TPLP **24** (4): 716–736, 2025.
doi:10.1017/S1471068424000395

Quantifying over Optimum Answer Sets[*]

GIUSEPPE MAZZOTTA and FRANCESCO RICCA
University of Calabria, Arcavacata di Rende, Italy
(*e-mails:* giuseppe.mazzotta@unical.it, francesco.ricca@unical.it)

MIREK TRUSZCZYNSKI
University of Kentucky, Lexington, USA
(*e-mail:* mirek@cs.uky.edu)

submitted 14 August 2024; accepted 10 September 2024

Abstract

Answer Set Programming with Quantifiers (ASP(Q)) has been introduced to provide a natural extension of ASP modeling to problems in the polynomial hierarchy (PH). However, ASP(Q) lacks a method for encoding in an elegant and compact way problems requiring a polynomial number of calls to an oracle in Σ_n^p (that is, problems in Δ_{n+1}^p). Such problems include, in particular, optimization problems. In this paper, we propose an extension of ASP(Q), in which component programs may contain weak constraints. Weak constraints can be used both for expressing local optimization within quantified component programs and for modeling global optimization criteria. We showcase the modeling capabilities of the new formalism through various application scenarios. Further, we study its computational properties obtaining complexity results and unveiling non-obvious characteristics of ASP(Q) programs with weak constraints.

KEYWORDS: design, analysis and implementation of languages, knowledge representation and nonmonotonic reasoning, logic programming methodology and applications

1 Introduction

Answer set programming (ASP) (Gelfond and Lifschitz 1991; Brewka *et al.* 2011) has been proposed over two decades ago as a variant of logic programming for modeling and solving search and optimization problems (Marek and Truszczynski 1999; Niemelä 1999). Today, it is among the most heavily studied declarative programming formalisms with highly effective processing tools and an ever-growing array of applications (Brewka *et al.* 2011, 2016). Focusing on decision problems, the scope of applicability of ASP is that of the class Σ_2^P (Dantsin *et al.* 2001). This class includes a vast majority of problems of practical interest. However, many important decision problems belong to higher complexity classes (Stockmeyer 1976; Schaefer and Umans 2002). For this reason, several language

[*] Partially supported by MISE under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, and MUR under projects: PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006.

extensions have been proposed that expand the expressivity of ASP (Bogaerts *et al.* 2016; Amendola *et al.* 2019; Fandinno *et al.* 2021). Among these, Answer Set Programming with Quantifiers (ASP(Q)) (Amendola *et al.* 2019) has been recently introduced to offer a natural declarative means to model problems in the entire Polynomial Hierarchy (PH).

Roughly speaking, the definition of a problem in Σ_n^P can be often reformulated as "there is an answer set of a program P_1 such that for every answer set of a program P_2, ... there is an answer set of P_n, so that a stratified program with constraint C, modeling admissibility of a solution, is coherent," (and a similar sentence starting with "for all answer set of program P_1" can be used to encode problems Π_n^P).

Both the original paper (Amendola *et al.* 2019) on ASP(Q), and the subsequent one (Amendola *et al.* 2022) presented several examples of problems outside the class Σ_2^P that allow natural representations as ASP(Q) programs. Furthermore, Amendola *et al.* (2022) first, and Faber et al. (2023) later, provided efficient tools for evaluating ASP(Q) specifications providing empirical evidence of practical potential of ASP(Q).

However, ASP(Q) lacks a convenient method for encoding in an elegant way preference and optimization problems (Buccafurri *et al.* 2000; Schaefer and Umans 2002).

In this paper, we address this issue by proposing an extension of ASP(Q) with *weak constraints* or ASP$^\omega$(Q), in short. Weak constraints were introduced in ASP by Buccafurri et al., (2000) to define preferences on answer sets. They are today a standard construct of ASP (Calimeri *et al.* 2020), used to model problems in the class Δ_3^P (i.e., the class of problems that can be solved by a polynomial number of calls to a Σ_2^P oracle). In ASP$^\omega$(Q), weak constraints have dual purposes: expressing local optimization within quantified subprograms and modeling global optimization criteria. Both features increase the modeling efficacy of the language, which we demonstrate through example problems. Further, we investigate the computational properties of ASP(Q) programs with weak constraints and obtain complexity results that reveal some non-obvious characteristics of the new language. Among these, the key positive result states that ASP$^\omega$(Q) programs with n alternating quantifiers can model problems complete for Δ_{n+1}^P.

2 Answer Set Programming

We now recall Answer Set Programming (ASP) (Gelfond and Lifschitz 1991; Brewka *et al.* 2011) and introduce the notation employed in this paper.

2.1 The syntax of ASP

Variables are strings starting with uppercase letters, and constants are non-negative integers or strings starting with lowercase letters. A *term* is either a variable or a constant. A *standard atom* is an expression of the form $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. A standard atom $p(t_1, \ldots, t_n)$ is *ground* if $t_1, \ldots, t_n$ are constants. A *standard literal* is an atom p or its negation $\sim p$. An *aggregate element* is a pair $t_1, \ldots, t_n : conj$, where $t_1, \ldots, t_n$ is a non-empty list of terms, and $conj$ is a non-empty conjunction of standard literals. An *aggregate atom* is an expression $f\{e_1; \ldots; e_n\} \prec T$, where $f \in \{\#count, \#sum\}$ is an *aggregate function symbol*, $\prec \in \{<, \leq, >, \geq, =\}$ is a comparison operator, T is a term called the *guard*, and $e_1, \ldots, e_n$ are aggregate elements. An *atom* is either a standard atom or an aggregate atom. A *literal* is an atom (positive literal) or its negation (resp. negative literal). The *complement* of a literal l is denoted by $\bar{l}$, and it is $\sim a$, if $l = a$, or a, if $l = \sim a$, where a is an atom. For a set of literals L, L^+, and L^- denote the set of positive and negative literals in L, respectively. A *rule* is an expression of the form:

$$h \leftarrow b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_m. \tag{1}$$

where $m \geq k \geq 0$. Here h is a standard atom or is empty, and all b_i with $i \in [1, m]$ are atoms. We call h the *head* and $b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_m$ the *body* of the rule (1). If the head is empty, the rule is a *hard constraint*. If a rule (1) has a non-empty head and $m = 0$, the rule is a *fact*. Let r be a rule, h_r denotes the head of r, and $B_r = B_r^+ \cup B_r^-$ where B_r^+ (resp. B_r^-) is the set of all positive (resp. negative) literals in the body of r.

A *weak constraint* (Buccafurri *et al.* 2000) is an expression of the form:

$$\leftarrow_w b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_m \; [w@l, T], \tag{2}$$

where, $m \geq k \geq 0$, $b_1, \ldots, b_k, b_{k+1}, \ldots, b_m$ are standard atoms, w and l are terms, and $T = t_1, \ldots, t_n$ is a tuple of terms with $n \geq 0$. Given an expression ϵ (atom, rule, weak constraint, etc.), $\mathcal{V}(\epsilon)$ denotes the set of variables appearing in ϵ; $at(\epsilon)$ denotes the set of standard atoms appearing in ϵ; and $\mathcal{P}(\epsilon)$ denotes the set of predicates appearing in ϵ. For a rule r, the *global variables* of r are all those variables appearing in h_r or in some standard literal in B_r or in the guard of some aggregates. A rule r is *safe* if its global variables appear at least in one positive standard literal in B_r, and each variable appearing into an aggregate element e either is global or appears in some positive literal of e (Ceri *et al.* 1990; Faber *et al.* 2011); a weak constraint v of the form (2) is safe if $\mathcal{V}(B_v^-) \subseteq \mathcal{V}(B_v^+)$ and $\mathcal{V}(\{w, l\}) \cup \mathcal{V}(T) \subseteq \mathcal{V}(B_v^+)$. A *program* P is a set of safe rules and safe weak constraints. Given a program P, $\mathcal{R}(P)$ and $\mathcal{W}(P)$ denote the set of rules and weak constraints in P, respectively, and $\mathcal{H}(P)$ denotes the set of atoms appearing as heads of rules in P.

A choice rule (Simons *et al.* 2002) is an expression of the form: $\{e_1; \ldots; e_k\} \leftarrow l_1, \ldots, l_n$, where each choice element e_i is of the form $a^i : b_1^i, \ldots, b_{m_i}^i$, where a^i is a standard atom, $m_i \geq 0$, and $b_1^i, \ldots, b_{m_i}^i$ is a conjunction of standard literals. For simplicity, choice rules can be seen as a shorthand for certain sets of rules. In particular, each choice element e_i corresponds to: $a^i \leftarrow b_1^i, \ldots, b_{m_i}^i, l_1, \ldots, l_n, \;\sim na^i$, $na^i \leftarrow b_1^i, \ldots, b_{m_i}^i, l_1, \ldots, l_n, \;\sim a^i$ where na^i denotes the standard atom obtained from a^i by substituting the predicate of a, say p, with a fresh predicate p' not appearing anywhere else in the program.

2.2 The semantics of ASP

Assume a program P is given. The *Herbrand Universe* is the set of all constants appearing in P (or a singleton set consisting of any constant, if no constants appear in P) and is denoted by HU_P; whereas the *Herbrand Base*, that is the set of possible ground standard atoms obtained from predicates in P and constants in HU_P, is denoted by HB_P. Moreover, $ground(P)$ denotes the set of possible ground rules obtained from rules in P by proper variable substitution with constants in HU_P. An *interpretation* $I \subseteq HB_P$ is a set of standard atoms. A ground standard literal $l = a$ (resp. $l = \sim a$) is true w.r.t. I if $a \in I$ (resp. $a \notin I$), otherwise it is false. A conjunction *conj* of literals is true w.r.t. I if every literal in *conj* is true w.r.t. I, otherwise it is false. Given a ground set of aggregate elements $S = \{e_1; \ldots; e_n\}$, $eval(S, I)$ denotes the set of tuples of the form $(t_1, \ldots, t_m)$ such that there exists an aggregate element $e_i \in S$ of the form $t_1, \ldots, t_m : conj$ and $conj$ is true w.r.t. I; $I(S)$, instead, denotes the multi-set $[t_1 \mid (t_1, \ldots, t_m) \in eval(S, I)]$. A ground aggregate literal of the form $f\{e_1; \ldots; e_n\} \succ t$ (resp. $\sim f\{e_1; \ldots; e_n\} \succ t$) is true w.r.t. I if $f(I(\{e_1, \ldots, e_n\})) \succ t$ holds (resp. does not hold); otherwise it is false. An interpretation I is a *model* of P iff for each rule $r \in ground(P)$ either the head of r is true w.r.t. I or the body of r is false w.r.t. I. Given an interpretation I, P^I denotes the *FLP-reduct* (cfr. Faber *et al.* (2011)) obtained by removing all those rules in P having their body false, and removing negative literals from the body of remaining rules. A model I of P is also an *answer set* of P if for each $I' \subset I$, I' is not a model of P^I. We write $AS(P)$ for the set of answer sets of P. A program P is *coherent* if it has at least one answer set (i.e., $AS(P) \neq \emptyset$); otherwise, P is *incoherent*. For a program P and an interpretation I, let the set of weak constraint violations be $ws(P, I) = \{(w, l, T) \mid \leftarrow_w b_1, \ldots, b_m\ [w@l, T] \in ground(\mathcal{W}(P))$, $b_1, \ldots, b_m$ are true w.r.t. I, w, and l are integers and T is a tuple of ground terms$\}$, then the cost function of P is $\mathcal{C}(P, I, l) = \Sigma_{(w,l,T) \in ws(P,I)} w$, for every integer l. Given a program P and two interpretations I_1 and I_2, we say that that I_1 is *dominated* by I_2 if there is an integer l such that $\mathcal{C}(P, I_2, l) < \mathcal{C}(P, I_1, l)$ and for all integers $l' > l$, $\mathcal{C}(P, I_2, l') = \mathcal{C}(P, I_1, l')$. An answer set $M \in AS(P)$ is an *optimal* answer set if it is not dominated by any $M' \in AS(P)$. Intuitively, optimality amounts to minimizing the weight at the highest possible level, with each level used for tie breaking for the level directly above. The set $OptAS(P) \subseteq AS(P)$ denotes the set of optimal answer sets of P.

3 Quantified Answer Set Programming with weak constraints

In this section, we introduce an extension of Answer Set Programming with Quantifiers (ASP(Q)) (Amendola *et al.* 2019) that explicitly supports weak constraints (Buccafurri *et al.* 2000) for modeling optimization problems.

It is worth noting that ASP(Q) can be used to model problems with model preferences and optimization criteria; however, this comes at the price of non-elegant and somehow redundant modeling. For this reason, in analogy to what has been done for ASP, it makes sense to contemplate weak constraints in ASP(Q).

A *quantified ASP program with weak constraints* (ASP$^\omega$(Q) program) Π is of the form:

$$\Box_1 P_1\ \Box_2 P_2\ \cdots\ \Box_n P_n : C : C^w, \tag{3}$$

where, for each $i = 1, \ldots, n$, $\Box_i \in \{\exists^{st}, \forall^{st}\}$, P_i is an ASP program possibly with weak constraints, C is a (possibly empty) *stratified* program (Ceri *et al.* 1990) with constraints, and C^w is a (possibly empty) set of weak constraints such that $B_{C^w} \subseteq B_{P_1}$. The number of quantifiers in Π is denoted by $nQuant(\Pi)$.

As it was in the base language, ASP$^\omega$(Q) programs are quantified sequences of subprograms ending with a *constraint program* C. Differently from ASP(Q), in ASP$^\omega$(Q) weak constraints are allowed in the subprograms P_i $(1 \leq i \leq n)$, that is, quantification is over optimal answer sets. Moreover, the *global weak constraints* subprogram C^w is introduced to specify (global) optimality criteria on quantified answer sets.

Formally, the *coherence* of ASP$^\omega$(Q) programs is defined as follows:

- $\exists^{st} P : C : C^w$ is coherent, if there exists $M \in OptAS(P)$ such that $C \cup \mathit{fix}_P(M)$ admits an answer set;
- $\forall^{st} P : C : C^w$ is coherent, if for every $M \in OptAS(P)$, $C \cup \mathit{fix}_P(M)$ admits an answer set;
- $\exists^{st} P\ \Pi$ is coherent, if there exists $M \in OptAS(P)$ such that $\Pi_{P,M}$ is coherent;
- $\forall^{st} P\ \Pi$ is coherent, if for every $M \in OptAS(P)$, $\Pi_{P,M}$ is coherent.

where $\mathit{fix}_P(M)$ denotes the set of facts and constraints $\{a \mid a \in M \cap HB_P\} \cup \{\leftarrow a \mid a \in HB_P \setminus M\}$, and $\Pi_{P,M}$ denotes the ASP$^\omega$(Q) program of the form (3), where P_1 is replaced by $P_1 \cup \mathit{fix}_P(M)$, that is, $\Pi_{P,M} = \Box_1(P_1 \cup \mathit{fix}_P(M))\ \Box_2 P_2\ \cdots \Box_n P_n : C : C^w$.

For an existential ASP$^\omega$(Q) program Π, $M \in OptAS(P_1)$ is a *quantified answer set* of Π, if $((\Box_2 P_2 \cdots \Box_n P_n : C) : C^w)_{P_1,M}$ is coherent. We denote by $QAS(\Pi)$ the set of all quantified answer sets of Π.

To illustrate the definitions above, let us consider the following ASP$^\omega$(Q) program $\Pi = \exists^{st} P_1 \forall^{st} P_2 \cdots \exists^{st} P_{n-1} \forall^{st} P_n : C : C^w$. "Unwinding" the definition of coherence yields that Π is coherent if there exists an *optimal* answer set M_1 of P'_1 such that for every *optimal* answer set M_2 of P'_2 there exists an *optimal* answer set M_3 of P'_3, and so on until there exists an *optimal* answer set M_{n-1} of P'_{n-1} such that for every *optimal* answer set M_n of P'_n, there exists an answer set of $C \cup \mathit{fix}_{P'_n}(M_n)$, where $P'_1 = P_1$, and $P'_i = P_i \cup \mathit{fix}_{P'_{i-1}}(M_{i-1})$ with $i \geq 2$. Note that, as in ASP(Q), the constraint program C has the role of selecting admissible solutions. Weak constraints could be allowed in C, but they would be redundant. Indeed, C, being stratified with constraints, admits at most one answer set, which would necessarily be optimal. In contrast, the *local weak constraints* (possibly) occurring in subprograms P_i are essential for determining coherence.

Example 3.1 (Impact of local weak constraints).
Let $\Pi_1 = \exists P_1 \forall P_2 : C$, and $\Pi_2 = \exists Q_1 \forall Q_2 : C$, where $C = \{\leftarrow d, f\}$ and also:

$$P_1 = \left\{ \begin{array}{l} \{a; b\} = 1 \leftarrow \\ \{c; d\} = 1 \leftarrow \\ \leftarrow_w c\ [1@1] \end{array} \right\} \quad P_2 = \left\{ \begin{array}{l} \{e, f\} \leftarrow \\ \leftarrow \sim e, \sim f \\ \leftarrow_w e, f\ [1@1] \end{array} \right\} \quad Q_1 = \left\{ \begin{array}{l} \{a; b\} = 1 \leftarrow \\ \{c; d\} = 1 \leftarrow \end{array} \right\}$$

$$Q_2 = \left\{ \begin{array}{l} \{e, f\} \leftarrow \\ \leftarrow \sim e, \sim f \end{array} \right\}$$

Note that, Π_2 can be obtained from Π_1 by discarding weak constraints. First, we observe that Π_1 is incoherent. Indeed, the optimal answer sets of P_1 are $OptAS(P_1) = \{\{a, d\}, \{b, d\}\}$. By applying the definition of coherence, when we consider $M = \{a, d\}$, we have that $OptAS(P'_2) = \{\{e, a, d\}, \{f, a, d\}\}$. Once we set $M' = \{f, a, d\}$, the program C' is not coherent, and so $M = \{a, d\}$ is not a quantified answer set. Analogously, when we consider the second answer set of P_1, that is, $M = \{b, d\}$, we have that $OptAS(P'_2) = \{\{e, a, d\}, \{f, a, d\}\}$. But, when we set $M' = \{f, b, d\}$, the program C' is not coherent. Thus, Π_1 is incoherent. On the contrary, Π_2 is coherent. Indeed, $AS(Q_1) = \{\{a, d\}, \{b, d\}, \{a, c\}, \{b, c\}\} = OptAS(P_1) \cup \{\{a, c\}, \{b, c\}\}$. The first two, we know, do not lead to a quantified answer set. But, when we set $M = \{a, c\}$, since d is false, it happens that C' is coherent (e.g., when we consider the answer set $\{e, a, c\}$ of Q_2'). Thus, local weak constraints can affect coherence by discarding not optimal candidates.

Global weak constraints in C^w do not affect coherence, but they serve to define optimality criteria across quantified answer sets. For this reason, we require that C^w is defined over the same Herbrand base of P_1. Furthermore, note that C^w plays no role in universal ASP$^\omega$(Q) programs, where coherence is the sole meaningful task.

Given an existential ASP$^\omega$(Q) program Π and two quantified answer sets $Q_1, Q_2 \in QAS(\Pi)$, we say that Q_1 is dominated by Q_2 if there exists an integer l such that $\mathcal{C}(P_1^*, Q_2, l) < \mathcal{C}(P_1^*, Q_1, l)$ and for every integer $l' > l$, $\mathcal{C}(P_1^*, Q_2, l) = \mathcal{C}(P_1^*, Q_1, l)$, where $P_1^* = P_1 \cup C^w$. An *optimal quantified answer set* is a quantified answer set $Q \in QAS(\Pi)$ that is not dominated by any $Q' \in QAS(\Pi)$.

Example 3.2 (Optimal quantified answer sets)
Let $\Pi = \exists P_1 \forall P_2 : C : C^w$ be such that:

$$P_1 = \{\{a; b; c\} \leftarrow\} \quad P_2 = \left\{ \begin{array}{l} \{a'; b'; c'\} \leftarrow \\ \leftarrow a', \sim b' \\ \leftarrow \sim a', \sim b' \\ \leftarrow a', \sim c' \\ \leftarrow \sim a', \sim c' \end{array} \right\} \quad C = \left\{ \begin{array}{l} \leftarrow a, \sim a' \\ \leftarrow b, \sim b' \\ \leftarrow c, \sim c' \end{array} \right\} \quad C^w = \left\{ \begin{array}{l} \leftarrow \sim a\ [1@1, a] \\ \leftarrow \sim b\ [1@1, b] \\ \leftarrow \sim c\ [1@1, c] \end{array} \right\}$$

Given that $QAS(\Pi) = \{\{\}, \{b\}, \{c\}, \{b, c\}\}$, we have that: the cost of $\{\}$ is 3, since it violates all weak constraints in C^w; {b} and {c} cost 2, since $\{b\}$ (resp. $\{c\}$) violates the first and the third (resp. second) weak constraint; and, {b,c} costs 1, because it only violates the first weak constraint.

Let $\Pi = \Box_1 P_1 \ldots \Box_n P_n$ be an ASP$^\omega$(Q) program. Π satisfies the stratified definition assumption if for each $1 \leq i \leq n$, $\mathcal{H}(P_i) \cap at(P_j) = \emptyset$, with $1 \leq j < i$. In what follows, we assume w.l.o.g. that ASP$^\omega$(Q) programs satisfy the *stratified definition assumption*.

It is worth noting that, standard ASP(Q) allows for the specification of preferences and optimization. The basic pattern for obtaining optimal models in ASP(Q) is to "clone" a program and use an additional quantifier over its answer sets. This allows us to compare pairs of answer sets and, by means of a final constraint program, to select optimal ones. For example, assume program P_1 models the candidate solutions of a problem and, for the sake of illustration, that we are interested in those minimizing the number of atoms of the form $a(X)$. This desideratum can be modeled directly in standard ASP by adding a weak constraint $\leftarrow_w a(X)[1@1, X]$. On the other hand, in ASP(Q) we can model it with the program $\exists P_1 \forall P_2 : C$ such that $P_2 = clone^s(P_1)$, and $C = \{\leftarrow \#count\{X : a(X)\} = K, \#count\{X : a^s(X)\} < K\}$. Here, we are comparing the answer sets of P_1 with all their "clones", and keep those that contain a smaller (or equal) number of atoms of the form $a(X)$. This pattern is easy to apply, but it is redundant; also note that checking coherence of an ASP(Q) program with two quantifiers is in Σ_2^p (Amendola *et al.* 2019), whereas optimal answer set checking of a program with weak constraints is in Δ_2^p (Buccafurri *et al.* 2000). These observations motivate the introduction of weak constraints in ASP(Q), which will be further strengthened in the following sections.

4 Modeling examples

We showcase the modeling capabilities of ASP$^\omega$(Q) by considering two example scenarios where both global and local weak constraints play a role: the Minmax Clique problem (Cao *et al.* 1995), and Logic-Based Abduction (Eiter and Gottlob 1995a).

4.1 Minmax Clique problem

Minimax problems are prevalent across numerous research domains. Here, we focus on the Minmax Clique problem, as defined by Ko (1995), although other minimax variants can be also modeled.

Given a graph $G = \langle V, E\rangle$, let I and J be two finite sets of indices, and $(A_{i,j})_{i\in I, j\in J}$ a partition of V. We write J^I for the set of all total functions from I to J. For every total function $f\colon I \to J$ we denote by G_f the subgraph of G induced by $\bigcup_{i\in I} A_{i,f(i)}$. The Minmax Clique optimization problem is defined as follows: Given a graph G, sets of indices I and J, a partition $(A_{i,j})_{i\in I, j\in J}$, find the integer k ($k \leq |V|$), such that

$$k = \min_{f\in J^I} \max\{|Q| : Q \text{ is a clique of } G_f\}.$$

The following program of the form $\Pi = \exists P_1 \exists P_2 : C : C^w$, encodes the problem:

$$P_1 = \left\{ \begin{array}{rcl} v(i,j,a) & \leftarrow & \qquad \forall i \in I, j \in J, a \in A_{i,j} \\ inI(i) & \leftarrow & \qquad \forall i \in I \\ inJ(j) & \leftarrow & \qquad \forall j \in J \\ e(x,y) & \leftarrow & \qquad \forall (x,y) \in E \\ \{f(i,j) : inJ(j)\} = 1 & \leftarrow inI(i) & \\ \{valK(1); \ldots; valK(|V|)\} = 1 & \leftarrow & \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{rcl} n_f(X) & \leftarrow & f(I,J), v(I,J,X) \\ e_f(X,Y) & \leftarrow & n_f(X), n_f(Y), e(X,Y) \\ \{inClique(X) : n_f(X)\} & \leftarrow & \\ & \leftarrow & inClique(X), inClique(Y), X < Y, {\sim} e_f(X,Y) \\ & \leftarrow_w & n_f(X), {\sim} inClique(X) \quad [1@1, X] \end{array} \right\}$$

$$C = \{ \leftarrow \quad valK(K), \#count\{X : inClique(X)\} \neq K \}$$

$$C^w = \{ \leftarrow_w val(K) \quad [K@1] \}$$

The input is modeled in program P_1 as follows: Node partitions are encoded as facts of the form $v(i,j,x)$ denoting that a node x belongs to the partition $a \in A_{i,j}$; facts of the form $inI(i)$ and $inJ(j)$ model indexes $i \in I$ and $j \in J$, respectively; and, the set of edges E is encoded as facts of the form $e(x,y)$ denoting that the edge $(x,y) \in E$. The first choice rule in P_1 guesses one total function $f : I \to J$, which is encoded by binary predicate $f(i,j)$ denoting that the guessed function maps i to j. The second choice rule guesses one possible value for k, modeled by predicate $valK(x)$. Thus, there is an answer set of program P_1 for each total function f and a possible value for k.

Given an answer set of P_1, program P_2 computes the maximum clique of the subgraph of G induced by f, that is, G_f. To this end, the first rule computes the nodes of G_f in predicate $n_f(X)$, by joining predicate $f(I,J)$ and $v(I,J,X)$. The second rule computes the edges of G_f considering the edges of G that connect nodes in G_f. The largest clique in G_t is computed by a (i) choice rule that guesses a set of nodes (in predicate *inClique*), (ii) a constraint requiring that nodes are mutually connected, and (iii) a weak constraint that minimizes the number of nodes that are not part of the clique. At this point, the program C verifies that the size of the largest clique in the answer set of P_2 is exactly the value for k in the current answer set of P_1. Thus, each quantified answer set of Π models a function f, such that the largest clique of induced graph G_f has size k. Now, the global weak constraints in C^w prefer the ones that give the smallest value of k.

The decision version of this problem is Π_2^p-complete (Ko 1995). Thus, a solution to the Minmax Clique can be computed by a logarithmic number of calls to an oracle in Π_2^p, so the problem belongs to Θ_3^P (Wagner 1990). It is (somehow) surprising that we could write a natural encoding without alternating quantifiers (indeed, Π features two *existential* quantifiers). This phenomenon is more general. We will return to it in Section 6.

Logic-Based Abduction Abduction plays a prominent role in Artificial Intelligence as an essential common-sense reasoning mechanism (Morgan 1971; Pople 1973).

In this paper we focus on the Propositional Abduction Problem (PAP) (Eiter and Gottlob 1995a). The PAP is defined as a tuple of the form $\mathcal{A} = \langle V, T, H, M\rangle$, where V is a set of variables, T is a consistent propositional logic theory over variables in V, $H \subseteq V$ is a set of hypotheses, and $M \subseteq V$ is a set of manifestations. A *solution* to the PAP problem $\mathcal{A}$ is a set $S \subseteq H$ such that $T \cup S$ is consistent and $T \cup S \models M$. Solutions to $\mathcal{A}$, denoted by $sol(\mathcal{A})$, can be ordered by means of some preference relation $<$. The set of optimal solutions to $\mathcal{A}$ is defined as $sol_<(\mathcal{A}) = \{S \in sol(\mathcal{A}) \mid \nexists\ S' \in sol(\mathcal{A})\ such\ that\ |S'| < |S|\}$. A hypothesis $h \in H$ is *relevant* if h appears at least in one solution $S \in sol_<(\mathcal{A})$. The main reasoning tasks for PAP are beyond NP (Eiter and Gottlob 1995a).

In the following, we assume w.l.o.g. that the theory T is a boolean 3-CNF formula over variables in V. Recall that, a 3-CNF formula is a conjunction of clauses $C_1 \wedge \ldots \wedge C_n$, where each clause is of the form $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, and each literal l_i^j (with $1 \leq j \leq 3$) is either a variable $a \in V$ or its (classical) negation $\neg a$.

Given a PAP problem $\mathcal{A} = \langle V, T, H, M\rangle$ we aim at computing a solution $S \in sol_<(\mathcal{A})$. To this end, we use an ASP$^\omega$(Q) program of the form $\exists P_1 \forall P_2 : C : C^w$, where:

$$P_1 = \left\{ \begin{array}{rcll} v(x) & \leftarrow & & \forall\ x \in V \\ lit(C_i, a, t) & \leftarrow & & \forall\ a \in V \mid a \in C_i \\ lit(C_i, a, f) & \leftarrow & & \forall\ a \in V \mid {\sim} a \in C_i \\ h(x) & \leftarrow & & \forall\ x \in H \\ m(x) & \leftarrow & & \forall\ x \in M \\ cl(X) & \leftarrow & lit(X, \_, \_) & \\ \{s(X) : h(X)\} & \leftarrow & & \\ \{tau(X,t); tau(X,f)\} = 1 & \leftarrow & v(X) & \\ satCl(C) & \leftarrow & lit(C, A, V), tau(A, V) & \\ & \leftarrow & cl(C), {\sim} satCl(C) & \\ & \leftarrow & s(X), tau(X, f) & \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{rcl} \{tau'(X,t); tau'(X,f)\} = 1 & \leftarrow & v(X) \\ satCl'(C) & \leftarrow & lit(C,A,V), tau'(A,V) \\ unsatTS' & \leftarrow & cl(C), \sim satCl'(C) \\ unsatTS' & \leftarrow & s(X), \sim tau'(X,f) \end{array} \right\}$$

$$C = \{ \leftarrow \sim unsatTS', m(X), tau'(X,f) \}$$

$$C^w = \{ \leftarrow_w s(X) \ [1@1, X] \}$$

The aim of P_1 is to compute a candidate solution $S \subseteq H$ such that $T \cup S$ is consistent; P_2 and C ensure that $T \cup S \vDash M$, and C^w ensures that S is cardinality minimal. More in detail, in program P_1, the variables V, hypothesis H, and manifestations M, are encoded by means of facts of the unary predicates v, h, and m, respectively. The formula T is encoded by facts of the form $lit(C,x,t)$ (resp. $lit(C,x,f)$) denoting that a variable x occurs in a positive (resp. negative) literal in clause C. Then, to ease the presentation, we compute in a unary predicate cl the set of clauses. The first choice rule guesses a solution (a subset of H), and the last five rules verify the existence of a truth assignment τ for variables in V, encoded with atoms of the form $tau(x,t)$ (resp. $tau(x,f)$) denoting that a variable x is true (resp. false), such that $unsatTS$ is not derived (last constraint). Note that, $unsatTS$ is derived either if a clause is not satisfied or if a hypothesis is not part of the assignment. Thus, the assignment τ satisfies $T \cup S$, that is, $T \cup S$ is consistent. It follows that the answer sets of P_1 correspond to candidate solutions $S \subseteq H$ such that $T \cup S$ is consistent. Given a candidate solution, program P_2 has one answer set for each truth assignment τ' that satisfies $T \cup S$, and the program C checks that all such τ' satisfy also the manifestations in M. Thus, every $M \in QAS(\Pi)$ encodes a solution $S \in sol(\mathcal{A})$. The weak constraint in C^w ensures we single out cardinality minimal solutions by minimizing the extension of predicate s. Finally, let h be a hypothesis, we aim at checking that h is relevant, that is, $h \in S$ s.t. $S \in sol_<(\mathcal{A})$. We solve this task by taking the program Π above that computes an optimal solution and adding to C^w an additional (ground) weak constraint, namely $\leftarrow_w \sim s(h) \quad [1@0]$. Intuitively, optimal solutions not containing h violate the weak constraint, so if any optimal answer set contains $s(h)$ then h is relevant.

4.2 Remark

Checking that a solution to a PAP is minimal belongs to Π_2^p (Eiter and Gottlob 1995a), so the task we have considered so far is complete for Θ_3^P (Wagner 1990). The programs above feature only two quantifiers, whereas alternative encodings in ASP(Q) (i.e., without weak constraints) would have required more. Moreover, we observe that the programs above are rather natural renderings of the definition of the problems that showcase the benefit of modeling optimization in subprograms and at the global level.

5 Rewriting into plain ASP(Q)

In this section, we describe a mapping that transforms an ASP$^{\omega}$(Q) program Π into a plain (i.e., without weak constraints) quantifier-alternating ASP(Q) program Π' that is coherent iff Π is coherent. This transformation is crucial for enabling the study of the complexity of the primary reasoning tasks of ASP$^{\omega}$(Q). Additionally, it could be applied in an implementation that extends current solvers such as that by Faber *et al.* (2023).

The transformation works by calling a number of intermediate rewritings until none of them can be applied anymore. They (i) absorb consecutive quantifiers of the same kind; and, (ii) eliminate weak constraints from a subprogram by encoding the optimality check in the subsequent subprograms. We first introduce some useful definitions. Given program Π of the form (3) we say that two consecutive subprograms P_i and P_{i+1} are *alternating* if $\Box_i \neq \Box_{i+1}$, and are *uniform* otherwise. A program Π is *quantifier-alternating* if $\Box_i \neq \Box_{i+1}$ for $1 \leq i < n$. A subprogram P_i is *plain* if it contains no weak constraint $\mathcal{W}(P_i) = \emptyset$, and Π is *plain* if both all P_i are plain, and $C^w = \emptyset$. In the following, we assume that Π is an ASP$^{\omega}$(Q) program of the form (3).

5.1 Rewriting uniform plain subprograms

Two plain uniform subprograms can be absorbed in a single equi-coherent subprogram by the transformation $col_1(\cdot)$ defined as follows.

Lemma 1 (Correctness $col_1(.)$ transformation)
Let program Π be such that $n \geq 2$ and the first two subprograms are plain and uniform, that is, $\Box_1 = \Box_2$, and $\mathcal{W}(P_1) = \mathcal{W}(P_2) = \emptyset$, then Π is coherent if and only if $col_1(\Pi) = \Box_1 P_1 \cup P_2 \Box_3 P_3 \ldots \Box_n P_n : C$ is coherent.

Intuitively, if the first two subprograms of Π are uniform and plain then Π can be reformulated into an equi-coherent (i.e., Π is coherent iff $col_1(\Pi)$ is coherent) program with one fewer quantifier.

5.2 Rewriting uniform notplain-plain subprograms

Next transformations apply to pairs of uniform subprograms P_1, P_2 such that P_1 is not plain and P_2 is plain. To this end, we first define the $or(\cdot,\cdot)$ transformation. Let P be an ASP program, and l be a fresh atom not appearing in P, then $or(P,l) = \{H_r \leftarrow B_r, \sim l \mid r \in P\}$.

Observation 1 (Trivial model existence)
Let P be an ASP program, and l be a fresh literal not appearing in P, then the following hold: $\{l\}$ is the unique answer set of $or(P,l) \cup \{l \leftarrow\}$; and $AS(or(P,l) \cup \{\leftarrow l\}) = AS(P)$.

Intuitively, if the fact $l \leftarrow$ is added to $or(P,l)$ then the interpretation $I = \{l\}$ trivially satisfies all the rules and is minimal, thus it is an answer set. On the other hand, if we add the constraint $\leftarrow l$, requiring that l is false in any answer set, then the resulting program behaves precisely as P since literal $\sim l$ is trivially true in all the rule bodies.

We are now ready to introduce the next rewriting function $col_2(\cdot)$. This transformation allows to absorbe a plain existential subprogram into a non plain existential one, thus reducing by one the number of quantifiers of the input ASP$^\omega$(Q) program.

Definition 1 (Collapse notplain-plain existential subprograms)
Let Π be an ASP$^\omega$(Q) program of the form $\exists P_1 \exists P_2 \ldots \Box_n P_n : C$, where $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$, with $1 < i \leq n$, and $\Box_i \neq \Box_{i+1}$ with $1 < i < n$, then:

$$col_2(\Pi) = \begin{cases} \exists P_1 \cup or(P_2, unsat) \cup W : C \cup \{\leftarrow unsat\} & n = 2 \\ \exists P_1 \cup or(P_2, unsat) \cup W \;\; \forall P_3' : C \cup \{\leftarrow unsat\} & n = 3 \\ \exists P_1 \cup or(P_2, unsat) \cup W \;\; \forall P_3' \;\; \exists P_4 \cup \{\leftarrow unsat\} \ldots \Box_n P_n : C & n > 3 \end{cases}$$

where $W = \{\{unsat\} \leftarrow\} \cup \{\leftarrow_w unsat\ [1@l_{min} - 1]\}$, with l_{min} be the lowest level in $\mathcal{W}(P_1)$ and $unsat$ is a fresh symbol not appearing anywhere else, and $P_3' = or(P_3, unsat)$.

Lemma 2 (Correctness $col_2(.)$ transformation)
Let Π be an ASP$^\omega$(Q) program of the form $\exists P_1 \exists P_2 \ldots \Box_n P_n : C$, where $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$, with $1 < i \leq n$, and $\Box_i \neq \Box_{i+1}$ with $1 < i < n$. Then Π is coherent if and only if $col_2(\Pi)$ is coherent.

A similar procedure is introduced for the universal case.

Definition 2 (Collapse notplain-plain universal subprograms)
Let Π be an ASP$^\omega$(Q) program of the form $\forall P_1 \forall P_2 \ldots \Box_n P_n : C$, where $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$, with $1 < i \leq n$, and $\Box_i \neq \Box_{i+1}$ with $1 < i < n$, then:

$$col_3(\Pi) = \begin{cases} \forall P_1 \cup or(P_2, unsat) \cup W : or(C, unsat) & n = 2 \\ \forall P_1 \cup or(P_2, unsat) \cup W \;\; \exists P_3' : or(C, unsat) & n = 3 \\ \forall P_1 \cup or(P_2, unsat) \cup W \;\; \exists P_3' \;\; \forall P_4 \cup \{\leftarrow unsat\} \ldots \Box_n P_n : C & n > 3 \end{cases}$$

where $W = \{\{unsat\} \leftarrow\} \cup \{\leftarrow_w unsat\ [1@l_{min} - 1]\}$, with l_{min} be the lowest level in $\mathcal{W}(P_1)$ and $unsat$ is a fresh symbol not appearing anywhere else, and $P_3' = or(P_3, unsat)$.

Lemma 3 (Correctness $col_3(.)$ transformation)
Let Π be an ASP$^\omega$(Q) program of the form $\forall P_1 \forall P_2 \ldots \Box_n P_n : C$, where $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$, with $1 < i \leq n$, and $\Box_i \neq \Box_{i+1}$ with $1 < i < n$. Then Π is coherent if and only if $col_3(\Pi)$ is coherent.

Roughly, if the first two subprograms of Π are uniform, P_1 is not plain, P_2 is plain, and the remainder of the program is alternating, then Π can be reformulated into an equi-coherent program with one fewer quantifier.

5.3 Rewrite subprograms with weak constraints

The next transformations have the role of eliminating weak constraints from a subprogram by encoding the optimality check in the subsequent subprograms. To this end, we define the $check(\cdot)$ transformation that is useful for simulating the cost comparison of two answer sets of an ASP program P.

First, let ϵ be an ASP expression and s an alphanumeric string. We define $clone^s(\epsilon)$ as the expression obtained by substituting all occurrences of each predicate p in ϵ with p^s which is a fresh predicate p^s of the same arity.

Definition 3 (Transform weak constraints)
Let P be an ASP program with weak constraints, then

$$check(P) = \left\{ \begin{array}{rcl} v_c(w,l,T) & \leftarrow & b_1,\ldots,b_m \\ & & \qquad\qquad \forall c:\ \leftarrow_w b_1,\ldots,b_m[w@l,T] \in P \\ cl_P(C,L) & \leftarrow & level(L), C = \#sum\{w_{c_1};\ldots;w_{c_n}\} \\ clone^o(v_c(w,l,T) & \leftarrow & b_1,\ldots,b_m) \\ & & \qquad\qquad \forall c:\ \leftarrow_w b_1,\ldots,b_m[w@l,T] \in P \\ clone^o(cl_P(C,L) & \leftarrow & level(L), C = \#sum\{w_{c_1};\ldots;w_{c_n}\}) \\ diff(L) & \leftarrow & cl_P(C1,L), cl^o_P(C2,L), C1 \neq C2 \\ hasHigher(L) & \leftarrow & diff(L), diff(L1), L < L1 \\ higest(L) & \leftarrow & diff(L), \sim hasHigher(L) \\ dom_P & \leftarrow & higest(L), cl_P(C1,L), cl^o_P(C2,L), C2 < C1 \end{array} \right.$$

where each w_{c_i} is an aggregate element of the form $W,T : v_{c_i}(W,L,T)$.

Thus, the first two rules compute in predicate cl_P the cost of an answer set of P w.r.t. his weak constraints, and the following two rules do the same for $clone^o(P)$. Then, the last four rules derive dom_P for each answer set of P that is dominated by $clone^o(P)$.

We now introduce how to translate away weak constraints from a subprogram.

Definition 4 (Transform existential not-plain subprogram)
Let Π be an existential alternating ASP$^\omega$(Q) program such that all subprograms are plain except the first one (i.e., $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$, $1 < i \leq n$), then

$$col_4(\Pi) = \left\{ \begin{array}{ll} \exists\mathcal{R}(P_1)\forall clone^o(\mathcal{R}(P_1)) \cup check(P_1) : \{\leftarrow dom_{P_1}\} \cup C & n = 1 \\ \exists\mathcal{R}(P_1)\forall P'_2 : \{\leftarrow dom_{P_1}\} \cup C & n = 2 \\ \exists\mathcal{R}(P_1)\forall P'_2 \exists P_3 \cup \{\leftarrow dom_{P_1}\} \ldots \Box_n P_n : C & n \geq 3 \end{array} \right.$$

where $P'_2 = clone^o(\mathcal{R}(P_1)) \cup check(P_1) \cup or(P_2, dom_{P_1})$.

Lemma 4 (Correctness $col_4(.)$ transformation)
Let Π be an existential alternating ASP$^\omega$(Q) program such that all subprograms are plain except the first, then Π is coherent if and only if $col_4(\Pi)$ is coherent.

Intuitively, for a pair $M_1, M_2 \in AS(P_1)$, M_1 is dominated by M_2 if and only if $check(P) \cup fix_P(M_1) \cup clone^o(fix_P(M_2))$ admits an answer set M such that $dom_P \in M$. Thus, coherence is preserved since dom_{P_1} discards not optimal candidates such as M_1.

A similar procedure can be defined for universal subprogram.

Definition 5 (Transform universal not-plain subprogram).
Let Π be a universal alternating ASP$^\omega$(Q) program such that all subprograms are plain except the first one (i.e., $\mathcal{W}(P_1) \neq \emptyset$, $\mathcal{W}(P_i) = \emptyset$ $1 < i \leq n$), then

Algorithm 1 Rewriting from ASP$^\omega$(Q) to ASP(Q)

```
Input  : An ASP^ω(Q) program $\Pi$
Output: A quantifier-alternating ASP(Q) program
begin
    $s := 0$;    $\Pi_0 := \Pi$
    do
        $stop := \top$
        for all $ProgramType \in [1,5]$ do
            Let $i \in [1,n]$ be the largest index such that $\Pi_s^{\geq i}$ is of the type $ProgramType$
            if  $i \neq \bot$ then
                $\Pi_{s+1} := replace(\Pi_s, i, col_{ProgramType}(\Pi_s^{\geq i}))$
                $s := s+1$;
                $stop := \bot$
                $break$                                        // go to line 12
    while $stop \neq \top$;
    return $removeGlobal(\Pi_s)$
```

$$col_5(\Pi) = \begin{cases} \forall \mathcal{R}(P_1)\exists clone^o(\mathcal{R}(P_1)) \cup check(P_1) : or(C, dom_{P_1}) & n=1 \\ \forall \mathcal{R}(P_1)\exists P_2' : or(C, dom_{P_1}) & n=2 \\ \forall \mathcal{R}(P_1)\exists P_2' \forall P_3 \cup \{\leftarrow dom_{P_1}\} \dots \Box_n P_n : C & n \geq 3 \end{cases}$$

where $P_2' = clone^o(\mathcal{R}(P_1)) \cup check(P_1) \cup or(P_2, dom_{P_1})$.

Lemma 5 (Correctness $col_5(.)$ transformation)
Let Π be a universal alternating ASP$^\omega$(Q) program such that all subprograms are plain except the first, then Π is coherent if and only if $col_5(\Pi)$ is coherent.

Translate ASP$^\omega$(Q) to ASP(Q).
Algorithm 1 defines a procedure for rewriting an ASP$^\omega$(Q) program Π into an ASP(Q) program Π', made of at most $n+1$ alternating quantifiers, such that Π is coherent if and only if Π' is coherent. In Algorithm 1, we make use of some (sub)procedures and dedicated notation. In detail, for a program Π of the form (3), $\Pi^{\geq i}$ denotes the *i-th suffix program* $\Box_i P_i \dots \Box_n P_n : C$, with $1 \leq i \leq n$ (i.e., the one obtained from Π removing the first $i-1$ quantifiers and subprograms). Moreover, the procedure $removeGlobal(\Pi)$ builds an ASP(Q) program from a plain one in input (roughly, it removes the global constraint program C^w). Given two programs Π_1 and Π_2, $replace(\Pi_1, i, \Pi_2)$ returns the ASP$^\omega$(Q) program obtained from Π_1 by replacing program $\Pi_1^{\geq i}$ by Π_2, for example $replace(\exists P_1 \forall P2 \exists P_3 : C,\ 2,\ \exists P_4 : C)$ returns $\exists P_1 \exists P_4 : C$.

In order to obtain a quantifier alternating ASP(Q) program from the input Π, Algorithm 1 generates a sequence of programs by applying at each step one of the col_T transformations. With a little abuse of notation, we write that a program is of type T ($T \in [1,5]$) if it satisfies the conditions for applying the rewriting col_T defined above (cfr., Lemmas 1–5). For example, when type $T=1$ we check that the first two subprograms of Π are plain and uniform so that col_1 can be applied to program Π. In detail, at each iteration s, the innermost suffix program that is of current type T is identified, say $\Pi_s^{\geq i}$. Then the next program Π_{s+1} is built by replacing $\Pi_s^{\geq i}$ by $col_T(\Pi_s^{\geq i})$. Algorithm 1 terminates when no transformation can be applied, and returns the program $removeGlobal(\Pi_s)$.

Theorem 1 (ASP$^{\Omega}$(Q) to ASP(Q) convergence and correctness)
Given program Π, Algorithm 1 terminates and returns an alternating ASP(Q) program Π' that is Π' is coherent iff Π is coherent, and $nQuant(\Pi') \leq nQuant(\Pi) + 1$.

Intuitively, the proof follows by observing that Algorithm 1 repeatedly simplifies the input by applying $col_T(\cdot)$ procedures ($T \in [1, 5]$) until none can be applied. This condition happens when the resulting Π' is plain alternating. Equi-coherence follows from Lemmas 1-5. Unless the innermost subprogram of Π is not plain, no additional quantifier is added by Algorithm 1, so $nQuant(\Pi') \leq nQuant(\Pi) + 1$, hence the proof follows.

Proof
At each step s, Algorithm 1 searches for the innermost suffix subprogram $\Pi_s^{\geq i}$ such that either (i) $\Pi_s^{\geq i}$ begins with two consecutive quantifiers of the same type (i.e., it is of type 1,2 or 3), or (ii) $\Pi_s^{\geq i}$ begins with a not plain subprogram followed by a quantifier alternating sequence of plain subprograms (i.e., it is of type 4 or 5). In case (i), one of the subprocedures col_1, col_2, or col_3 is applied, which results in the computation of program Π_{s+1} having one less pair of uniform subprograms (i.e., $nQuant(\Pi_{s+1}) = nQuant(\Pi_s) - 1$). In case (ii), one of the subprocedures col_4, col_5 is applied, which results in the computation of program Π_{s+1} such that its i-th subprogram is plain. After applying col_4, col_5 we have that $nQuant(\Pi_{s+1}) \leq nQuant(\Pi_s) + 1$, indeed if $i = nQuant(\Pi_s)$ one more quantifier subprogram is added. So the algorithm continues until neither condition (i) nor (ii) holds. This happens when Π_s is a plain quantifier alternating program. Note that, unless the innermost subprogram of Π is not plain, no additional quantifier is added during the execution of Algorithm 1 (if anything, some may be removed), so $nQuant(\Pi') \leq nQuant(\Pi) + 1$. □

Additionally, it is easy to see that quantified answer set of existential programs can be preserved if only atoms of the first subprogram are made visible.

Corollary 1.1 (Quantified answer set preservation)
Let Π be an existential ASP$^{\omega}$(Q) program of the form (3) and Π' be the result of the application of Algorithm 1 on Π. Then, $M \in QAS(\Pi)$ if and only if there exists $M' \in QAS(\Pi')$ such that $M' \cap HB_{P_1} = M$.

The Corollary above follows (straightforwardly) from Theorem 1 because the only cases in which the first subprogram P_1 of Π undergoes a modification during the rewriting is through a collapse operation, which, by definition, does not "filter" out any answer sets of the modified program. Since coherence is preserved by Theorem 1, a quantified answer set of Π can be obtained from a quantified answer set of Π' by projecting out atoms that are not in HB_{P_1} (i.e., those not in the "original" P_1).

6 Complexity issues

In this section, we investigate the complexity of problems related to ASP$^{\omega}$(Q) programs. We first study the complexity of the coherence problem. For that problem, global constraints can be ignored. Interestingly, the presence of local constraints leads to some unexpected phenomena. Next, we study the complexity of problems concerning membership of atoms in optimal answer sets. For this study, we restrict attention to existential programs with only global constraints.

Theorem 2 (Upper bound)
The coherence problem of an ASP$^{\omega}$(Q) program Π is in: (i) Σ^p_{n+1} for existential programs, and (ii) Π^p_{n+1} for universal programs, where $n = nQuant(\Pi)$.

Proof
Let Π' be the result of applying Algorithm 1 to Π. Then, Π' is a quantifier-alternating plain program with at most $n = nQuant(\Pi) + 1$ quantifiers that is coherent iff Π is coherent (Theorem 1). Thesis follows from Theorem 3 in the paper by Amendola et al., (2019). □

Theorem 3 (Lower bound)
The coherence problem of an ASP$^{\omega}$(Q) program is hard for (i) Σ^p_n for existential programs, and hard for (ii) Π^p_n for universal programs, where $n = nQuant(\Pi)$.

The result above follows trivially from the observation that any quantifier-alternating ASP(Q) program with n quantifiers is a plain ASP$^{\omega}$(Q) program where $C^w = \emptyset$.

The lower and upper bounds offered by the two previous results do not meet in the general case. However, for some classes of ASP$^{\omega}$(Q) programs they do, which leads to completeness results. For instance, note that Algorithm 1 produces a quantifier-alternating plain ASP(Q) program with at most n quantifiers when the last subprogram is plain.

Corollary 3.1 (First completeness result)
The coherence problem of an ASP$^{\omega}$(Q) program where the last subprogram is plain (i.e., $\mathcal{W}(P_n) = \emptyset$) is (i) Σ^p_n-complete for existential programs, and (ii) Π^p_n-complete for universal programs, where $n = nQuant(\Pi)$.

Proof. (Sketch)
The assertion follows from Theorem 3 in the paper by Amendola *et al.* (2019) and from properties of Algorithm 1. □

Note that, in plain ASP(Q) (as well as in related formalisms such as those considered by Stockmeyer (1976) and Fandinno *et al.* (2021)), the complexity of coherence correlates directly with the number of quantifier alternations (Amendola *et al.* 2019). Perhaps somewhat unexpectedly at first glance, it is not the case of ASP$^{\omega}$(Q). There, when local constraints are present, one can "go up" one level with two consecutive quantifiers of the same kind. This phenomenon is exemplified below.

Theorem 4 (Second completeness results)
Deciding coherence of uniform existential ASP$^{\omega}$(Q) programs with two quantifiers (i.e., $n = 2$) such that P_2 is not plain is Σ^p_2-complete.

Proof. (Sketch)
(Membership) By applying Algorithm 1 on a uniform existential ASP$^{\omega}$(Q) programs with two quantifiers where the program P_2 is not plain, we obtain an equi-coherent ASP(Q) of

the form $\exists P_1' \forall P_2' : C'$. Thus, the membership to Σ_2^P follows from Theorem 3 of Amendola *et al.* (2019). $\square$

Hardness is proved by a reduction of an existential 2QBF in DNF by adapting the QBF encoding in ASP(Q) from Theorem 2 of Amendola *et al.* (2019). In detail, a weak constraint in P_2 simulates the forall quantifier by preferring counterexamples that are later excluded by the final constraint C.

The proof offers insights into this phenomenon, revealing that the second quantifier, governing optimal answer sets, essentially "hides" a universal quantifier. The following result closes the picture for uniform plain programs with two existential quantifiers.

Proposition 1 (Third completeness results)
Deciding coherence of plain uniform $\mathrm{ASP}^\omega(\mathrm{Q})$ programs with 2 quantifiers is (i) NP-complete for existential programs; and (i) coNP-complete for universal programs.

The result follows trivially from Lemma 1, once we observe that one application of col_1 builds an equi-coherent program with one quantifier.

We will now turn our attention to problems involving optimal quantified answer sets.

Observe that, as for the case of plain ASP, verifying the existence of an optimal quantified answer set has the same complexity as verifying the existence of a quantified answer set. Indeed, if a quantified answer set exists, there is certainly an optimal one. Thus, a more interesting task is to verify *whether an atom a belongs to some optimal quantified answer sets*. (This is important as it supports brave reasoning as well as allows one to compute an optimal quantified answer set, if one exists).

We will now study this problem for *plain* $\mathrm{ASP}^\omega(\mathrm{Q})$ programs with global constraints that seem to be especially relevant to practical applications. Similarly to what was proved by Buccafurri *et al.* (2000) for ASP, the task in question results in a jump in complexity. Specifically, it elevates the complexity to being complete for Δ_n^P in the general case.

Theorem 5 (Fourth completeness results)
Deciding whether an atom a belongs to an optimal quantified answer set of a plain alternating existential $\mathrm{ASP}^\omega(\mathrm{Q})$ program with n quantifiers is Δ_{n+1}^P-complete.

Proof. (Sketch)
(Hardness) Hardness can be proved by resorting the observations used in the proof by Buccafurri et al., (2000). More precisely, let $X_1, \ldots, X_n$ be disjoint sets of propositional variables, and Φ be a QBF formula of the form $\forall X_2 \exists X_3 \ldots \mathcal{Q} X_n\ \phi$, where each $\mathcal{Q} \in \{\exists, \forall\}$, and ϕ is a formula over variables in $X_1, \ldots, X_n$ in 3-DNF if n is even, otherwise it is in 3-CNF, and $X_1 = \{x_1, \ldots, x_m\}$. Deciding whether the lexicographically minimum truth assignment τ of variables in X_1, such that $\forall X_2 \exists X_3 \ldots \mathcal{Q} X_n\ \phi_\tau$ is satisfied (assuming such τ exists), satisfies the condition $\tau(x_m) = \top$ is a Δ_{n+1}^P-complete problem (Krentel 1992). Such a problem can be encoded as a plain alternating $\mathrm{ASP}^\omega(\mathrm{Q})$ program Π with n quantifiers such that an atom x_m appears in some optimal quantified answer set of Π if and only if the answer to the problem is "yes". $\square$

(Membership) As observed by Buccafurri *et al.* (2000) and Simons *et al.* (2002), an optimal solution can be obtained with binary search on the value of maximum possible cost, namely k. Since k can be exponential in the general case, then an optimal quantified answer set of Π can be obtained with a polynomial number of calls to the oracle in Σ_n^P, with $n = nQuant(\Pi)$. Finally, an extra oracle call checks that the atom a appears in some optimal quantified answer sets.

Another interesting result regards a specific class of plain ASP$^\omega$(Q) programs, namely those programs in which there is only one level and all the weights are the same. In this particular case the complexity lowers to Θ_{n+1}^P. Recall that, Θ_{n+1}^P is the class of problems that can be solved by a logarithmic number of calls to an oracle in Σ_n^P, that in the literature is also denoted by $\Delta_{n+1}^p[O(log\ m)]$ (Wagner 1986). The next result shows ASP$^\omega$(Q) can optimally encode optimization problems in this complexity class (Wagner 1986, 1990), such as the Propositional Abduction Problem discussed in Section 4.

Theorem 6 (Fifth completeness results)
Deciding whether an atom a belongs to an optimal quantified answer set of a plain alternating existential ASP$^\omega$(Q) program with n quantifiers is Θ_{n+1}^P-complete if there is only one level and all the weights are the same.

Proof
(Hardness) Let a QBF formula Φ be an expression of the form $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \phi$, where $X_1, \dots, X_n$ are disjoint sets of propositional variables, $\mathcal{Q}_i \in \{\exists, \forall\}$ for all $1 \leq i \leq n$, $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$ for all $1 \leq i < n$, and ϕ is a 3-DNF formula over variables in $X_1, X_2, \dots, X_n$ of the form $D_1 \vee \dots \vee D_n$, where each conjunct $D_i = l_1^i \wedge l_2^i \wedge l_3^i$, with $1 \leq i \leq n$. A k-existential QBF formula Φ is a QBF formula where $n = k$ and $\mathcal{Q}_1 = \exists$.

Given a sequence of m k-existential QBF formulas $\Phi_1, \dots, \Phi_m$, with k being even and greater than or equal to 2, and such that if Φ_j is unsatisfiable then also Φ_{j+1} is unsatisfiable, where $1 \leq j < m$, deciding whether $v(\Phi_1, \dots, \Phi_m) = max\{j \mid 1 \leq j \leq m \wedge \Phi_j\ is\ satisfiable\}$ is odd is Θ_{k+1}-complete (Buccafurri *et al.* 2000).

The above problem can be encoded into an ASP$^\omega$(Q) program Π such that a literal, namely *odd*, appears in some optimal quantified answer set of Π if and only if $v(\Phi_1, \dots, \Phi_m)$ is odd. For simplicity, we introduce notation for some sets of rules that will be used in the construction of Π. More precisely, given a QBF formula Φ, $sat(\Phi)$ denotes the set of rules of the form $sat_\Phi \leftarrow l_1^i, l_2^i, l_3^i$, where $D_i = l_1^i \wedge l_2^i \wedge l_3^i$ is a conjunct in ϕ; whereas for a set of variables $X_i = \{x_1^i, \dots, x_n^i\}$ in Φ, and an atom a, $choice(X_i, a)$ denotes the choice rule $\{x_1^i; \dots; x_n^i\} \leftarrow a$. We are now ready to construct the program Π.

First of all, we observe that all the formulas $\Phi_1, \dots, \Phi_m$ have the same alternation of quantifiers. Thus, there is a one-to-one correspondence between the quantifiers in the QBF formulas and those in Π. Let Π be of the form $\Box_1 P_1 \Box_2 P_2 \dots \Box_k P_k : C : C^w$ where $\Box_i = \exists$ if $\mathcal{Q}_i = \exists$ in a formula Φ_j, otherwise $\Box_i = \forall$. The program P_1 is of the form

$$P_1 = \left\{ \begin{array}{rcll} \{solve(1); \dots; solve(m)\} = 1 & \leftarrow & & \\ unsolved(i) & \leftarrow & solve(j) & \forall\ j, i \in [1, \dots, m] s.t.\ i > j \\ odd & \leftarrow & solve(j) & \forall j \in [1, \dots, m] s.t.\ j\ is\ odd \\ choice(X_1^j, solve(j)) & & & \forall 1 \leq j \leq m \end{array} \right\},$$

while, for each $2 \leq i \leq k$, the program P_i is of the form

$$P_i = \left\{ choice(X_i^j, solve(j)) \quad \forall 1 \leq j \leq m \right\},$$

where each X_i^j denotes the set of variables appearing in the scope of the i-th quantifier of the j-th QBF formula Φ_j. Finally, the programs C and C^w are of the form

$$C = \left\{ \begin{array}{rll} & sat(\Phi_j) & \forall 1 \leq j \leq m \\ \leftarrow & solve(j),\ {\sim} sat_{\Phi_j} & \forall 1 \leq j \leq m \end{array} \right\}$$

$$C^w = \left\{ \leftarrow_w unsolved(i)\ [1@1, i] \quad \forall 1 \leq i \leq m \right\}.$$

Intuitively, the first choice rule in P_1 is used to guess one QBF formula, say Φ_j, among the m input ones, for which we want to verify the satisfiability. The guessed formula is encoded with the unary predicate *solve*, whereas, all the following formulas Φ_i, with $i > j$, are marked as unsolved by means of the unary predicate *unsolved*.

Then, P_1 contains different rules of the form $odd \leftarrow solve(j)$ for each odd index j in $[1, m]$. Thus the literal *odd* is derived whenever a QBF formula Φ_j in the sequence $\Phi_1, \ldots, \Phi_m$ is selected (i.e., $solve(j)$ is true) and j is odd. The remaining part of P_1 shares the same working principle of the following subprograms P_i, with $i \geq 2$. More precisely, for each QBF formula Φ_j in the sequence $\Phi_1, \ldots, \Phi_m$, they contain a choice rule over the set of variables quantified by the i-th quantifier of Φ_j. Note that the atom $solve(j)$ in the body of these choice rules guarantees that only one gets activated, and so the activated choice rule guesses a truth assignment for the variables in the i-th quantifier of Φ_j. Similarly, the constraint program C contains, for each QBF formula Φ_j in the sequence $\Phi_1, \ldots, \Phi_m$, (i) a set of rules that derives an atom sat_{ϕ_j} whenever the truth assignment guessed by the previous subprograms satisfies ϕ_j, and (ii) a strong constraint imposing that is not possible that we selected the formula Φ_j (i.e., $solve(j)$ is true) and ϕ_j is violated (i.e., sat_{Φ_j} is false). Thus, there exists a quantified answer set of Π if and only if there exists a formula Φ_j in the sequence $\Phi_1, \ldots, \Phi_m$ such that Φ_j is satisfiable. Since the program C^w contains the set of weak constraints of the form $\leftarrow_w unsolved(j)\ [1@1, j]$ for each $j \in [1, \ldots, m]$, the cost of each quantified answer set is given by the index j of the selected formula. Thus, by minimizing the number of unsolved formulas we are maximizing the index of the satisfiable formula Φ_j. Thus, an optimal quantified answer set corresponds to a witness of coherence for a formula Φ_j, s.t. for each $\Phi_{j'}$, with $j' > j$, $\Phi_{j'}$ is unsatisfiable. By construction *odd* is derived whenever j is odd and so the hardness follows.

(Membership) According to Theorem 3 of Amendola *et al.* (2019), we know that the coherence of an existential plain alternating program with n quantifiers falls within the complexity class Σ_n^P-complete. By following an observation employed in the proofs by Buccafurri *et al.* (2000), the cost of an optimal solution can be obtained by binary search that terminates in a logarithmic, in the value of the maximum cost, number of calls to an oracle in Σ_n^P that checks whether a quantified answer set with a lower cost with respect to the current estimate of the optimum exists. Once the cost of an optimal solution is determined, one more call to the oracle (for an appropriately modified instance), allows one to decide the existence of an optimal solution containing a. Since each weak constraint has the same weight and the same level, then we can consider as the maximum cost the number of weak constraint violations. Thus, the number of oracle calls is at most logarithmic in the size of the problem and the membership follows. □

7 Related work

Disjunctive ASP programs can be used to model problems in the second level of the PH using programming techniques, such as *saturation* (Eiter and Gottlob 1995b; Dantsin *et al.* 2001), but it is recognized that they are not intuitive. As a consequence, many language extensions have been proposed that expand the expressivity of ASP (Bogaerts *et al.* 2016; Amendola *et al.* 2019; Fandinno *et al.* 2021). This paper builds on one of these, namely: Answer Set Programming with Quantifiers (ASP(Q)) (Amendola *et al.* 2019). ASP(Q) extends ASP, allowing for declarative and modular modeling of problems of the entire PH (Amendola *et al.* 2019). We expand ASP(Q) with weak constraints to be able to model combinatorial optimization problems. In Section 4, we show the efficacy of ASP$^\omega$(Q) in modeling problems that would require cumbersome ASP(Q) representations.

The two formalisms most closely related to ASP(Q) are the stable–unstable semantics (Bogaerts *et al.* 2016), and quantified answer set semantics (Fandinno *et al.* 2021). We are not aware of any extension of these that support explicitly weak constraints or alternative optimization constructs.Amendola *et al.* (2019) and Fandinno et al. (2021) provided an exhaustive comparison among ASP extensions for problems in the PH.

It is worth observing that ASP$^\omega$(Q) extends ASP(Q) by incorporating weak constraints, a concept originally introduced in ASP for similar purposes (Buccafurri *et al.* 2000). Clearly, ASP$^\omega$(Q) is a strict expansion of ASP, indeed it is easy to see that any ASP program P is equivalent to a program of the form (3) with only one existential quantifier, where $P_1 = \mathcal{R}(P)$, $C^w = \mathcal{W}(P)$, $C = \emptyset$. Related to our work is also a formalism that has been proposed for handling preferences in ASP, called *asprin* (Brewka *et al.* 2023). *asprin* is effective in defining preferences over expected solutions, nonetheless, the complexity of main reasoning tasks in *asprin* is at most Σ_3^P (Brewka *et al.* 2023), with optimization tasks belonging at most to Δ_3^P (Brewka *et al.* 2023); thus, in theory, ASP$^\omega$(Q) can be used to model more complex optimization problems (unless $\mathrm{P} = \mathrm{NP}$).

8 Conclusion

We proposed an extension of ASP(Q) enabling the usage of weak constraints for expressing complex problems in Δ_n^p, called ASP$^\omega$(Q). We demonstrated ASP$^\omega$(Q)'s modeling capabilities providing suitable encodings for well-known complex optimization problems. Also, we studied complexity aspects of ASP$^\omega$(Q), establishing upper and lower bounds for the general case, and revealing intriguing completeness results. Future work involves tightening the bounds from Theorem 2 for arbitrary n, extending ASP$^\omega$(Q) to support subset minimality, and design a complexity-aware implementation for ASP$^\omega$(Q) based on the translation of Section 5 and extending the system PyQASP (Faber *et al.* 2023).

Supplementary material

The supplementary material for this article can be found at http://dx.doi.org/10.1017/S1471068424000395.

References

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

Amendola, G., Ricca, F. and Truszczynski, M. 2019. Beyond NP: Quantifying over answer sets. *Theory and Practice of Logic Programming* 19, 5-6, 705–721.

Bogaerts, B., Janhunen, T. and Tasharrofi, S. 2016. Stable-unstable semantics: Beyond NP with normal logic programs. *Theory and Practice of Logic Programming* 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.

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

Buccafurri, F., Leone, N. and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering* 12, 5, 845–860.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. ASP-core-2 input language format. *Theory and Practice of Logic Programming* 20, 2, 294–309.

Cao, F., Du, D.-Z., Gao, B., Wan, P.-J. and Pardalos, P. M. (1995) *Minimax Problems in Combinatorial Optimization.* Boston, MA, pp. 269–292

Ceri, S., Gottlob, G. and Tanca, L. (1990) Logic Programming and atabases, *Surveys in Computer Science*

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

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

Eiter, T. and Gottlob, G. 1995b. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15b, 3-4, 289–323.

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

Faber, W., Pfeifer, G. and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175, 1, 278–298.

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

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

Ko, K. I. and Lin, C.-L. 1995. *On the complexity of min-max optimization problems and their approximation*, Boston, MA, pp. 219–239.

Krentel, M. W. 1992. Generalizations of Opt P to the polynomial hierarchy. *Theoretical Computer Science* 97, 2, 183–198.

Marek, V. W. and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In The Logic Programming Paradigm - A 25-Year Perspective, 375–398.

Morgan, C. G. 1971. Hypothesis generation by machine. *Artificial Intelligence* 2, 2, 179–187.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3-4, 241–273.

Pople, H. E. 1973. On the mechanization of abductive logic. In *Proceedings of IJCAI*, 147–152

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

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138, 1-2, 181–234.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theoretical Computer Science* 3, 1, 1–22.

Wagner, K. W. 1986. More complicated questions about maxima and minima, and some closures of NP. In ICALP, vol. 226 of Lecture Notes in Computer Science, 434–443.

Wagner, K. W. 1990. Bounded query classes. *SIAM Journal on Computing* 19, 5, 833–846.