ASPQ: An ASP-Based 2QBF Solver

ASPQ: An ASP-Based 2QBF Solver

Giovanni Amendola, Carmine Dodaro, Francesco Ricca

Department of Mathematics and Computer Science
University of Calabria, Italy
{amendola,dodaro,ricca}@mat.unical.it

Abstract. Answer Set Programming (ASP) is an established logic-based programming paradigm which has been successfully applied for solving complex problems. Since ASP can model problems up to the second level of the polynomial hierarchy, it can be used to model and solve the 2QBF problem. In this paper we show how to obtain a fairly effective 2QBF solver by just resorting to state-of-the-art ASP solvers.

1 Introduction

Answer Set Programming (ASP) [8] is a declarative programming paradigm that has been developed in the field of logic programming and nonmonotonic reasoning. The idea of ASP is to represent a given computational problem by means of a logic program whose stable models [12] (or answer sets) correspond to the desired solutions, and then to use an ASP solver to actually compute the stable models.

ASP has been used in numerous scientific applications in the areas of artificial intelligence [5], bioinformatics [14], databases [7, 15], and game theory [4]. Moreover, ASP is attracting increasing interest also beyond the scientific community, and counts already some successful application in industrial products [13]. ASP has become a popular choice for solving complex problems since it combines an expressive language with efficient implementations. Indeed, the results of the latest ASP Competition series [9] witness the continuous improvements achieved in the field of ASP solving.

The core language of ASP, which features disjunction in rule heads and nonmonotonic negation in rule bodies, can be used to solve all problems at the second level of the polynomial hierarchy. This result was obtained by Eiter and Gottlob [10] that provided a reduction from 2QBF to the problem of verifying the existence of an answer set of a disjunctive ASP program. Given the large progress measured in the last few years in ASP solving, it is natural to ask whether this solving technology can applied profitably also for solving 2QBFs.

In this paper we provide a first answer to this question, by showing that a fairly effective 2QBF solver can be obtained by using state-of-the-art ASP solving technology. To this end, we implemented a tool whose input is a formula in QDIMACS format and produces as output the corresponding ASP program applying the encoding proposed by Eiter and Gottlob. The ASP program obtained is subsequently evaluated combining two state-of-the-art ASP solvers (that employ different techniques for handling hard problems). The resulting proof-of-concept 2QBF solver, called ASPQ, entered as non-competitive participant the QBF competition in 2016. ASPQ obtained a fairly acceptable

result in the 2QBF track, obtaining (virtually) the fifth place, thus performing better than various native QBF solvers. These results witness that the capabilities of state-of-the-art ASP solvers can be exploited for solving 2QBF.

2 Answer Set Programming

In this section we overview the language of ASP. Following the traditional grounding view [12], we concentrate on programs over a propositional signature Λ. A *disjunctive rule* r is of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1, ..., b_m, \texttt{ not } b_{m+1}, ..., \texttt{ not } b_n, \tag{1}$$

where all a_i and b_j are atoms (from Λ) and $l \geq 0$, $n \geq m \geq 0$ and $l+n > 0$; `not` represents *negation-as-failure*. The set $H(r) = \{a_1, ..., a_l\}$ is the *head* of r, while $B^+(r) = \{b_1, ..., b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ are the *positive body* and the *negative body* of r, respectively; the *body* of r is $B(r) = B^+(r) \cup B^-(r)$. We denote by $At(r) = H(r) \cup B(r)$ the set of all atoms occurring in r. A rule r is a *fact*, if $B(r) = \emptyset$ (we then omit $\leftarrow$); a *constraint*, if $H(r) = \emptyset$; *normal*, if $|H(r)| \leq 1$; and *positive*, if $B^-(r) = \emptyset$. A *(disjunctive logic) program* P is a finite set of disjunctive rules. P is called *normal* [resp. *positive*] if each $r \in P$ is normal [resp. positive]. We let $At(P) = \bigcup_{r \in P} At(r)$.

Any set $I \subseteq \Lambda$ is an *interpretation*; it is a *model* of a program P (denoted $I \models P$) if, and only if, for each rule $r \in P$, $I \cap H(r) \neq \emptyset$ if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ (denoted $I \models r$). A model M of P is *minimal*, if, and only if, no model $M' \subset M$ of P exists. We denote by $MM(P)$ the set of all minimal models of P and by $AS(P)$ the set of all *answer sets (or stable models)* of P, i.e., the set of all interpretations I such that $I \in MM(P^I)$, where P^I is the *Gelfond-Lifschitz reduct* [12] of P with respect to I, i.e., the set of rules $a_1 \vee ... \vee a_l \leftarrow b_1, ..., b_m$, obtained from rules $r \in P$ of form (1), such that $B^-(r) \cap I = \emptyset$. A program P such that $AS(P) \neq \emptyset$ is called *coherent*, otherwise it is called *incoherent* [3].

Example 1. Consider the program $P = \{b \vee c \leftarrow \texttt{not } a;\ d \leftarrow c, \texttt{not } b\}$. It has the minimal models $MM(P) = \{\{a\}, \{b\}, \{c,d\}\}$. Instead, the set of all answer sets is $AS(P) = \{\{b\}, \{c,d\}\}$. Hence, P is coherent. Note that $I = \{a\}$ is not an answer set of P, since I is not a minimal model of $P^I = \{d \leftarrow c\}$.

3 Encoding 2QBF in ASP

In this section, we introduce the translation from 2QBFs to logic programs proposed by Eiter and Gottlob [10] to prove the Σ_2^P-hardness of checking whether a disjunctive logic program has some answer set.

To describe the translation, let $\Phi = \exists X \forall Y F$ be a quantified boolean formula, where we may assume that $X = \{x_1, \ldots, x_e\}$, $Y = \{y_1, \ldots, y_a\}$ and $F = D_1 \vee \ldots \vee D_m$, such that $D_i = L_{i,1} \wedge \ldots \wedge L_{i,k}$, where $1 \leq k \leq a+e$ and $L_{i,j}$ are literals over $X \cup Y$. For every atom $z \in X \cup Y$, we introduce a fresh atom z'. Moreover, let σ be a function mapping literals from atoms $z \in X \cup Y$ to atoms as follows:

$$\sigma(L) = \begin{cases} z' \text{ if } L = \neg z, \\ L \text{ otherwise} \end{cases}$$

Algorithm 1: ASPQ-main

Input : A 2-*QBF* formula Φ
Output: SAT or UNSAT

```
1  begin
2      T_bloqqer := 120s; T_clasp := 60s          // QBFEval settings;
3      Φ := BLOQQER (T_bloqqer, Φ);
4      if Φ = ⊤ then return SAT;                  // solved by BLOQQER
5      if Φ = ⊥ then return UNSAT;                // solved by BLOQQER
6      Π := QDimacs2ASP(Φ);                       // encode the logic program
7      res := CLASP (T_clasp, Π);                 // run CLASP for T_clasp seconds
8      if res = UNKNOWN then res := WASP (Π);     // run WASP if unsolved
9      if res = COHERENT then return UNSAT;
10     else return SAT;
```

Finally, we introduce one more fresh atom, say w, and define a disjunctive logic program P_Φ to consist of the following rules:

$z \vee z'$	for each $z \in X \cup Y$
$y \leftarrow w$ and $y' \leftarrow w$	for each $y \in Y$
$w \leftarrow \sigma(L_{i,1}), \ldots, \sigma(L_{i,k})$	for each D_i, $i = 1, \ldots, m$
$w \leftarrow$ `not` w	

Eiter and Gottlob [10] proved that Φ is true if, and only if, P_Φ has an answer set.

Example 2. Consider the 2QBF $\Phi = \exists x \forall y \forall z((x \wedge y) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg y \wedge z))$. Therefore, the corresponding logic program is $P_\Phi = \{x \vee x';\ y \vee y';\ z \vee z';\ y \leftarrow w;\ y' \leftarrow w;\ z \leftarrow w;\ z' \leftarrow w;\ w \leftarrow x, y;\ w \leftarrow x, y', z';\ w \leftarrow y', z;\ w \leftarrow$ `not` $w\}$, which has as unique answer set $\{x, y, y', z, z', w\}$, corresponding to set x to true in Φ, so that Φ is true.

Example 3. Consider the 2QBF $\Phi = \exists x \exists y \forall z((x \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z))$. Therefore, the corresponding logic program is $P_\Phi = \{x \vee x';\ y \vee y';\ z \vee z';\ z \leftarrow w;\ z' \leftarrow w;\ w \leftarrow x, z';\ w \leftarrow x', y, z'; w \leftarrow$ `not` $w\}$, which is incoherent. Indeed, there are only two choices to infer w, $\{x, z'\}$ and $\{x', y, z'\}$. Therefore, an interpretation candidate must contain one of the two sets. In both cases, it cannot be an answer set. Indeed, we can have $I_1 = \{x, y, z, z', w\}$, $I_2 = \{x, y', z, z', w\}$, and $I_3 = \{x', y, z, z', w\}$; but $\{x, y, z\} \in MM(P^{I_1})$, $\{x, y', z\} \in MM(P^{I_2})$, and $\{x', y, z\} \in MM(P^{I_3})$. In conclusion, Φ is false.

4 The ASPQ 2QBF Solver

Main Algorithm. The main algorithm implemented in ASPQ is reported as Algorithm 1. The input formula Φ is first simplified by the preprocessor BLOQQER [6], which replaces Φ by a (usually) smaller equisatisfiable formula. The simplification process can take significant time in case of huge formulas. Hence, the tool is allowed to run for at most $T_{bloqqer}$ seconds. Φ is not modified if BLOQQER exceeds the allotted time. Note that BLOQQER might be able to simplify the formula up to solving it. In that case, it

	Total		SAT		UNSAT	
Solver	**#**	**Time**	**#**	**Time**	**#**	**Time**
areqs	235	2963.33	179	2136.52	56	826.81
rareqs+bloqqer	232	5287.58	156	2084.94	76	3202.64
depqbf-v2	223	5135.23	142	1553.21	81	3582.02
xb-qsts	206	5581.42	154	3354.41	52	2227.01
ASPQ	**188**	**741.09**	**141**	**275.41**	**47**	**465.68**
$\downarrow +15$	...	...	...	...	...	

Table 1. QBFEval 2016 Official Results of the 2QBF Track.

(conventionally) returns a tautology for SAT formulas or a contradiction for UNSAT. This case is exploited to terminate immediately the computation and return the result. Otherwise, Φ is encoded as a propositional ASP program Π as detailed in Section 3. The program Π is subsequently provided as input of the ASP solver CLASP [11], which is executed for T_{clasp} seconds. If CLASP is not able to find an answer set within the allotted time, then WASP [2] is executed without time limits. The reason for using two solvers comes from the observation that CLASP and WASP employ different strategies for solving disjunctive programs (see [1, 11] for details), which may solve different sets of instances.

QBFEval setting. ASPQ entered the QBFEval 2016 in the 2QBF track as non competing system (it was submitted two days after the official solver submission deadline). We set $T_{bloqqer} = 60s$, so that preprocessing never occupies more than 10% of the allotted time (the timeout was set by the organizers to $600s$); and we set $T_{clasp} = 60s$. This choice is motivated by the results of a preliminary experiment. In fact, we observe that CLASP and WASP show a complementary behavior, i.e. the former shows good performances on unsatisfiable instances whereas the latter shows good performances on satisfiable instances. Moreover, in our experimental setting, CLASP finds a solution after few minutes of the computation whereas WASP on average needs more than 5 minutes. Thus, we allotted 10% of the available time to CLASP and the remaining 80% to WASP.

Notes on performance. Table 1 reports for ease of presentation the first five positions of the QBF competition in 2016 (full results are available at `http://www.qbflib.org/index_eval.php`), and ASPQ is outlined using a boldface font. We note that, despite ASPQ is based on a simple architecture, and we used a straightforward static parameter setting strategy, ASPQ could solve 188 instances of the 305 used in the competition, (virtually) ranking on the fifth position over 21 participants (one of which was disqualified as problematic solver). ASPQ is not far from *xb-qst* occupying the fourth position, whereas *areqs* (the winner of the track) solves 20% more instances. We thus note that the straight application of ASP solving techniques lead to a fairly efficient 2QBF solver, outperforming 15 solvers designed explicitly for solving QBF formulas, which is a remarkable result. Moreover, 2QBF instances from the QBFEval can be used as a reference benchmark for improving ASP solvers on hard problems for the second level of the polynomial hierarchy.

5 Conclusion

The main goal of this paper was to provide an assessment of the applicability of ASP solver technology for solving 2QBF formulas. The resulting solver, called ASPQ, entered as non-competitive partecipant the QBF competition in 2016 obtaining a fairly acceptable result in the 2QBF track, that is (virtually) the fifth place. The solver demonstrates that it is reasonable to exploit the capabilities of state-of-the-art ASP solvers for solving 2QBF instances. At the same time, it confirms that 2QBF instances can be used to provide a hard benchmark to assess and improve the performance of ASP systems [16]. As far as future work is concerned, we are considering to tune our system by improving the encoding in ASP and exploring the possibility of using techniques from ASP portfolios for further improving the performance of our system.

References

1. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: LPNMR. LNCS, vol. 8148, pp. 54–66. Springer (2013)
2. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: LPNMR. LNCS, vol. 9345, pp. 40–54. Springer (2015)
3. Amendola, G., Eiter, T., Fink, M., Leone, N., Moura, J.: Semi-equilibrium models for paracoherent answer set programs. Artif. Intell. 234, 219–271 (2016)
4. Amendola, G., Greco, G., Leone, N., Veltri, P.: Modeling and reasoning about NTU games via answer set programming. In: IJCAI. pp. 38–45 (2016)
5. Balduccini, M., Gelfond, M., Watson, R., Nogueira, M.: The USA-Advisor: A Case Study in Answer Set Planning. In: LPNMR. LNCS, vol. 2173, pp. 439–442. Springer (2001)
6. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: CADE. LNCS, vol. 6803, pp. 101–115. Springer (2011)
7. Bravo, L., Bertossi, L.E.: Logic programs for consistently querying data integration systems. In: IJCAI. pp. 10–15 (2003)
8. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
9. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. 231, 151–181 (2016)
10. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. 15(3-4), 289–323 (1995)
11. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp Series 3. In: LPNMR. LNCS, vol. 9345, pp. 368–383. Springer (2015)
12. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Comput. 9(3/4), 365–386 (1991)
13. Grasso, G., Leone, N., Manna, M., Ricca, F.: ASP at work: Spin-off and applications of the DLV system. In: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565. Springer (2011)
14. Koponen, L., Oikarinen, E., Janhunen, T., Säilä, L.: Optimizing phylogenetic supertrees using answer set programming. TPLP 15(4-5), 604–619 (2015)
15. Manna, M., Ricca, F., Terracina, G.: Taming primary key violations to query large inconsistent data via ASP. TPLP 15(4-5), 696–710 (2015)
16. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in DLV: implementation, evaluation, and comparison to QBF solvers. J. Algorithms 63(1-3), 70–89 (2008)