2-ASP(Q) Solving Based on CEGAR

Andrea Cuteri[1] , **Giuseppe Mazzotta**[1] , **Francesco Ricca**[1]
[1]University of Calabria, Rende, Italy
andrea.cuteri@unical.it, giuseppe.mazzotta@unical.it, francesco.ricca@unical.it

1 Context and Motivation

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczynski 2011; Brewka, Eiter, and Truszczynski 2016) is a well-recognized KRR formalism based on the stable model semantics (Gelfond and Lifschitz 1991) which allows to compactly model problems in NP via the typical Guess & Check paradigm (Brewka, Eiter, and Truszczynski 2011). Even though ASP is expressive enough to model problems up to the second level of the Polynomial Hierarchy (PH), the modeling of problems beyond NP within ASP is neither elegant nor intuitive (Gebser, Kaminski, and Schaub 2011). Answer Set Programming with Quantifiers (ASP(Q)) extends the ASP language by introducing existential and universal quantifiers over the answer sets of ASP programs, thus enabling the modeling of problems across the entire Polynomial Hierarchy (PH) (Amendola, Ricca, and Truszczynski 2019). In particular, by alternating existential and universal quantifiers over n ASP programs, it is possible to encode problems at the n-th level of the PH, namely the complexity classes Σ_n^P and Π_n^P. For instance, complete problems for the second level of the PH (i.e., Σ_2^P and Π_2^P) can be naturally represented as 2-ASP(Q) programs, that are, programs consisting of two quantifiers.

More formally, a 2-ASP(Q) program Π is of the form:

$$\Box_1 P_1 \Box_2 P_2 : C \tag{1}$$

where $\Box_1, \Box_2 \in \{\exists^{st}, \forall^{st}\}$ with $\Box_1 \neq \Box_2$, P_1 and P_2 are programs, and C is a stratified program with constraints (Ceri, Gottlob, and Tanca 1990). Intuitively, the semantics of 2-ASP(Q) programs where $\Box_1 = \exists^{st}$, namely existential 2-ASP(Q), can be formulated as: "there is an answer set of a program P_1 such that for every answer set of a program P_2, the program C, modeling admissibility of a solution, is coherent," (a dual sentence starting with "for all answer sets of a program P_1" can be used for 2-ASP(Q) where $\Box_1 = \forall^{st}$, that are universal 2-ASP(Q)). Determining whether such a condition holds corresponds to the main reasoning task in ASP(Q), known as the *coherence problem*. The evaluation of the coherence of ASP(Q) programs is therefore the central task performed by ASP(Q) solvers.

State-of-the-art ASP(Q) solvers (Amendola et al. 2022; Faber, Mazzotta, and Ricca 2023) translate ASP(Q) into QBF and then exploit efficient QBF solvers (Hecking-Harbusch and Tentrup 2018; Lonsing and Egly 2017; Janota et al. 2012) to determine the coherence of ASP(Q) programs. Even though this rewriting represents a viable solution, it has been empirically proved that that obtained QBF formulae tend to be very large and hard to solve for QBF solvers (Amendola et al. 2022; Faber, Mazzotta, and Ricca 2023). This naturally raises the question of whether a direct implementation, taking inspiration from techniques used in QBF solvers, can overcome this limitation. In this work, we lift the Counterexample Guided Abstraction Refinement (CEGAR) (Clarke et al. 2003) technique to 2-ASP(Q) solving. In particular, we propose an evaluation approach which relies on programs transformations and on ASP solvers used as oracles. We then present an implementation of this technique in a novel system called CASPER. Empirical results show that CASPER consistently outperforms state-of-the-art ASP(Q) solvers.

2 Solving 2-ASP(Q) via CEGAR

CEGAR (Clarke et al. 2003) is an iterative abstraction-refinement technique which was successfully applied for QBF solving (Janota et al. 2016). In particular, Janota et al. defined a CEGAR-based approach for QBF solving rooted in a game-centric view of the QBF semantics. Following the same line, we now introduce a game-centric view of 2-ASP(Q) semantics on top of which it is possible to build a CEGAR-based approach for evaluating 2-ASP(Q) program.

Game Centric View of 2-ASP(Q). Let Π be a 2-ASP(Q) program of the form (1). An answer set M_1 of P_1 is a *move* for $\Box_1$. *Candidate countermoves* to M_1 for $\Box_2$ are all the answer sets of P_2 given M_1 (i.e., the program P_2 is augmented with facts and constraint encoding M_1). A candidate countermove M_2 is a *countermove* if either C given M_2 is incoherent and $\Box_2 = \forall^{st}$; or C given M_2 is coherent and $\Box_2 = \exists^{st}$. A *winning move* for $\Box_1$ is a move M_1 of $\Box_1$ such that there does not exist a countermove for $\Box_2$ to M_1. $\Box_1$ *wins* if there exists a winning move for them.

Proposition 1. *Let Π be a 2-ASP(Q) program of the form (1) then there exists a winning move for $\Box_1$ iff (i) Π is incoherent and $\Box_1 = \forall^{st}$ or (ii) Π is coherent and $\Box_1 = \exists^{st}$.*

CEGAR for 2-ASP(Q) Based on the game-centric view for the semantics of 2-ASP(Q) programs, it is possible to

define a CEGAR-based approach for deciding the coherence of 2-ASP(Q) programs. To this end, we need to define a way for (i) computing countermoves for $\square_2$ to moves of $\square_1$ (i.e., counterexample search); and (ii) computing moves of $\square_1$ which avoid known countermoves for $\square_2$ (abstraction refinement). These two tasks can be carried out by defining ad-hoc ASP programs, namely countermove program ctr and refined program ref, whose answer sets correspond to countermoves and possible winning moves respectively. Intuitively, the countermove program ctr can be obtained by combining the programs P_2 and C according to the quantifier $\square_2$. If $\square_2 = \forall^{st}$, then countermoves correspond to answer sets of P_2 which make the program C incoherent. Thus, ctr can be obtained as $P_2 \cup \neg C$ where $\neg C$ is obtained from C by enforcing the violation of at least one constraint. Otherwise, since $\square_2 = \exists^{st}$, then countermoves correspond to answer sets of P_2 which make the program C coherent. In this case, ctr is trivially obtained as $P_2 \cup C$.

Conversely, the refined program ref is aimed at computing moves for $\square_1$ that do not admit previously discovered countermoves. Let us consider a move M_1 for $\square_1$, and a countermove M_2 to M_1 for $\square_2$. Intuitively, M_1' is a new move for $\square_1 = \exists^{st}$ (resp. $\square_1 = \forall^{st}$) that does admit M_2 as countermove if one of the following conditions hold:

1. M_2 is not an answer set of P_2 given M_1';
2. M_2 is an answer set of P_2 given M_1' but C given M_2 is coherent (resp. incoherent).

Starting from this intuition, the refined program can be defined by combining P_1, P_2, and C. More precisely, the program P_1 is kept in its original form, since newly computed moves must still be answer sets of P_1. The program P_2 is simplified according to M_2 by considering the GL-reduct (Gelfond and Lifschitz 1991) of P_2 with respect to M_2, in order to verify whether M_2 remains an answer set of P_2 under the new move. Finally, the program C is rewritten into a program C' that is activated only if M_2 is again an answer set; C' then checks the coherence of C according to $\square_1$, that is, C must be coherent if $\square_1 = \exists^{st}$ and incoherent otherwise. In this way, the refined program admits only answer sets of the P_1 such that M_2 is not again a countermove. For further details refer to Definitions 3 and 4 from Cuteri, Mazzotta, and Ricca. Starting from these intuitive definitions of countermove and refined programs, it is possible to design a CEGAR-based Algorithm to decide the coherence of 2-ASP(Q) programs (see Algorithm 1). The idea behind this approach is to interleave two calls to an ASP solver acting as an oracle: one to compute possible winning moves and another to attempt to disprove them by computing countermoves. These two operations are repeated in a loop until one of the following conditions holds: (i) there exists a move for which no countermove can be found, in which case the move is winning; (ii) every possible move admits at least one of the discovered countermoves, implying that no winning move exists. Consequently, if condition (i) holds, Algorithm 1 returns a winning move M_1; otherwise, it returns $NULL$, indicating there is no winning move.

The CEGAR for 2-ASP(Q) approach has been implemented into a new ASP(Q) solver named CASPER, which leverages the CLINGO Python API to compute answer sets of the refined and countermove programs. CASPER is available on github and ready to use as a python package (https://github.com/ndria00/Casper.git).

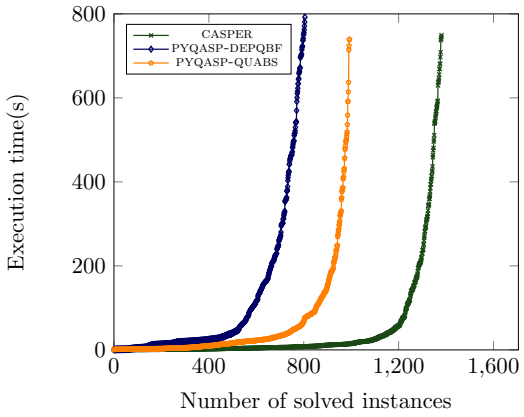

Figure 1: Overall execution time

Algorithm 1 CEGAR for 2-ASP(Q)

Input: A 2-ASP(Q) program Π of the form $\square_1 P_1 \square_2 P_2 : C$
Output: A winning move M_1 for $\square_1$

$CMs = \emptyset$
M_1 = solve(P_1)
while $M_1 \neq \bot$ **do**
  $M_2 = solve(ctr(\Pi) \cup fix_{P_1}(M_1))$.
  if $M_2 = \bot$ **then**
    return M_1
  else
    $M = M_2|_{\mathcal{H}(P_2)}$
    $CMs = CMs \cup \{M\}$
    $Ref = P_1 \cup \bigcup_{M \in CMs} \Pi^M$
  end if
  $Next$ = solve(Ref)
  $M_1 = Next|_{\mathcal{H}(P_1)}$
end while
return $NULL$

Experiments and Conclusion. To assess the impact of the proposed approach, we conducted an experimental evaluation. Benchmarks and source code are available at https://osf.io/wcy5n. The evaluation includes hard benchmarks from the literature (Amendola et al. 2022; Faber, Mazzotta, and Ricca 2023), as well as two newly introduced benchmarks. The results, summarized in the cactus plot in Figure 1, clearly show that CASPER consistently outperforms state-of-the-art ASP(Q) systems. This advantage can be attributed to two main factors: first, CASPER avoids the overhead of translating ASP(Q) programs into QBF formulae; second, the proposed refinement transformation effectively prunes the space of candidate solutions (i.e., answer sets of the first program) and rapidly converges to prove (in)coherence of 2-ASP(Q) programs.

In conclusion, this approach gives birth to a new class of *efficient* ASP(Q) solvers, not requiring translation to QBF.

3 Acknowledgments

This work has been partially funded by the Italian Ministry of Industrial Development (MISE) under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005; by the Italian Ministry of Research (MUR) under project PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006; and also partially by projects SIGENERA (CUP J29I24001770005) and MOZART (J49I24001740005) selected within the framework of the PR FESR – FSE Calabria 2021/2027 and implemented with the support of the Italian State and the Calabria Region.

References

Amendola, G.; Cuteri, B.; Ricca, F.; and Truszczynski, M. 2022. Solving problems in the polynomial hierarchy with ASP(Q). In *LPNMR*, volume 13416 of *Lecture Notes in Computer Science*, 373–386. Springer.

Amendola, G.; Ricca, F.; and Truszczynski, M. 2019. Beyond NP: quantifying over answer sets. *Theory Pract. Log. Program.* 19(5-6):705–721.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2016. Answer set programming: An introduction to the special issue. *AI Mag.* 37(3):5–6.

Ceri, S.; Gottlob, G.; and Tanca, L. 1990. *Logic Programming and Databases*. Surveys in computer science.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50(5):752–794.

Cuteri, A.; Mazzotta, G.; and Ricca, F. 2026. 2-asp(q) solving based on cegar. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Faber, W.; Mazzotta, G.; and Ricca, F. 2023. An efficient solver for ASP(Q). *Theory Pract. Log. Program.* 23(4):948–964.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex optimization in answer set programming. *TPLP* 11(4-5):821–839.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.* 9(3/4):365–386.

Hecking-Harbusch, J., and Tentrup, L. 2018. Solving QBF by abstraction. In *GandALF*, volume 277 of *EPTCS*, 88–102.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2012. Solving QBF with counterexample guided refinement. In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, 114–128. Springer.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2016. Solving QBF with counterexample guided refinement. *Artif. Intell.* 234:1–25.

Lonsing, F., and Egly, U. 2017. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *CADE*, volume 10395 of *Lecture Notes in Computer Science*, 371–384. Springer.