leq i \leq n$, is an atom in σ. We will use the abbreviated form for rule (1), i.e., $a \leftarrow B$, where B stands for the right hand side of the arrow in (1) and is also called a *body*. By B^+ we denote the *positive* part of body B, i.e., $b_1, \ldots, b_\ell$. We sometimes identify body B with the propositional formula $b_1 \wedge \ldots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \ldots \wedge \neg b_m \wedge \neg\neg b_{m+1} \wedge \ldots \wedge \neg\neg b_n$, and rule (1) with the propositional formula $B \rightarrow a$. The expression a is the *head* of the rule. A rule whose head is symbol $\bot$ is called a denial. For a logic program Π (a propositional formula F), by $At(\Pi)$ (by $At(F)$) we denote the set of atoms occurring in Π (in F).

For the definition of an *input answer set relative to a vocabulary* we refer to (Lierler and Susman 2017, Definition 3). This concept is used in defining semantics of constraint answer set programs below.

Constraints. Lierler and Susman (2017) illustrated that the notion of a constraint syntactically coincides with ground literals of SMT. Furthermore, a constraint satisfaction problem (CSP) can be identified with the conjunction of ground literals, which is evaluated by means of first-order logic interpretations/structures representative of a particular "uniform" SMT-logic (Lierler and Susman 2017). Uniform SMT-logics are defined via interpretations/structures whose domain, interpretation of predicates, and "interpreted" function symbols are fixed. In practice, special forms of constraints are commonly used. *Integer linear constraints* are examples of these special cases. For instance, $2x + 3y > 0$ is a common abbreviation for an integer linear constraint

$> (+(\times(2, x), \times(3, y)), 0)$, where we assume SMT-logic called *Integer Linear Arithmetic (ILA)*. This logic is defined by interpretations, whose domain is the set of integers, predicate $>$ is interpreted as an arithmetic greater relation/predicate symbol; function symbols $+$ and $\times$ are interpreted as usual in arithmetic; 0-arity function symbols 2, 3, and 0 are interpreted by mapping these into respective domain elements (identified with the same symbol). Constraint $2x + 3y > 0$ contains un-interpreted 0-arity function symbols x and y that are frequently referred to as object constants (in logic literature) or variables (in constraint processing literature). We call an interpretation satisfying a CSP its solutions. We identify this interpretation with a function called *valuation* that provides a mapping for uninterpreted function symbols to domain elements. For example, one of the solutions to CSP composed of a single constraint $2x + 3y > 0$ within ILA-logic is valuation that maps x to 0 and y to 1.

Constraint Answer Set Programs. Let σ_r and σ_i be two disjoint vocabularies. We refer to their elements as *regular* and *irregular* atoms, respectively.

Definition 1. *Let $\sigma = \sigma_r \cup \sigma_i$ be a vocabulary so that σ_r and σ_i are disjoint; $\mathcal{B}$ be a set of constraints; γ be an injective function from the set of irregular literals over σ_i to $\mathcal{B}$.*

We call a triple $P = \langle \Pi, \mathcal{B}, \gamma \rangle$ an EZ program *over vocabulary $\sigma_r \cup \sigma_i$, when Π is a logic program over $\sigma_r \cup \sigma_i$ such that any rule that contains atoms in σ_i is a rule with symbol $\bot$ in its head.*

A set $X \subseteq At(\Pi)$ of atoms is an answer set *of P if*

(a) X is an input answer set of Π relative to σ_i, and

(b) the following CSP has a solution:

$$\{\gamma(a) | a \in X \cap \sigma_i\} \cup \{\gamma(\neg a) | a \in \sigma_i \setminus X\}. \quad (2)$$

A pair $\langle X, \nu \rangle$ is an extended answer set *of P if X is an* answer set *of P and valuation ν is a solution to the CSP constructed in (b).*

We call a triple $\mathcal{F} = \langle F, \mathcal{B}, \gamma \rangle$ an SMT formula *over vocabulary σ, when F is a propositional formula over σ.*

A set $X \subseteq At(F)$ is a model *of SMT formula $\mathcal{F}$ if*

(a) X is a model of F, and

(b) the CSP (2) has a solution.

A pair $\langle X, \nu \rangle$ is an extended model *of $\mathcal{F}$ if X a* model *of $\mathcal{F}$ and ν is a solution to the CSP constructed in (b).*

Lierler and Susman (2017) illustrated that for an EZ program over integer linear arithmetic one can construct an SMT formula so that its models/extended models coincide with the answer sets/extended answer sets of the given program. The generalizations of concepts of completion and level ranking (originally introduced in (Clark 1978) and (Niemela 2008) respectively) are essential in the construction of such an SMT formula. Here we provide the translation presented in (Lierler and Susman 2017). We utilize vertical bars to mark the irregular atoms (introduced within the translation) that have intuitive mappings into respective integer linear constraints. For instance, expression $|lr_b - 1 \geq lr_a|$ introduces an irregular atom that is mapped into $lr_b - 1 \geq lr_a$, where lr_a and lr_b are variables over integers. Let $P = \langle \Pi, \mathcal{B}, \gamma \rangle$ be an EZ program over $\sigma_r \cup \sigma_i$. For every atom a in σ_r that occurs in Π we introduce an integer variable lr_a. The SMT formula $\mathcal{F}^P = \langle F^P, \mathcal{B}^P, \gamma^P \rangle$ is constructed as follows (i) its F^P is a conjunction of the following

- implications corresponding to rules (1) in Π;
- the implications $a \rightarrow \bigvee_{a \leftarrow B \in \Pi} B$ for all atoms a in σ_r;
- for each atom $a \in \sigma_r$ the implication
$$a \rightarrow \bigvee_{a \leftarrow B \in \Pi} (B \wedge \bigwedge_{b \in B^+ \setminus \sigma_i} |lr_a - 1 \geq lr_b|)$$

(ii) its $\mathcal{B}^P$ and γ^P extend $\mathcal{B}$ and γ in intuitive way to accommodate irregular atoms such as $|lr_b - 1 \geq lr_a|$.

Niemela (2008) introduced strong level rankings and also illustrated how strongly connected components of a dependency graph of a normal program can be used to enhance the transformation from a normal program to an SMT formula. Shen and Lierler (2018) generalized these results to logic programs whose rules are of the form (1). They also showed that bounds of an integer variable lr_a can be safely set as 0 and the size of the strongly connected component containing atom a. These ideas are applicable within EZ programs and are utilized in the implementation of EZSMT$^+$.

EZSMT$^+$ Architecture and Experiments

We start by discussing some unique features of the EZSMT$^+$ system. It utilizes SMT solvers as its search backend. Thus, EZSMT$^+$ can use the variety of offered SMT-logics that go beyond ILA. For example, logic AUFLIRA (Tinelli and Barrett 2015) enables us to state linear constraints that may simultaneously contain integer and real variables. Logic AUFNIRA permits nonlinear constraints, too. System EZSMT$^+$ accepts programs written in the fragment of EZ language supported by the solver EZCSP and best documented in (Balduccini and Lierler 2017). We extend this language by several directives that allow us to specify a domain for a constraint variable.

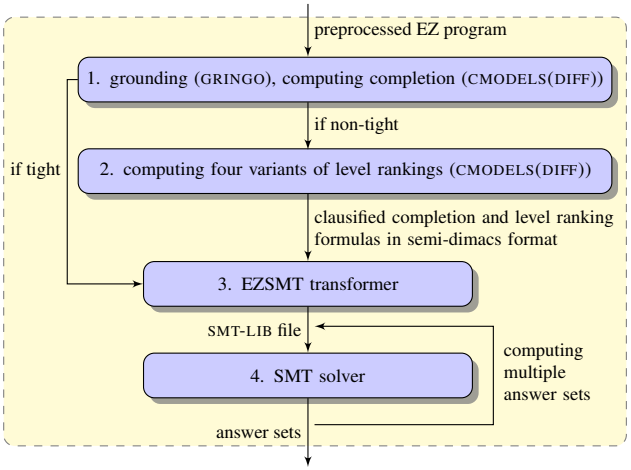

Figure 1: EZSMT$^+$ Architecture

Figure 1 illustrates the alirchitecture of EZSMT$^+$[1]. This system takes an arbitrary (tight or non-tight) CASP program preprocessed by EZCSP (Balduccini and Lierler 2017) as an input. It then utilizes grounder GRINGO (Gebser et al. 2011) for eliminating ASP variables. Routines of system CMODELS(DIFF) (Shen and Lierler 2018) are used to compute completion and level rankings of the program. Then, EZSMT$^+$ system translates the completion augmented with level rankings into SMT formulas, after which an SMT solver is called to find models of these formulas (that correspond to answer sets).

(1) Computing completion; (2) Computing level ranking formulas and clausification The EZSMT$^+$ system utilizes CMODELS(DIFF) (Shen and Lierler 2018) to compute a program's completion. During this step, we also determine whether the program is tight or not. If the program is not tight, the corresponding level ranking formula will be added.

A procedure used by EZSMT$^+$ to perform this task is identical to that of CMODELS(DIFF) (Shen and Lierler 2018). Just as in the case of CMODELS(DIFF), we can instruct EZSMT$^+$ to construct different kinds of level ranking formulas using flags `-levelRanking`, `-levelRankingStrong`, `-SCClevelRanking`, and `-SCClevelRankingStrong`. Default behavior of system EZSMT$^+$ is captured by `-SCClevelRanking`. By default, we set upper bounds for a level ranking variable as the number of atoms inside the strongly connected component containing the atom corresponding to this variable. We provide an option to set a bigger upper bound as the total number of ground atoms when EZSMT$^+$ is called with the flag `-biggerUpperBound`. Finally, the resulting formulas are clausified to produce an output in semi-Dimacs format (Susman and Lierler 2016).

(3) Transformation The semi-Dimacs output from step (2) is transformed into SMT-LIB syntax (a standard input language for SMT solvers (Barrett, Fontaine, and Tinelli 2015)) using an EZSMT procedure described in (Susman and Lierler 2016). We extend the transformer of EZSMT v1.0. to allow the constraint variables be of different domains in the same program, i.e., integers and reals. This feature is invoked when the input program contains the special/directive atom $cspdomain(mixed)$. Special atoms of the form $cspvardomain(*var\_name*, int)$ instruct the transformer to declare a variable whose name is $*var\_name*$ as a variable over integers. By default, the variables are assumed to be over reals.

(4) Solving Finally, one of the SMT solvers CVC4 (Barrett et al. 2011), Z3 (De Moura and Bjørner 2008), or YICES (Dutertre 2017) is called to compute models. In fact, any other SMT solver supporting SMT-LIB can be utilized easily.

[1]https://www.unomaha.edu/college-of-information-science-and-technology/natural-language-processing-and-knowledge-representation-lab/software/ezsmt.php

The EZSMT$^+$ system allows us to compute multiple (extended) answer sets. It utilizes ideas exploited in the implementation of CMODELS(DIFF) (Shen and Lierler 2018, Section 5). In summary, after computing an (extended) answer set X of a program it invokes an SMT solver again by adding formulas encoding the fact that newly computed model should be different from X. This process is repeated until the pre-specified number of solutions is enumerated or it has been established that no more solutions exist.

We benchmark EZSMT$^+$ to compare its performance with the state-of-the-art CASP solvers ingcCLINGCON (Banbara et al. 2017; Ostrowski 2018) and EZCSP. The benchmarks are posted at the EZSMT$^+$ website (see Footnote 1).

Three benchmarks, namely, Reverse Folding (RF), Incremental Scheduling (IS), and Weighted Sequence (WS), come from the Third Answer Set Programming Competition (Calimeri et al. 2011). We include a benchmark called Blending (BL) problem (Biavaschi 2017) and extend it to BL*, which contains variables over both integers and reals. Also, we use Bouncing Ball (BB) domain (Bartholomew 2016). It is important to remark that the encoding for BB domain results in a tight program. This domain uses nonlinear constraints over real numbers. In such a case EZSMT$^+$ is unable to handle nontight programs. Yet, it can process (non-)tight programs with integer nonlinear constraints (utilizing Z3 or YICES) or (non-)tight programs with linear constraints over real or mixed domain (using Z3 or CVC4). Three more benchmarks, namely, RoutingMin (RMin), RoutingMax (RMax), and Travelling Salesperson (TS) are obtained from Liu, Janhunen, and Niemela (2012). The original TS benchmark is an optimization problem, and we turn it into a decision one. The Labyrinth (LB) benchmark is extended from the domain presented in the Fifth Answer Set Programming Competition (Calimeri et al. 2016). This extension allows us to add integer linear constraints into the problem encoding. Next benchmark, Robotics (RB), comes from Young, Balduccini, and Israney (2017). Also, we present results on two benchmarks from Balduccini et al. (2017), namely, Car and Generator (GN).

All benchmarks are run on an Ubuntu 16.04.1 LTS (64-bit) system with an Intel core i5-4250U processor. The resource allocated for each benchmark is limited to one cpu core and 4GB RAM. We set a timeout of 1800 seconds. No problems are solved simultaneously.

Systems that we use to compare the performance of variants of EZSMT$^+$ (invoking SMT solver CVC4 v. 1.4; Z3 v. 4.5.1; YICES v. 2.5.4) are CLINGCON v. 3.3.0 and the variants of EZCSP v. 2.0.0 (invoking ASP solvers CLASP v. 3.2.0 or CMODELS v. 3.86.1; and constraint solver Bprolog v. 7.5, SWIprolog v. 7.4.1, or MINIZINC v. 2.0.2). System GRINGO v. 4.5.3 is used as grounder for EZSMT$^+$ and EZCSP with one exception: GRINGO v. 3.0.5 is utilized for EZCSP for the Reverse Folding benchmark (due to some incompatibility issues).

Figure 2 summarizes main results. In this figure, we

Category	Benchmark	CLINGCON	EZSMT$^+$(Z3)		EZSMT$^+$(YICES)		EZCSP CLASP-SWI	EZCSP CLASP-MZN
			SCC	SCCStrong	SCC	SCCStrong		
NT-IL	RMin (100)	**4.68**	8.76	11.8	5.81	7.57	126007(70)	126002(70)
	RMax (100)	3144	459	**22.4**	5190	5945	180000(100)	180000(100)
	TS (30)	455	7347(4)	43620(24)	1881(1)	75.2	**14.3**	54000(30)
	LB (22)	**3002(1)**	9510(1)	10089(2)	4399(2)	5512(2)	12558(6)	12638(6)
T-IL	RF (50)	326	6058(2)		27840(14)		**101**	7218(4)
	IS (30)	54000(30)	9200(5)		**9098 (5)**		41446(21)	39458(21)
	WS (30)	52.5	29.2		**5.23**		54000(30)	54000(30)
T-INL	Car (8)	/	0.32		**0.25**		10.1	2.34
T-RL	BL (30)	/	88.4		**47.4**		18322(9)	/
	GN (8)	/	0.58		**0.48**		5641(3)	/
	RB (8)	/	0.4		**0.39**		2.04	/
T-RNL	BB (5)	/	3663(2)		**0.98**		9000(5)	/
T-ML	BL* (30)	/	**5573(2)**		/		/	/

Figure 2: Summary of Experimental Data

use EZSMT$^+$(CVC4), EZSMT$^+$(Z3), and EZSMT$^+$(YICES) to denote three variants of EZSMT$^+$. Acronym EZCSP-CLASP-SWI (EZCSP-CLASP-MZN) stands for a variant of EZCSP, where CLASP is utilized as the answer set solver and SWIprolog (MINIZINC, respectively) is utilized as a constraint solver. We note that EZSMT$^+$(CVC4) compares worse or at most compatible with other variants of EZSMT$^+$ except on BL*. For the lack of space, we do not present the results for this configuration. Yet, we make all the experimental data available at the EZSMT$^+$ website (see Footnote 1). Also, variants of EZCSP utilizing CMODELS as an answer set solver and/or Bprolog as a constraint solver are consistently outperformed by EZCSP-CLASP-SWI and EZCSP-CLASP-MZN on our benchmarks. Thus, we do not show these results here.

In Figure 2, we present cumulative time of all instances for each benchmark with numbers of unsolved instances due to timeout or insufficient memory inside parentheses. The "/" sign indicates that this solver or its variant does not support the kinds of constraints occurring in the encoding. For example, CLINGCON does not support constraints over real numbers or nonlinear constraints. Total number of used instances is shown in parentheses after a benchmark name. All the steps involved, including grounding and transformation, are reported as parts of solving time. The benchmarks are divided into categories by double separations. The acronyms T and NT, in the names of the categories, indicate that the programs are tight and non-tight, respectively. The letters I and R indicate that the constraints are over integer and real numbers, respectively. The letter M says that the constraints are over mixed real and integer variables. The acronyms L and NL state that the constraints are linear and nonlinear, respectively.

Systems CLINGCON, EZCSP-CLASP-SWI, and EZCSP-CLASP-MZN are run in their default settings. We benchmarked non-tight programs with the three variants of EZSMT$^+$ using several configurations. For non-tight benchmarks, EZSMT$^+$ invoked with `-levelRanking` and `-levelRankingStrong` flags shows substantially worse performance than settings `-SCClevelRanking` and `-SCClevelRankingStrong`, respectively. Among the latter two configurations, one cannot point at a definite winner. Thus, we present both of them and label them as SCC and SCCStrong, respectively. Choosing minimal upper bounds for level ranking variables tends to help EZSMT$^+$ perform better. We do not show the results for the non-default `-biggerUpperBound` configuration.

In summary, we observe that CLINGCON, EZSMT$^+$(Z3) and EZCSP-CLASP-SWI wins in two benchmarks, respectively, while EZSMT$^+$(YICES) ranks first in seven benchmarks. Variants of the EZSMT$^+$ systems display the best overall results. Utilizing different SMT solvers may improve the performance of EZSMT$^+$ in the future.

The results show that EZSMT$^+$ is a viable tool for finding *answer sets* of CASP programs, and can solve some difficult instances where its peers time out. Also, it provides new capabilities towards utilizing declarative answer set programming paradigm for problems containing a wide variety of constraints including linear constraints over real numbers, mixed integer and real numbers, as well as nonlinear constraints.

Related and Future Work Recent CASP solver CLINGO[LP] (Janhunen et al. 2017) handles linear constraints over integers or reals and permits optimization statements. The experimental analysis presented in (Janhunen et al. 2017) only considers programs with constraints over integers. On these benchmarks, CLINGCON outperforms CLINGO[LP]. The other related CASP systems are solvers MINGO (Liu, Janhunen, and Niemela 2012) and ASPMT2SMT (Bartholomew and Lee 2014). Susman and Lierler (2016) compare the performance of MINGO and EZSMT on tight programs. The later consistently has better performance. The ASPMT2SMT system is a close relative of EZSMT$^+$ in the sense that it utilizes SMT solver z3 for search. Solver ASPMT2SMT is restricted to tight programs. We expect that EZSMT$^+$(Z3) times mimic these of ASPMT2SMT on tight programs.

In the future, we will perform more extensive experimental analysis to compare EZSMT$^+$ to the mentioned CASP systems. We envision an extension of EZSMT$^+$ to allow processing of optimization statements and more sophisticated enumeration of solutions.

Acknowledgements We are grateful to Marcello Balduccini for valuable discussions on the subject of the paper and the EZCSP system. We are also thankful to Ben Susman and Sara Biavaschi. Da Shen was supported by the 2017-FUSE (Fund for Undergraduate Scholarly Experiences) Grant from the University of Nebraska at Omaha. Yuliya Lierler was partially supported by the NSF 1707371 grant.

References

[2017] Balduccini, M., and Lierler, Y. 2017. Constraint answer set solver ezcsp and why integration schemas matter. *Theory and Practice of Logic Programming* 17(4):462–515.

[2017] Balduccini, M.; Magazzeni, D.; Maratea, M.; and Leblanc, E. C. 2017. Casp solutions for planning in hybrid domains. *Theory and Practice of Logic Programming* 17(4):591–633.

[2017] Banbara, M.; Kaufmann, B.; Ostrowski, M.; and Schaub, T. 2017. Clingcon: The next generation. *TPLP* 17(4):408–461.

[2014] Barrett, C., and Tinelli, C. 2014. Satisfiability modulo theories. In Clarke, E.; Henzinger, T.; and Veith, H., eds., *Handbook of Model Checking*. Springer.

[2011] Barrett, C.; Conway, C. L.; Deters, M.; Hadarean, L.; Jovanović, D.; King, T.; Reynolds, A.; and Tinelli, C. 2011. Cvc4. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*.

[2015] Barrett, C.; Fontaine, P.; and Tinelli, C. 2015. The SMT-LIB Standard: Version 2.5. Technical report, The University of Iowa.

[2014] Bartholomew, M., and Lee, J. 2014. System aspmt2smt: Computing aspmt theories by smt solvers. In *European Conference on Logics in Artificial Intelligence, JELIA*, 529–542. Springer.

[2016] Bartholomew, M. 2016. *Answer Set Programming Modulo Theories*. Ph.D. Dissertation, Arizona State University.

[2017] Biavaschi, S. 2017. Automated reasoning methods in hybrid systems. Annual Report of "Scuola Superiore dell'Università di Udine".

[2011] Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

[2011] Calimeri, F.; Ianni, G.; Ricca, F.; and et al. 2011. The third answer set programming competition: Preliminary report of the system competition track. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 388–403. Springer.

[2016] Calimeri, F.; Gebser, M.; Maratea, M.; and Ricca, F. 2016. Design and results of the fifth answer set programming competition. *Artif. Intell.* 231(C):151–181.

[1978] Clark, K. 1978. Negation as failure. In *Logic and Data Bases*. Plenum Press. 293–322.

[2008] De Moura, L., and Bjørner, N. 2008. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340.

[2017] Dutertre, B. 2017. yices. http://yices.csl.sri.com/.

[1994] Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science* 1:51–60.

[2011] Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in gringo series 3. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 345–351. Springer.

[2009] Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In *Proceedings of 25th International Conference on Logic Programming*, 235–249. Springer.

[2017] Janhunen, T.; Kaminski, R.; Ostrowski, M.; Schaub, T.; Schellhorn, S.; and Wanko, P. 2017. Clingo goes linear constraints over reals and integers. *CoRR* abs/1707.04053.

[2013] Lee, J., and Meng, Y. 2013. Answer set programming modulo theories and reasoning about continuous changes. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13), Beijing, China, August 3-9, 2013*.

[2017] Lierler, Y., and Susman, B. 2017. On relation between constraint answer set programming and satisfiability modulo theories. *Theory and Practice of Logic Programming* 17(4):559–590.

[2012] Liu, G.; Janhunen, T.; and Niemela, I. 2012. Answer set programming via mixed integer programming. In *Knowledge Representation and Reasoning Conference*.

[1998] Marriott, K., and Stuckey, P. J. 1998. *Programming with Constraints: An Introduction*. MIT Press.

[2008] Niemela, I. 2008. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence* 53:313–329.

[2018] Ostrowski, M. 2018. *Modern Constraint Answer Set Solving*. Dissertation, University of Potsdam.

[2018] Shen, D., and Lierler, Y. 2018. Smt-based answer set solver cmodels-diff (system description). In *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*.

[2016] Susman, B., and Lierler, Y. 2016. SMT-Based Constraint Answer Set Solver EZSMT (System Description). In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*, volume 52, 1:1–1:15.

[2015] Tinelli, C., and Barrett, C. 2015. Auflira. http://smtlib.cs.uiowa.edu/logics-all.shtml#AUFLIRA.

[2017] Young, R.; Balduccini, M.; and Israney, A. 2017. Casp for robot control in hybrid domains. In *In Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP17)*.