TPLP: Page 1–18.

doi:10.1017/S1471068426100477

*2-ASP(Q) Programs with Weak Constraints: Complexity and Efficient Implementation**

ANDREA CUTERI, GIUSEPPE MAZZOTTA and FRANCESCO RICCA
Department of Mathematics and Computer Science, University of Calabria, Italy
(*e-mails:* andrea.cuteri@unical.it, giuseppe.mazzotta@unical.it, ricca@mat.unical.it)

submitted 29 May 2026 UTC; revised 29 May 2026 UTC; accepted 1 June 2026 UTC

Abstract

ASP(Q) extends Answer Set Programming (ASP) with Quantifiers over answer sets. In this paper, we focus on the class of ASP(Q) programs with two quantifiers and weak constraints, denoted as 2-ASP^w(Q). 2-ASP^w(Q) is a practically relevant fragment of ASP(Q) that is expressive enough to capture optimization problems up to the class Δ_3^P. On the theoretical side, we provide a complete complexity characterization of the main computational tasks for 2-ASP^w(Q) programs, including tight completeness results and the analysis of nontrivial cases that have not been addressed in previous works. On the practical side, we introduce novel strategies for computing (optimal) quantified answer sets in the CASPER system, that rely on a Counterexample-Guided Abstraction Refinement (CEGAR) technique tailored to ASP(Q). An experimental evaluation on hard benchmarks from different application domains shows that the proposed techniques are effective in practice.

Keywords: ASP, quantified ASP, combinatorial optimization problems

1 Introduction

Answer Set Programming (Brewka *et al*. 2011) (ASP) is a well-established declarative formalism that has been widely adopted for modeling and solving hard combinatorial problems. Over the years, ASP has been successfully applied in a variety of real-world domains, including planning (Son *et al*. 2023), scheduling (Dodaro *et al.* 2018; Cardellini *et al*. 2021), business process management (Chiariello *et al*. 2024; Fionda *et al*. 2024), among many others (Eiter *et al*. 2008). A substantial body of research has focused on extending ASP modeling and solving capabilities, leading to the proposal of several language extensions (Bogaerts *et al*. 2016; Amendola *et al.* 2019; Fandinno *et al*. 2021). Among these, Answer Set Programming with Quantifiers (Amendola *et al.* 2019)

* 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 and also partially by projects SIGEN ERA (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.

(ASP(Q)) introduces the possibility to quantify over answer sets (Gelfond and Lifschitz 1991). This extension enhances the expressive power of ASP, allowing for the natural modeling of problems spanning the entire polynomial hierarchy (PH). ASP(Q) has found interesting applications in several contexts, including outlier detection (Bellusci *et al.* 2022), planning (Faber *et al.* 2022), and related domains (Faber 2024), establishing it as a promising framework for solving problems beyond NP. More recently, ASP(Q) has been extended with *weak constraints*, denoted by ASP^w(Q) (Mazzotta *et al.* 2024), further enhancing its ability to naturally represent hard optimization problems. Weak constraints in ASP^w(Q) can be applied both locally to enable quantification over optimal answer sets and globally to express preferences among solutions, thus capturing optimization problems throughout the PH. While weak constraints improve the modeling capabilities of ASP(Q), they introduce additional sources of computational complexity, making the design of effective solving techniques challenging (Azzolini *et al.* 2026).

Within this context, we focus on ASP(Q) programs with two quantifiers and weak constraints, denoted as 2-ASP^w(Q), which are expressive enough to model complex optimization problems up to Δ_3^P. Indeed, the expressive power of 2-ASP^w(Q) is not yet fully characterized, as existing completeness results are available only for specific language fragments (Mazzotta *et al.* 2024). As a consequence, the development of efficient and complexity-aware implementations for the entire 2-ASP^w(Q) has so far been left unexplored. To fill this gap, this paper undertakes a complexity analysis of the main reasoning tasks for 2-ASP^w(Q) programs, namely coherence and brave reasoning, and derives tight completeness results for the general case.

Moreover, the paper introduces efficient solving techniques for 2-ASP^w(Q). In particular, we propose an efficient evaluation technique based in Counterexample-Guided Abstraction Refinement (CEGAR) (Clarke *et al.* 2003; Janota *et al.* 2016), which serves as the foundation for two algorithms computing optimal quantified answer sets. Although inspired by the lower- and upper-bound improving strategies used in ASP solvers (Alviano *et al.* 2020), our lower-bound improving approach departs from existing methods by realizing lower-bound improvement via abstraction refinement instead of unsatisfiable-core extraction, thereby yielding a distinctive and novel technique.

Finally, the proposed techniques have been implemented on top of the CASPER system (Cuteri *et al.* 2026). Experimental results demonstrate the effectiveness of our approach across several benchmarks drawn from diverse application domains, confirming both its practical viability and its alignment with the theoretical complexity results.

2 Background

In this section, we provide some preliminaries and notation on ASP and ASP(Q).

2.1 Answer Set Programming

2.1.1 Syntax

A term is a constant (i.e., an integer or a string starting with lowercase letter) or a variable (i.e., a string starting with uppercase letter). An *atom* is an expression of the form $p(\vec{t})$ with $\vec{t} = t_1, \ldots, t_n$ being a list of terms and p being a *predicate* of arity $n \geq 0$. An atom $p(\vec{t})$ is *ground* if all terms in $\vec{t}$ are constants. A *literal* is either an atom a or

its negation $\sim a$. The *complement* of a literal $l = a$ (resp. $l = \sim a$) is $\bar{l} = \sim a$ (resp. $\bar{l} = a$). Given a set of literals L, L^+ (resp. L^-) denotes the set of positive (resp. negative) literals appearing in L. A *rule* is an expression of the form $h \leftarrow l_1, \ldots, l_n$ where h is an atom referred to as *head*, and $l_1, \ldots, l_n$ with $n \geq 0$ is a conjunction of literals referred to as *body*. A rule with empty body is called *fact*, while a rule with empty head is called *hard constraint*. A *weak constraint* is an expression of the form $\leftarrow^{\omega} l_1, \ldots, l_n \ [w@l, \vec{t}]$ where $l_1, \ldots, l_n$ is a conjunction of literals referred to as body, w and l are terms, and $\vec{t} = t_1, \ldots, t_m$ is a possibly empty list of terms. Given a rule (resp. a weak constraint) r, H_r denotes the set of atoms appearing in the head of r while B_r denotes the set of literals appearing in the body of r. A rule r (resp. a weak constraint) is *safe* if each variable appears in at least one literal in B_r^+. A *program* P is a set of safe rules and weak constraints. Given a program P, $\mathcal{R}(P)$, and $\mathcal{W}(P)$ denote, respectively, the sets of rules and weak constraints appearing in P; while $\mathcal{H}(P)$ denotes the set of atoms appearing in the head of some rules in P. Given an expression ϵ (atom, program, etc.), $at(\epsilon)$ denotes the set of atoms appearing in ϵ.

2.1.2 Semantics

Given an ASP program P, the *Herbrand Universe*, HU_P, of P is the set of all constants appearing in P; the *Herbrand Base*, B_P, is the set of all possible ground atoms that can be obtained from predicates and constants in P; $ground(P)$ denotes the set of all ground rules that can be obtained from P by proper substitutions of variables in P with constants in HU_P. An *interpretation* $I \subseteq B_P$ is a set of atoms. A ground literal $l = a$ (resp. $l = \sim a$) is true w.r.t. I if $a \in I$ (resp. $a \notin I$), false otherwise. A conjunction of literals *conj* is true w.r.t. I if all the literals in *conj* are true w.r.t. I, false otherwise. An interpretation I is an *answer set* of P iff (i) I is a model of P, namely for each rule $r \in ground(P)$ either H_r is true w.r.t. I or B_r is false w.r.t. I; and (ii) I is a minimal model of its GL-reduct (Gelfond and Lifschitz 1991). Let $AS(P)$ be the set of answer set of a program P, then P is *coherent* iff $AS(P) \neq \emptyset$. For a program P and an interpretation I, let the set of weak constraint violations be $ws(P, I) = \{(w, l, \vec{t}) \mid \leftarrow^{\omega} b_1, \ldots, b_m \ [w@l, \vec{t}] \in ground(P)$ and $b_1, \ldots, b_m$ are true w.r.t. $I\}$, then the cost function of P is defined as $\mathcal{C}(P, I, k) = \sum_{(w,k,\vec{t}) \in ws(P,I)} w$ for every integer k. Let $M_1, M_2 \in AS(P)$ then M_1 is *dominated* by M_2 if there exists an integer l such that $\mathcal{C}(P, M_1, l) > \mathcal{C}(P, M_2, l)$ and for each $l' > l$, $\mathcal{C}(P, M_1, l') = \mathcal{C}(P, M_2, l')$. Let $M \in AS(P)$ then M is an *optimal answer set* iff M is not dominated by any $M' \in AS(P)$. We denote by $OptAS(P)$ the set of optimal answer set of P.

2.2 ASP with two quantifiers

A 2-ASP^w(Q) program (Mazzotta *et al.* 2024) is an expression of the form $\Box_1 P_1 \Box_2 P_2 : C : C^{\omega}$, where $\Box_1, \Box_2$ are quantifiers in $\{\exists^{st}, \forall^{st}\}$, P_1, P_2 are ASP programs possibly with weak constraints, C is a stratified program (Ceri *et al.* 1990) with hard constraints, and C^{ω} is a set of weak constraints such that $B_{C^{\omega}} \subseteq B_{P_1}$. Weak constraints appearing P_1 and P_2 are said *local*; whereas those in C^{ω} are said *global*. A 2-ASP^w(Q) program Π is said to be *existential* if $\Box_1 = \exists^{st}$, otherwise it is *universal*. Moreover, Π is said to be *alternating* if $\Box_1 \neq \Box_2$, and *plain* if it contains no weak constraints. We now define the semantics of 2-ASP^w(Q) programs. Let P be an ASP program and $M \subseteq B_P$, then $fix_P(M)$ denotes the

set of facts and hard constraints of the form $\{a \leftarrow \mid a \in M\} \cup \{\leftarrow a \mid a \in B_P \setminus M\}$. Then, $M \in AS(P)$ *satisfies* a program P' if $P' \cup fix_P(M)$ is coherent.

At this point the coherence of 2-ASP^w(Q) can be defined as follows:

- $\exists^{st} P : C : C^\omega$ is coherent iff there exists $M \in OptAS(P)$ such that M satisfies C.
- $\forall^{st} P : C : C^\omega$ is coherent iff for each $M \in OptAS(P)$, M satisfies C.
- $\exists^{st} P_1 \square_2 P_2 : C : C^\omega$ is coherent iff there exists $M_1 \in OptAS(P_1)$ such that $\square_2 P_2 \cup fix_{P_1}(M_1) : C : C^\omega$ is coherent.
- $\forall^{st} P_1 \square_2 P_2 : C : C^\omega$ is coherent iff for each $M_1 \in OptAS(P_1)$, $\square_2 P_2 \cup fix_{P_1}(M_1) : C : C^\omega$ is coherent.

For an existential 2-ASP^w(Q) program Π, an optimal answer set $M_1 \in OptAS(P_1)$ is a *quantified answer set* of Π if $\square_2 P_2 \cup fix_{P_1}(M_1) : C : C^\omega$ is coherent. We denote by $QAS(\Pi)$ the set of quantified answer sets of Π. Let Π be an existential 2-ASP^w(Q) program, l be an integer, and $M \in QAS(\Pi)$, then the cost of M at level l is defined as $\mathcal{C}(M, \Pi, l) = \mathcal{C}(M, P_1 \cup C^\omega, l)$. Let $M_1, M_2 \in QAS(\Pi)$, then M_1 is *dominated* by M_2 if there exists l such that $\mathcal{C}(M_1, \Pi, l) > \mathcal{C}(M_2, \Pi, l)$ and for each $l' > l$, $\mathcal{C}(M_1, \Pi, l') = \mathcal{C}(M_2, \Pi, l')$. Thus, M is an *optimal quantified answer set* if M is not dominated by any $M' \in QAS(\Pi)$. We denote by $OptQAS(\Pi)$ the set of optimal quantified answer sets.

Example 1.
Let Π be a 2-ASP^w(Q) program of the form $\exists^{st} P_1 \forall^{st} P_2 : C$, where $C = \{\leftarrow nb,\ nc\}$ and

$$P_1 = \left\{ \begin{array}{ll} a \leftarrow \sim na & b \leftarrow \sim nb \\ na \leftarrow \sim a & nb \leftarrow \sim b \end{array} \right\} \qquad P_2 = \left\{ \begin{array}{ll} c \leftarrow \sim nc & \leftarrow^\omega a,\ \sim c\ [1@1] \\ nc \leftarrow \sim c & \leftarrow^\omega b,\ \sim nc\ [1@1] \end{array} \right\}$$

Let us consider $M_1 = \{na, nb\} \in AS(P_1)$. In this case, $\{na, nb, nc\} \in OptAS(P_2 \cup fix_{P_1}(M_1))$ violates the constraint $\leftarrow nb, nc \in C$. Thus, $M_1 \notin QAS(\Pi)$. Conversely, $M_1' = \{a, nb\} \in AS(P_1)$ is such that $P_2 \cup fix_{P_1}(M_1')$ admits only one optimal answer set $M_2 = \{a, nb, c\}$ which satisfies the constraint in C and so $M_1 \in QAS(\Pi)$. If we remove weak constraints from P_2, then $AS(P_2 \cup fix_{P_1}(M_1)) = \{M_2, M_2'\}$ where $M_2' = \{a, nb, nc\}$. Here, M_2' violates the constraint in C and so M_1 is not a quantified answer set anymore.

3 Complexity results for 2-ASP^w(Q)

In this section, we study the main computational tasks for 2-ASP^w(Q): Coherence and Brave reasoning. For 2-ASP^w(Q) programs, the coherence problem checks whether a program is coherent, while brave reasoning verifies whether an atom occurs in an optimal quantified answer set. Unlike standard ASP, local weak constraints in ASP^w(Q) programs introduce an additional source of computational complexity, as observed by Mazzotta *et al.* (2024). Even though complexity results for these tasks have been established for specific subclasses of ASP^w(Q), a complete characterization is still missing.

In particular, it was shown that verifying the coherence for 2-ASP^w(Q) programs (i) is in Σ_3^P for existential programs and in Π_3^P for universal programs, and (ii) is hard for Σ_2^P and Π_2^P, respectively. Starting from these results, we prove that the coherence problem for 2-ASP^w(Q) is complete for the second level of the PH, namely Σ_2^P for existential programs and Π_2^P for universal programs (full proofs in the online appendix).

Theorem 1 (Membership).
The coherence problem for 2-ASP^w(Q) programs of the form $\square_1 P_1 \square_2 P_2 : C$ is in: Σ_2^P if $\square_1 = \exists^{st}$; Π_2^P otherwise, no matter whether $\square_2 = \exists^{st}$ or $\square_2 = \forall^{st}$

Proof (Sketch)
To prove our thesis we need to distinguish three different scenarios.

Uniform quantifiers $\square_1 = \square_2$. By applying the transformation (Algorithm 1) by Mazzotta *et al.* (2024), it is possible to obtain a plain 2-ASP(Q) program Π' such that Π' is coherent if and only if Π is coherent. From the result of Amendola *et al.* (2019), verifying the coherence of Π' is Σ_2^P-complete if $\square_1 = \exists^{st}$; otherwise is in Π_2^P-complete. Thus the thesis holds for uniform quantifiers.

Case $\square_1 = \exists^{st}$ and $\square_2 = \forall^{st}$. In this case, it is possible to translate Π into an existential 2-ASP^w(Q) program Π' where weak constraints appear only in the first subprogram. Since verifying the coherence of Π' is Σ_2^P-complete (Corollary 3.1 by Mazzotta *et al.* 2024), then the thesis holds also in this case.

More precisely, the 2-ASP^w(Q) program Π' can be obtained by copying the rules of the program P_2 in P_1 and then adding rules in the program C for verifying the optimality of an answer sets of P_2. As a result, we obtain Π' of the form $\exists P_1' \forall P_2' : C'$ where:

- optimal answer sets M of P_1' are of the form $M_1 \cup M_2^c$ where M_1 is an optimal answer set of P_1 and M corresponds to an answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$;
- answer sets N of $P_2' \cup \mathit{fix}_{P_1'}(M)$ are of the form $M_1 \cup M_2^c \cup M_2$ where $M_1 \cup M_2$ is an answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$;
- an answer set N of $P_2' \cup \mathit{fix}_{P_1'}(M)$ satisfies C' if and only if $M_1 \cup M_2^c$ is not dominated by $M_1 \cup M_2$ and one of the following holds:
  - there exists l such that the cost of $M_1 \cup M_2$ is different from the cost of $M_1 \cup M_2^c$;
  - the cost of $M_1 \cup M_2$ is equal to the cost of $M_1 \cup M_2^c$ for each level and $M_1 \cup M_2$ satisfies C.

Let $M = M_1 \cup M_2^c$ be an optimal answer set of P_1'. If M is not an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$, then M cannot be a quantified answer set as there exists $N = M_1 \cup M_2^c \cup M_2$ in the answer sets of $P_2' \cup \mathit{fix}_P(M)$, where $M_1 \cup M_2$ is an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$ and N violates C' since $M_1 \cup M_2^c$ is dominated by $M_1 \cup M_2$. On the other hand, since M is an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$, then for each $N = M_1 \cup M_2^c \cup M_2$ in the answer sets of $P_2' \cup \mathit{fix}_{P_1'}(M)$, $M_1 \cup M_2^c$ is not dominated by $M_1 \cup M_2$ (i.e., Condition 1 is always satisfied). Thus, let $N = M_1 \cup M_2^c \cup M_2$ be an answer set of $P_2 \cup \mathit{fix}_{P_1'}(M)$, then N satisfies C' if and only if one between Conditions 1 and 2 holds. Here we can observe that Condition 1 holds if and only if $M_1 \cup M_2$ is not an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$; whereas Condition 2 holds if and only if $M_1 \cup M_2$ is an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$ and $M_1 \cup M_2$ satisfies C. Thus, M is a quantified answer set of Π' if and only if each optimal answer set $M_1 \cup M_2$ of $P_2 \cup \mathit{fix}_{P_1}(M_1)$ satisfies C. Hence, M is a quantified answer set of Π' if and only if M_1 is a quantified answer set of Π. Finally, we can conclude that Π is coherent iff Π' is coherent.

Case $\square_1 = \forall^{st}$ and $\square_2 = \exists^{st}$. By following the same working principle as before, it is possible to encode Π into a 2-ASP^w(Q) program Π' which preserves the coherence

of Π and in which weak constraints appear only in the first subprogram. In this case, verifying the coherence of Π' is Π_2^P-complete (Corollary 3.1 by Mazzotta *et al.* 2024), then the thesis holds also in this last case. In this case, the 2-ASP^w(Q) program Π' is of the form $\forall P_1' \exists P_2' : C'$ where:

- optimal answer sets M of P_1' are of the form $M_1 \cup M_2^c$ where M_1 is an answer set of P_1 and M is an answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$;
- answer sets N of $P_2' \cup \mathit{fix}_{P_1'}(M)$ are of the form $M_1 \cup M_2^c \cup M_2$ where $M_1 \cup M_2$ is an answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$;
- an answer set N of $P_2' \cup \mathit{fix}_{P_1'}(M)$ satisfies C' if and only if one of the following holds:
  - $M_1 \cup M_2^c$ is dominated by $M_1 \cup M_2$;
  - the cost of $M_1 \cup M_2$ is equal to the cost of $M_1 \cup M_2^c$ for each level and $M_1 \cup M_2$ satisfies C.

According to the ASP^w(Q) semantics, Π' is incoherent if and only if there exists an optimal answer set M_1 of P_1' such that for every answer set N of $P_2' \cup \mathit{fix}_{P_1'}(M))$, N does not satisfy C'. Let $M = M_1 \cup M_2^c$ be an optimal answer set of P_1', with $M_1 \in \mathit{OptAS}(P_1)$ and $M \in \mathit{AS}(P_2 \cup \mathit{fix}_{P_1}(M_1))$.

If $M_1 \cup M_2^c$ is not an optimal answer set of $P_2 \cup \mathit{fix}_{P_1}(M_1)$, then there exists $M_1 \cup M_2 \in \mathit{OptAS}(P_2 \cup \mathit{fix}_{P_1}(M_1))$ such that $M_1 \cup M_2^c$ is dominated by $M_1 \cup M_2$. Consequently, $N = M_1 \cup M_2^c \cup M_2 \in \mathit{AS}(P_2' \cup \mathit{fix}_P(M))$ is such that Condition 1 is satisfied, and thus M cannot witness the incoherence of Π'.

Otherwise, $M_1 \cup M_2^c$ is optimal for $P_2 \cup \mathit{fix}_{P_1}(M_1)$. In this case, for every $N = M_1 \cup M_2^c \cup M_2 \in \mathit{AS}(P_2' \cup \mathit{fix}_{P_1'}(M))$, N satisfies C' if and only if Condition 2 holds, that is, if and only if $M_1 \cup M_2$ has the same cost of $M_1 \cup M_2^c$ (i.e., $M_1 \cup M_2 \in \mathit{OptAS}(P_2 \cup \mathit{fix}_{P_1}(M_1))$) and $M_1 \cup M_2$ satisfies C.

Therefore, Π' is incoherent if and only if there exists $M_1 \in \mathit{AS}(P_1)$ such that no $M_1 \cup M_2 \in \mathit{OptAS}(P_2 \cup \mathit{fix}_{P_1}(M_1))$ satisfies C, which is exactly the condition for Π to be incoherent. Hence, Π' preserves the coherence of Π. □

Note that, the presence of weak constraints in both quantified programs do not allow for a simple quantifier elimination, and the existing translation proposed by Mazzotta *et al.* (2024) requires the introduction of an additional quantifier, resulting in a non necessary jump in complexity to the third level of the PH in case $\Box_1 \neq \Box_2$.

Theorem 2 (Hardness).
The coherence problem for 2-ASP^w(Q) programs of the form $\Box_1 P_1 \Box_2 P_2 : C$ is hard for: Σ_2^P if $\Box_1 = \exists^{st}$; Π_2^P otherwise, no matter whether $\Box_2 = \exists^{st}$ or $\Box_2 = \forall^{st}$.

Proof (Sketch).
Let us consider the different combination of quantifiers separately.

(Case $\Box_1 = \Box_2 = \exists^{st}$). From Theorem 4 by Mazzotta *et al.* (2024), deciding the coherence of Π is Σ_2^P-complete if $\mathcal{W}(P_1) = \emptyset$. Thus, the thesis holds since this is a particular case.

(Case $\square_1 = \square_2 = \forall^{st}$). Let $\Phi = \forall X \exists Y \phi$ be a 2-QBF, where X and Y are disjoint set of variables and ϕ is a boolean formula in 3-CNF. Verifying that Φ is true is a Π_2^P-complete problem (Schaefer and Umans 2002). We encode in polynomial time any Φ in a 2-ASP^w(Q) program of the form $\forall^{st} P_1 \forall^{st} P_2 : C$, as detailed in the online appendix.

(Case $\square_1 \neq \square_2$). Deciding the coherence of plain alternating 2-ASP(Q) programs Σ_2^P-complete if $\square_1 = \exists^{st}$; otherwise it is Π_2^P-complete (Amendola *et al.* 2019). Since this is a particular case of 2-ASP^w(Q) (i.e., $\mathcal{W}(P_1) = \mathcal{W}(P_2) = \emptyset$), then the thesis follows. □

Theorem 3 (Completeness).
The coherence problem for 2-ASP^w(Q) programs of the form $\square_1 P_1 \square_2 P_2 : C$ is: Σ_2^P-complete if $\square_1 = \exists^{st}$; Π_2^P-complete otherwise, no matter whether $\square_2 = \exists^{st}$ or $\square_2 = \forall^{st}$.

Proof.
The thesis follows from Theorems 1 to 2. □

From Theorem 3, we derive complete results for the brave reasoning task in 2-ASP^w(Q), which asks whether an atom a is true in some optimal quantified answer set.

Theorem 4.
Let Π be an existential 2-ASP^w(Q) program of the form $\square_1 P_1 \square_2 P_2 : C : C^\omega$ and a be a ground atom, then verifying whether $a \in M$, with $M \in OptQAS(\Pi)$, is Δ_3^P-complete no matter whether $\square_2 = \exists^{st}$ or $\square_2 = \forall^{st}$.

These results strengthen those of Mazzotta *et al.* (2024), as they establish completeness for all combinations of quantifiers, rather than for alternating programs only.

4 Solving 2-ASP^w(Q) via CEGAR

CEGAR (Clarke *et al.* 2003) is an iterative technique that starts from a simplified system representation (abstraction) and progressively refines it using counterexamples, until a solution is found or non-existence is proven. CEGAR-based techniques were successfully applied to QBF-satisfiability (Janota *et al.* 2016) and 2-ASP(Q) solving (Cuteri *et al.* 2026). Building on these ideas, we propose a CEGAR-based approach for the evaluation of 2-ASP^w(Q) programs. In the following we give a game-theoretic characterization of the semantics of 2-ASP^w(Q) and then we focus on the two stages of CEGAR: *abstraction refinement* and *counterexample search*.

4.1 The 2-ASP^w(Q) game

Let $\square \in \{\exists^{st}, \forall^{st}\}$ be a quantifier, then the *opponent* of $\square$ is $\overline{\square} = \forall^{st}$ if $\square = \exists^{st}$; otherwise $\overline{\square} = \exists^{st}$. Thus, for any alternating 2-ASP^w(Q) program $\square_2$ is the opponent of $\square_1$, that is, $\square_2 = \overline{\square_1}$. In what follows, w.l.o.g., we consider 2-ASP^w(Q) programs of the form:

$$\square P_1 \overline{\square} P_2 : C : C^\omega \tag{1}$$

Let Π be a 2-ASP^w(Q) program of the form (1), then a *move* for $\square$ is an answer set $M_1 \in OptAS(P_1)$. Given a move M_1 for $\square$, for each $M_2 \in OptAS(P_2 \cup fix_{P_1}(M_1))$, $M_2|_{\mathcal{H}(P_2)}$ is a *candidate countermove* to M_1 for $\overline{\square}$. Let $M_2 \in OptAS(P_2 \cup fix_{P_1}(M_1))$ be a candidate countermove then $M_2|_{\mathcal{H}(P_2)}$ is effectively a *countermove* to M_1 for $\overline{\square}$ if: (i) $\square = \exists^{st}$ and M_2 does not satisfies C; or (ii) $\square = \forall^{st}$ and M_2 satisfies C. A *winning move* for $\square$ is a

move M_1 such that no countermove to M_1 for $\overline{\square}$ exists. Thus, $\square = \exists^{st}$ wins if there exists $M_1 \in OptAS(P_1)$ such that no countermove to M_1 for $\overline{\square}$ exists.

Proposition 1.
Let Π be a 2-ASP^w(Q) of the form (1), there exists a winning move for $\square$ if and only if (i) $\square = \exists^{st}$ and Π is coherent, or (ii) $\square = \forall^{st}$ and Π is incoherent.

Intuitively, the notion of winning moves closely corresponds to the definition of coherence of 2-ASP^w(Q) programs. Let Π be a 2-ASP^w(Q) program of the form (1), and let M_1 be a winning move for $\square$. If $\square = \exists^{st}$, then there is no $M_2 \in OptAS(P_2 \cup fix_{P_1}(M_1))$ such that M_2 violates C. This implies that $\forall^{st} P_2 \cup fix_{P_1}(M_1) : C$ is coherent, and hence Π is coherent. Conversely, if $\square = \forall^{st}$, then there is no $M_2 \in OptAS(P_2 \cup fix_{P_1}(M_1))$ such that M_2 satisfies C. Therefore, $\exists^{st} P_2 \cup fix_{P_1}(M_1) : C$ is incoherent, and thus Π is incoherent.

4.2 Counterexample search in 2-ASP^w(Q)

Given a 2-ASP^w(Q) program Π of the form (1), from Proposition 1, the coherence of Π can be decided by searching for a winning move for $\square$. Thus, the program P_1, whose optimal answer sets coincide with the possible moves of $\square$, is a natural abstraction for Π. Hence, given $M_1 \in OptAS(P_1)$, the counterexample search aims at contradicting M_1, which means finding a countermove to M_1 for $\overline{\square}$. To this end, we define the *countermove program* whose optimal answer sets correspond to countermoves to M_1 for $\overline{\square}$.

We recall that countermoves correspond to optimal answer sets of $P_2 \cup fix_{P_1}(M_1)$ that either satisfy or not the program C according to $\overline{\square}$. Since C is stratified with hard constraints, it admits one answer set (Dantsin *et al.* 2001) iff hard constraints are satisfied. Thus, the following transformation can be used to capture the incoherence of C.

Definition 1 (Complement of stratified program).
Let P be a stratified ASP program with hard constraints, then the *complement* of P, denoted by $\neg P$, is obtained from P by (i) transforming hard constraints $\leftarrow l_1, \ldots, l_n$ into rules of the form $v \leftarrow l_1, \ldots, l_n$; and (ii) adding an hard constraint of the form $\leftarrow \sim v$.

Proposition 2.
Given a stratified ASP program P, its complement $\neg P$ is coherent iff P is incoherent.

At this point, if $\square = \exists^{st}$ (resp. $\square = \forall^{st}$) one might be tempted to define the counterexample program as $P_2 \cup \neg C \cup fix_{P_1}(M_1)$ (resp. $P_2 \cup C \cup fix_{P_1}(M_1)$). However, an optimal answer set of such a counterexample program may correspond to a non-optimal answer set of $P_2 \cup fix_{P_1}(M_1)$ that violates (resp. satisfies) C, which does not correspond to a countermove for $\overline{\square}$ to M_1. To address this issue, we need to relax hard constraints from C into weak constraints with a lower priority level w.r.t. weak constraints in P_2.

Definition 2 (Relaxed program).
Let P be a stratified program with hard constraints, l be an integer, and *unsat* be a fresh atom not appearing anywhere else. Then $relaxed(P, l)$ is defined as:

$$relaxed(P,l) = \left\{ \begin{array}{lll} H_r \leftarrow B_r. & & \forall r \in P s.t. H_r \neq \emptyset \\ unsat \leftarrow B_r. & & \forall r \in P s.t. H_r = \emptyset \\ \leftarrow^w unsat.[1@l] & & \end{array} \right\}$$

Intuitively, $relaxed(\cdot,\cdot)$ is similar to $\neg C$ but the fresh atom introduced as head of the hard constraints of P is used to assign a penalty if P is incoherent. As a result, $relaxed(\cdot,\cdot)$ has the following property, which is fundamental in the computation of countermoves.

Proposition 3.
Given a stratified ASP program P with hard constraints, and an integer l, $relaxed(P,l)$ is always coherent.

Thanks to the above property, the program $relaxed(\neg C,l)$ (resp. $relaxed(C,l)$) can be combined with P_2 in such a way that rules in $\neg C$ (resp. C) do not filter out any optimal answer set of $P_2 \cup fix_{P_1}(M_1)$. Thus, we are now ready to define the countermove program.

Definition 3 (Countermove program).
Let Π be a 2-ASP^w(Q) of the form (1) and $l_{\min}$ be the smallest priority level among weak constraints in P_2, then the *countermove program* for Π is $ctr(\Pi) = P_2 \cup relaxed(\neg C, l_{\min}-1)$ if $\square = \exists^{st}$; otherwise $ctr(\Pi) = P_2 \cup relaxed(C, l_{\min}-1)$.

Note that, the above definition aligns with the countermove definition from Section 4.1. Specifically, when $\square = \exists^{st}$, a countermove must violate C, so the countermove program incorporates rules from $relaxed(\neg C, l_{\min}-1)$. Conversely, when $\square = \forall^{st}$, a countermove must satisfy C, so the countermove program incorporates rules from $relaxed(C, l_{\min}-1)$. Consequently, countermoves to a move for $\square$ are given by the optimal answer sets of the counterexample program.

Proposition 4.
Let Π be a 2-ASP^w(Q) program of the form (1) and $M_1 \in OptAS(P_1)$ be a move for $\square$. There exists $M_2 \in OptAS(ctr(\Pi) \cup fix_{P_1}(M_1))$ such that $unsat \notin M_2$ if and only if $M_2|_{\mathcal{H}(P_2)}$ is a countermove to M_1 for $\overline{\square}$.

4.3 Refining abstractions in 2-ASP^w(Q)

The next step in designing our CEGAR-based approach for 2-ASP^w(Q) is to define a transformation for refining the abstraction according to known countermoves. Specifically, let M_1 be a move for $\square$ and M_2 be a countermove for $\overline{\square}$, then the goal of the refinement, namely $Ref(\Pi, M_2)$, is to obtain a set of rules such that optimal answer sets of $P_1 \cup Ref(\Pi, M_2)$ correspond to moves M_1' for $\square$ such that M_2 is not a countermove to M_1' for $\overline{\square}$. Note that, M_2 is not a countermove to M_1' for $\overline{\square}$ if one of the these condition holds:

1. there is no $M_2' \in AS(P_2 \cup fix_{P_1}(M_1'))$ such that $M_2'|_{\mathcal{H}(P_2)} = M_2$;
2. there exists $M_2' \in AS(P_2 \cup fix_{P_1}(M_1'))$ such that $M_2'|_{\mathcal{H}(P_2)} = M_2$ but $M_2' \notin OptAS(P_2 \cup fix_{P_1}(M_1'))$ (i.e., M_2' is dominated by some answer set of $P_2 \cup fix_{P_1}(M_1')$);
3. there exists $M_2' \in OptAS(P_2 \cup fix_{P_1}(M_1'))$ such that $M_2'|_{\mathcal{H}(P_2)} = M_2$ but M_2' satisfies C and $\square = \exists^{st}$ (resp. violates C and $\square = \forall^{st}$).

Thus, we define some transformations encoding these conditions to refine our abstraction.

First of all, we need a predicate substitution function that maps each predicate p to a fresh one of the form p^{α}_{β}, where α and β are strings, to obtain rules over a fresh signature for each discovered countermove. More in detail, let L be a set of literals and ϵ be an ASP expression (i.e., rule, program, etc.), then $\sigma^{\alpha}_{\beta}(L, \epsilon)$ denotes the expression obtained from ϵ by mapping each positive (resp. negative) literal $p(\vec{t}) \in L$ (resp. $\sim p(\vec{t}) \in L$) with $p^{\alpha}_{\beta}(\vec{t})$ (resp. $\sim p^{\alpha}_{\beta}(\vec{t})$). Now, we are ready to define the rules which verify Condition 1.

Definition 4 (Check Answer Set).
Let P be an ASP program and M be an interpretation, then $checkAS(P, M)$ is:

$$checkAS(P,M) = \begin{cases} \sigma^{-}_{M}(\mathcal{H}(P), \{a \leftarrow | a \in M\}) & \\ \sigma^{+}_{M}(\mathcal{H}(P), \sigma^{-}_{M}(\overline{\mathcal{H}(P)}, r)) & \forall r \in P, H(r) \neq \emptyset \\ fail_M \leftarrow \sigma^{+}_{M}(\mathcal{H}(P), \sigma^{-}_{M}(\overline{\mathcal{H}(P)}, B(r))) & \forall r \in P, H(r) = \emptyset \\ fail_M \leftarrow p(\vec{t})^{+}_{M}, \sim p(\vec{t})^{-}_{M} & \forall\ p(\vec{t}) \in \mathcal{H}(P) \\ fail_M \leftarrow p(\vec{t})^{-}_{M}, \sim p(\vec{t})^{+}_{M} & \forall\ p(\vec{t}) \in \mathcal{H}(P) \\ as_M \leftarrow \sim fail_M & \end{cases}$$

Basically, $checkAS(P, M)$ encodes the GL-reduct of a program P w.r.t. an interpretation M and checks whether M is a $\subset$-minimal model of the reduct. Specifically, M is encoded as facts over a fresh *negative signature* (i.e., each atom $p(\vec{t}) \in M$ is represented by a fact $p^{-}_{M}(\vec{t}) \leftarrow$). Atoms over the negative signature are used to rewrite negative body literals, yielding the GL-reduct of P w.r.t. M; whereas head atoms and positive body literals are mapped to a fresh *positive signature* (i.e., $p(\vec{t})$ mapped to $p^{+}_{M}(\vec{t})$). Finally, if the true atoms over positive and negative signatures coincide then M is an answer set of P.

We can now focus on Condition 2 which requires checking answer set optimality. To this end, we define a transformation to compute the cost of answer sets of an ASP program.

Definition 5 (Cost Program).
Let P be an ASP program, L be the set of priority levels in $\mathcal{W}(P)$, and v_P and cl_P are fresh predicates not appearing in P. Then, $cost(P)$ is defined as:

$$cost(P) = \begin{cases} v_P(w, l, \vec{t}) \leftarrow l_1, \ldots, l_n & \forall \leftarrow^{\omega} l_1, \ldots, l_n[w@l, \vec{t}\,] \in P \\ cl_P(T, l) \quad \leftarrow \#sum\{C, \vec{t} : v_P(C, l, \vec{t})\} = T & \forall l \in L \end{cases}$$

Intuitively, weak constraints of the form $\leftarrow^{\omega} l_1, \ldots, l_n[w@l, \vec{t}\,]$ can be rewritten into rules whose head is a fresh atom encoding violation tuples (i.e., $(w, l, \vec{t})$) and introduces a rule for each level l that sums up the the weights of the violation tuples at level l. To verify the optimality of an answer set $M \in AS(P)$, it is necessary to compare its cost with that of every $M' \in AS(P)$. To this end, we use $clone(P) = \sigma_{clone}(\mathcal{H}(P) \cup \overline{\mathcal{H}(P)}, P)$, which clones the program P, enabling the comparison between the cost of M and the cost of all answer sets of the cloned program.

Definition 6 (Dominated program).
Let P_1 and P_2 be two ASP programs, then $checkDom(P_1, P_2)$ is the following program:

$$\begin{array}{lll} diff_{P_1,P_2}(L) & \leftarrow & cl_{P_1}(C1, L), cl_{P_2}(C2, L), C1 \neq C2. \\ hasHigher_{P_1,P_2}(L) & \leftarrow & diff_{P_1,P_2}(L), diff_{P_1,P_2}(L1), L < L1. \\ highest_{P_1,P_2}(L) & \leftarrow & diff_{P_1,P_2}(L), \sim hasHigher_{P_1,P_2}(L). \\ dom_{P_1,P_2} & \leftarrow & highest_{P_1,P_2}(L), cl_{P_1}(C1, L), cl_{P_2}(C2, L), C2 < C1 \end{array}$$

where cl_{P_1} and cl_{P_2} are, respectively, the predicates introduced by $cost(P_1)$ and $cost(P_2)$; whereas $diff_{P_1,P_2}$, $hasHigher_{P_1,P_2}$, $highest_{P_1,P_2}$, and dom_{P_1,P_2} are fresh predicates.

Intuitively, let $M \in AS(P)$ and $M_c \in AS(clone(P))$, then $checkDom(P, clone(P))$ can be used to compare the cost of M and M_c. More precisely, $cost(P)$ and $cost(clone(P))$ compute, respectively, the cost of M and M_c, for each level l, as atoms of the form $cl_P(C, l)$ and $cl_{clone(P)}(C, l)$. Thus, rules from $dom(P, clone(P))$ derive $dom_{P,clone(P)}$ if and only if M is dominated by M_c, which means M is not an optimal answer set of P.

Finally, we can focus on Condition 3. Recall that Condition 3 requires to verify the coherence (resp. incoherence) of the program C. To this end, it is possible to leverage $relaxed(C, l)$ (resp. $relaxed(\neg C, l)$), as we have seen for counterexample program.

Now that, we have transformations for each condition (i.e., Conditions 1, 2, and 3), we need to control the activation of the different transformations in the refinement process.

Definition 7 (Controlled program).
Let P be a program and l a literal s.t. $at(l) \notin B_P$, then $or(P, l) = \{H_r \leftarrow B_r, l \mid r \in P\}$.

Intuitively, $or(P, l)$ controls the activation of P according to the literal l. If l is true, it can be removed from rules in $or(P, l)$, obtaining back the program P; otherwise rules in $or(P, l)$ are satisfied as l falsifies rules' body. We can now define the refinement program.

Definition 8 (Refinement Program).
Let Π be a 2-ASP^w(Q) program of the form (1), M be a move for $\square$ and let CE be a countermove to M for $\overline{\square}$. The *refinement program* is defined as follows:

$$ref(\Pi, CE) = \left\{ \begin{array}{l} checkAS(P_2, CE) \\ \leftarrow^w as_{CE}\ [1@l_{\min} - 1] \\ or(cost(P_2^{CE}), as_{CE}) \\ or(clone(\mathcal{R}(P_2)), as_{CE}) \\ or(cost(clone(P_2)), as_{CE}) \\ or(checkDom(P_2^{CE}, clone(P_2)), as_{CE}) \\ \leftarrow^w as_{CE}, {\sim}dom_{P_2^{CE},clone(P_2)}\ [1@l_{\min} - 2] \\ or(\sigma^-_{CE}(lits(P_2) \cup lits(C'), C'), as_{CE}) \end{array} \right\}$$

where $P_2^{CE} = \sigma^-_{CE}(lits(P_2), P_2)$, $lits(P_2) = \mathcal{H}(P_2) \cup \overline{\mathcal{H}(P_2)}$, as_{CE} *is the predicate introduced by* $checkAS(P_2, CE)$, $C' = relaxed(C, l_{\min} - 3)$ *if* $\square = \exists^{st}$; *otherwise* $C' = relaxed(\neg C, l_{\min} - 3)$, $lits(C') = \mathcal{H}(C') \cup \overline{\mathcal{H}(C')}$, *and* $l_{\min}$ *is the smallest priority level among weak constraints of* P_1.

Let Π be a 2-ASP^w(Q) of the form (1), M be a move for $\square$ and CE be a countermove to M for $\overline{\square}$, then we aim at computing a new move M' for $\square$ such that CE is not a countermove to M' for $\overline{\square}$. To this end, we use $ref(\Pi, CE)$ to check Conditions 1–3. More precisely, rules in $checkAS(P_2, CE)$ encode the reduct of P_2 w.r.t. CE and derive the atom as_{CE} iff there exists $M_2 \in AS(P_2 \cup fix_{P_1}(M'))$ such that $M_2|_{\mathcal{H}(P_2)} = CE$ (i.e., Condition 1 is not satisfied). Thus, the weak constraint $\leftarrow^\omega as_{CE}\ [1@l_{\min} - 1]$ expresses the preference to satisfy Condition 1. If Condition 1 is not satisfied, then there exists $M_2 \in AS(P_2 \cup fix_{P_1}(M'))$ such that $M_2|_{\mathcal{H}(P_2)} = CE$ and so as_{CE} is derived as true. At this point, as_{CE} activates the following blocks which check Conditions 2 and 3.

To check Condition 2, $or(cost(P_2^{CE}), as_{CE})$ computes the cost of M_2 w.r.t. weak constraints in P_2, $or(clone(\mathcal{R}(P_2)), as_{CE})$ clones the program P_2 to compare pair of answer set of $P_2 \cup fix_{P_1}(M')$, $or(cost(clone(P_2)), as_{CE})$ computes the cost of an answer set of the cloned program, and finally $or(checkDom(P_2^{CE}, clone(P_2)), as_{CE})$ derive the atom $dom_{P_2,clone(P_2)}$ iff M_2 is not optimal as M_2 is dominated by some answer set of the cloned program. As a result, the weak constraint $\leftarrow^w as_{CE}, {\sim}dom_{P_2^{CE},clone(P_2)}$ $[1@l_{\min} - 2]$ prefers answer sets of the cloned program that dominate M_2 (i.e., M_2 is not optimal). If M_2 cannot be dominated by any answer set of the cloned program then M_2 is optimal and so $or(\sigma^-_{CE}(lits(P_2) \cup lits(C'), C'), as_{CE})$ checks Condition 3. In particular, if $\square = \exists^{st}$ (resp. $\square = \forall^{st}$), then the atom $unsat$ is derived iff M_2 does not satisfy C (resp. $\neg C$), which means Condition 3 is not satisfied. Thus, the weak constraint $\leftarrow^{\omega} unsat, as_{CE}$ $[1@l_{\min} - 3]$ in C' models the preference of satisfying C (resp. $\neg C$) (i.e., Condition 3 is satisfied). Thus, if there exists an optimal answer set of $P_1 \cup ref(\Pi, CE)$ that satisfies at least one weak constraint in $ref(\Pi, CE)$ then this corresponds to a move M' that does not admit CE as countermove. A detailed example is provided in the online appendix.

5 Computing optimal quantified answer sets

Computing optimal (quantified) answer sets requires algorithms that make multiple oracle calls to search for such answer sets (Alviano *et al*. 2020; Azzolini *et al*. 2026).

5.1. Quantified answer set computation

The computation of quantified answer sets in presence of weak constraints can be obtained by plugging the transformations defined in previous section in the *CEGAR for 2-ASP(Q)* algorithm by Cuteri *et al.* (2026). In particular, we update both counterexample search and refinement procedure with those introduced in Section 4.2 (i.e., Definitions 3 and 8) and update the conditions of existence of a winning move accordingly. Indeed, in the original algorithm it was sufficient to check for existence of an answer set of the refined or counterexample programs, whereas here we have to look at the cost of their optimal answer sets. More in detail, at the least three levels the cost 1 if no winning exists; whereas for the counterexample program at the last level the cost is 1 if the current move is winning (full algorithm in the online appendix).

5.2. Upper-bound improving

Optimal quantified answer sets can be computed starting from a quantified answer set and iteratively searching for better ones until the incoherence is met (Alviano *et al*. 2020). A quantified answer set M of the input program Π is computed applying CEGAR for 2-ASP^w(Q), and the cost of M is our initial upper bound. Note that if M does not exist, no optimal quantified answer set exists and the search stops immediately. Next, to improve on this bound a constraint is added to Π to enforce a preference for answer sets of P_1 (i.e., candidate quantified answer sets) with a lower cost, and the solver is called again. The process repeats until the upper bound cannot be improved anymore, and the last quantified answer set is optimal.

5.3. Lower-bound improving

The core idea of these strategies is to fix an initial lower-bound cost and, if no answer set within this bound exists, iteratively relax the program to raise the bound until an optimal solution is obtained. Traditional ASP solvers start by treating all weak constraints as hard, aiming for a zero-cost lower bound. If no answer set exists, unsatisfiable cores guide the progressive relaxation of the program, incrementally raising the lower bound (roughly admitting some weak constraint must be violated) until an optimal solution is found (Alviano *et al.* 2020). Unluckily, this strategy cannot be ported as it is in our setting, since a notion of unsatisfiable core has never been defined for ASP^w(Q). Thus, we propose alternative ways for targeting a lower bound and also for relaxing the program so that the lower bound improves iteratively until the optimum is found. Intuitively, we aim to compute an answer set of P_1 that is optimal with respect to the global weak constraints. This is achieved by adding the global weak constraints C^ω to P_1 with the lowest priority and searching for the optimum answer sets of the resulting program P_1^{lower}. Note that, an optimum answer set of P_1^{lower} is a reasonable lower-bound candidate, since it is either an optimum for Π or it does not satisfy the subsequent quantifiers. In the latter case, following the CEGAR approach, an oracle call is used to compute a countermove, and the corresponding refinement is added to P_1^{lower}. This enables the next iteration to search for a new candidate optimal answer set. The procedure repeats until a winning move is found, incrementally improving the lower bound by discarding candidates that are not quantified answer sets of Π. To ensure the correctness of the approach the global weak constraints W are moved at the lowest priority levels w.r.t. both local weak constraints in P_1 and weak constraints that would be added to P_1^{lower} by the refinement procedure. More precisely, the initial abstraction is defined as $P_1 \cup W$, where $W = \{\leftarrow^\omega l_1, \ldots, l_k\ [w@\lambda + l, \vec{t}\,] \mid \leftarrow^\omega l_1, \ldots, l_k\ [w@l, \vec{t}\,] \in C^\omega\}$, with λ being an integer such that levels from C^ω are remapped to strictly lower priority levels w.r.t. $l_{\min}-3$ (i.e., the smallest priority level added by the refinement), and $l_{\min}$ being the lowest priority level in P_1. In this way, the first winning move will be a quantified answer set which is also optimal. Note that the proposed ordering of priority levels in the (refined) abstraction is essential: it first favors moves that admit no known countermove and only then prefers moves that are optimal with respect to C^ω. Under this ordering, the first winning move obtained is an optimal quantified answer set.

6 Implementation and experiments

We run an experimental campaign on an Intel(R) Xeon(R) CPU E7-8880 v4 @ 2.20 GHz, running Debian GNU/Linux 12, with memory and CPU (i.e., user+system) limited to 8 GB and 800s. Benchmarks and executables are available at https://osf.io/gmnjx

6.1. Implementation

The proposed approach was implemented on top of the CASPER (Cuteri *et al.* 2026). More in detail, CASPER is written in Python and uses CLINGO (Gebser *et al.* 2016) as an oracle for computing moves and countermoves. For non-alternating 2-ASP^w(Q)

programs, our implementation applies the transformations proposed by Mazzotta *et al.* (2024) for removing weak constraints, as the 2-ASP^w(Q) game is defined for alternating programs. Specifically, non-alternating 2-ASP^w(Q) programs are translated into 2-ASP(Q) program with two alternating quantifiers and no local weak constraints.

6.2. Benchmarks

We considered several benchmarks from diverse ASP(Q) applications (Faber *et al.* 2023; Azzolini *et al.* 2025, 2026): Propositional Abduction Problem (PAP) (Eiter and Gottlob 1995), Minmax Clique (MMC) (Cao *et al.* 1995), Max Term Deletion (MTD) (Schaefer and Umans 2002), Most Probable Explanation in Probabilistic ASP (MPE) (Azzolini et al. 2025), and Clique Coloring (CC) (Schaefer and Umans 2002). For PAP, we consider three reasoning tasks: PAP-OPT, PAP-REL, and PAP-NEC, corresponding to computing cardinality-minimal solutions, and relevant and necessary hypotheses, respectively. For MMC we considered both the decision and optimization variant of the problem, namely MMC-BOUND and MMC-OPT. For MTD, we considered the optimization version. Finally, for MPE we considered both COLORING and SMOKERS domains (Azzolini *et al.* 2025). For the CC benchmark, we considered graphs of varying size (from 10 to 120 nodes) and edge density (25%, 50%, and 75%), generated according to the Erdős–Rényi model provided by NetworkX library (https://pypi.org/project/networkx). For each combination of size and density, we generated 10 instances. For the other benchmarks, instances were drawn from previous experiments (Azzolini *et al.* 2025, 2025; Cuteri *et al.* 2026).

6.3. Compared methods

In our evaluation we compared the approach by Azzolini *et al.* (2026) implemented in PYQASP. Note that, PYQASP implements the upper-bound improving algorithm on top of a rewriting in QBF and it uses QUABS (https://github.com/ltentrup/quabs) as backend solver. Then, we consider also the proposed techniques denoted as CASPER-U (upper-bound improving) and CASPER-L (lower-bound improving).

6.4. Results

Obtained results are summarized in Table 1 which reports, for each benchmark, the number of instances (#inst), the type of task (TT) (*opt* for optimal answer set and *coh* for coherence), and, for each system, the number of solved instances (Sol.), total execution time (Sum t,(s)), and number of optimization steps (#Opt.). This latter is the sum of the number of quantified answer sets found to reach the solution; it is omitted for lower-bound improving and is not meaningful for local-weak-constraints-only problems.

We first focus on the benchmarks containing only local weak constraints, namely: PAP-NEC, PAP-REL, and MMC-BOUND. Observe that, in these cases, PYQASP could not be run (it does not support local weak constraints), and CASPER-L and CASPER-U clearly coincide, since the first winning move is the solution. For these benchmarks, nearly all instances were solved by CASPER within 2 min (only 12 of 813 timed out), confirming effectiveness of the systems herein introduced. To further assess the scalability

Table 1. *Overall results*

Bench	#inst	TT	CASPER-L		CASPER-U			PYQASP		
			Sol.	Sum t.(s)	Sol.	Sum t.(s)	#opt	Sol.	Sum t.(s)	#opt
PAP-OPT	294	*opt*	**123**	**6450.78**	33	9725.87	367	101	18076.72	410
MTD	80	*opt*	50	1698.24	**74**	**3916.51**	1184	48	6451.46	253
COLORING	94	*opt*	11	896.93	8	114.35	105	**25**	**3621.92**	278
SMOKERS	99	*opt*	**99**	**83.54**	19	3130.97	3724	14	1845.59	152
MMC-OPT	45	*opt*	**44**	**3565.79**	42	2912.66	106	5	904.65	5
PAP-NEC	294	*coh*	**282**	**2675.68**	**282**	**2675.68**	–	–	–	–
PAP-REL	294	*coh*	**294**	**2053.67**	**294**	**2053.67**	–	–	–	–
MMC-BOUND	225	*coh*	**225**	**919.39**	**225**	**919.39**	–	–	–	–
CC	420	*coh*	**370**	**5555.22**	**370**	**5555.22**	–	–	–	–

Boldface is used to highlight the best performing approach.

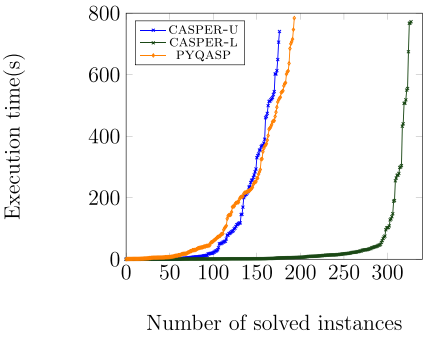

Fig 1. Overall execution time – *opt*.

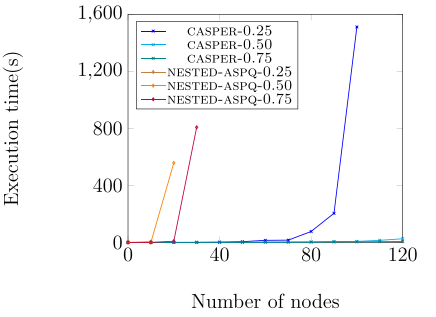

Fig 2. Solving time for CC.

of CASPER, we considered the CC benchmark. In particular, we compare CASPER with an enumeration-based solver, denoted as NESTED-ASPQ, which evaluates programs by enumerating answer sets of each subprogram according to the 2-ASP^w(Q) semantics. Figure 2 reports the execution times of the systems for graphs with fixed edge density. Instances are sorted by increasing graph size, and each point (x, y) represents the average runtime y (over 10 generated instances) required by a system on graphs with x nodes and a given edge density.

The generated instances are hard to solve and NESTED-ASPQ is unable to scale beyond graphs with 30 nodes, regardless of the edge density. In contrast, CASPER exhibits significantly better scalability, solving instances with up to 120 nodes. Furthermore, the edge density has a noticeable impact on CASPER runtime. For dense graphs, the runtime is considerably small, as dense graphs admits *few* maximal cliques and so the number of possible counterexample is reduced. Conversely, for sparse graphs, the runtime increases, due to the larger number of maximal cliques, which in turn leads to a higher number of counterexamples to be explored.

Let's now shift the attention to optimum quantified answer set search. For MTD and COLORING, the upper-bound improving strategy, implemented in PYQASP and CASPER-U, is preferred over the lower-bound improving strategy adopted by CASPER-L. The ASP-based CASPER-U is preferable to the QBF-based PYQASP in MTD, whereas the opposite holds for COLORING. Diving in the details, PYQASP could compute for COLORING quantified answer sets that are closer to the optimum so requiring few optimization steps; on the other hand, CASPER-U shows a faster computation of quantified answer sets leading to better performance. On the other hand, CASPER-L likely computes many locally optimal moves that are not quantified answer sets, thus resulting slower than upper-bound improving alternatives. Conversely, for PAP-OPT, MMC-OPT, and SMOKERS, CASPER-L solves substantially more instances than the others (146 more than PYQASP and 172 more than CASPER-U), since locally optimal moves frequently correspond to optimal solutions.

Overall, CASPER-L solves 134 more instances than PYQASP, and 151 more instances than CASPER-U with a lower average runtime. Figure 1 reports a traditional cactus plot on all the instances, confirming CASPER-L is the most effective system overall.

7 Related work

Many ASP extensions have been proposed for modeling hard combinatorial optimization problems. The standard ASP construct for expressing optimization problems is weak constraints (Buccafurri *et al*. 2000), which is equivalent to optimize statements (Gebser *et al*. 2012). The ASP^w(Q) language is based on the same construct but expands the modeling capabilities of ASP in the entire PH. For alternative formalism to ASP(Q), such as stable-unstable (Bogaerts *et al*. 2016) and quantified ASP (Fandinno *et al*. 2021), we are not aware of any extension considering optimization statements explicitly. Optimization in ASP might also be handled within the *asprin* framework (Brewka *et al*. 2023), which however targets preference modeling. An in-depth comparison of ASP(Q) with these formalism was provided by Amendola *et al*. (2019) and Fandinno *et al*. (2021).

Among ASP(Q) systems, we mention QASP (Amendola *et al.* 2022), PYQASP (Faber *et al*. 2023), and CASPER (Cuteri *et al.* 2026). The first two are based on a translation of ASP(Q) in QBF, they support an arbitrary number of quantifiers, but originally lacked support for weak constraints. Recently, PYQASP was extended to handle global weak constraints using an upper-bound improving strategy (Azzolini *et al*. 2026), though local weak constraints remain unsupported. CASPER (Cuteri *et al*. 2026) is the only CEGAR-based ASP(Q) system and originally supported only 2-ASP(Q) programs without weak constraints. This paper extends CASPER to support local and global weak constraints,

yielding the first implementation capable of evaluating 2-ASP^w(Q) programs. Solvers for ASP^w(Q) exploit optimization strategies used in ASP solvers (Alviano *et al.* 2020), however our lower-bound improving approach departs from existing methods by realizing improvements via abstraction refinement.

8 Conclusion

This paper focuses on 2-ASP^w(Q), the class of ASP^w(Q) programs with two quantifiers, for which tight complexity bounds and concrete implementations were previously missing. We fill this gap by providing a detailed complexity analysis and establishing tight completeness results: coherence checking is complete for the second level of the PH (i.e., Σ_2^P for existential programs and Π_2^P for universal ones), while reasoning over optimal quantified answer sets is Δ_3^P-complete. Moreover, building on the CEGAR framework, we developed two optimization techniques based on lower- and upper-bound improvement strategies. Experimental results demonstrate the effectiveness of our approach, advancing the state of the art in 2-ASP^w(Q) solving.

Future work includes extending both the complexity analysis and evaluation techniques to ASP^w(Q) programs with an arbitrary number of quantifiers.

Supplementary material

To view supplementary material for this article, please visit https://doi.org/10.1017/S1471068426100477.

References

Alviano, M., Dodaro, C., Marques-Silva, J. and Ricca, F. 2020. Optimum stable model search: Algorithms and implementation. *Journal of Logic and Computation* 30, 4, 863–897.

Amendola, G., Cuteri, B., Ricca, F. and Truszczynski, M. 2022. Solving problems in the polynomial hierarchy with ASP(Q). In *LPNMR*. LNCS, vol. 13416, 373–386.

Amendola, G., Ricca, F. and Truszczynski, M. 2019. Beyond NP: Quantifying over answer sets. *TPLP* 19, 5–6, 705–721.

Azzolini, D., Leone, N., Mazzotta, G. and Ricca, F. 2026. Solving hard combinatorial optimization problems with PyQASP. In *Practical Aspects of Declarative Languages - 28th International Symposium (PADL)*, Rennes, France, 12–13 Jan., 2026, N. Amin and J. Arias, Eds. LNCS, vol. 16401. Springer, 199–217.

Azzolini, D., Mazzotta, G., Ricca, F. and Riguzzi, F. 2025. Most probable explanation in probabilistic answer set programming. In *Proceedings of IJCAI 2025*, 9049–9057.

Bellusci, P., Mazzotta, G. and Ricca, F. 2022. Modelling the outlier detection problem in ASP(Q). In *PADL*. LNCS, vol. 13165. Springer, 15–23.

Bogaerts, B., Janhunen, T. and Tasharrofi, S. 2016. Stable-unstable semantics: Beyond NP with normal logic programs. *TPLP* 16, 5–6, 570–586.

Brewka, G., Delgrande, J. P., Romero, J. and Schaub, T. 2023. A general framework for preferences in answer set programming. *Artificial Intelligence* 325, 104023.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Buccafurri, F., Leone, N. and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *TKDE* 12, 5, 845–860.

Cao, F., Du, D.-Z., Gao, B., Wan, P.-J. and Pardalos, P. M. 1995. Minimax problems in combinatorial optimization. In *Minimax and Applications*, Springer, Boston, MA, 269–292.

Cardellini, M., Nardi, P. D., Dodaro, C., Galatà, G., Giardini, A., Maratea, M. and Porro, I. 2021. A two-phase ASP encoding for solving rehabilitation scheduling. In *RuleML+RR*. LNCS, vol. 12851. Springer, 111–125.

Ceri, S., Gottlob, G. and Tanca, L. 1990. *Logic Programming and Databases*. Surveys in Computer Science. Springer.

Chiariello, F., Fionda, V., Ielo, A. and Ricca, F. 2024. A direct ASP encoding for declare. In *PADL*. Lecture Notes in Computer Science. Springer, 116–133.

Clarke, E. M., Fehnker, A., Han, Z., Krogh, B. H., Ouaknine, J., Stursberg, O. and Theobald, M. 2003. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science* 14, 4, 583–604.

Cuteri, A., Mazzotta, G. and Ricca, F. 2026. 2-ASP(Q) solving based on CEGAR. *Proceedings of the AAAI Conference on Artificial Intelligence* 40, 23, 19030–19038.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33, 3, 374–425.

Dodaro, C., Galatà, G., Maratea, M. and Porro, I. 2018. Operating room scheduling via answer set programming. In *AI*IA*. LNCS, vol. 11298. Springer, 445–459.

Eiter, T., Fink, M., Greco, G. and Lembo, D. 2008. Repair localization for query answering from inconsistent databases. *ACM Transactions on Database Systems* 33, 2, 10:1–10:51.

Eiter, T. and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM* 42, 1, 3–42.

Faber, W. 2024. Solving argumentation problems using answer set programming with quantifiers: Preliminary report. In *ICLP Workshops. CEUR*, vol. 3799. CEUR-WS.org.

Faber, W., Mazzotta, G. and Ricca, F. 2023. An efficient solver for ASP(Q). *TPLP* 23, 4, 948–964.

Faber, W., Morak, M. and Chrpa, L. 2022. Determining action reversibility in STRIPS using answer set programming with quantifiers. In *PADL*. LNCS, vol. 13165. Springer, 42–56.

Fandinno, J., Laferrière, F., Romero, J., Schaub, T. and Son, T. C. 2021. Planning with incomplete information in quantified answer set programming. *TPLP* 21, 5, 663–679.

Fionda, V., Ielo, A. and Ricca, F. 2024. LTLf2ASP: LTLf bounded satisfiability in ASP. In *LPNMR*. Lecture Notes in Computer Science. Springer, 373–386.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Wanko, P. 2016. Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*. OASIcs, vol. 52. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1. 2:15.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3/4, 365–386.

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

Mazzotta, G., Ricca, F. and Truszczynski, M. 2024. Quantifying over optimum answer sets. *TPLP* 24, 4, 716–736.

Schaefer, M. and Umans, C. 2002. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News* 33,3, 32–49.

Son, T. C., Pontelli, E., Balduccini, M. and Schaub, T. 2023. Answer set planning: A survey. *TPLP* 23,1, 226–298.