Evaluating Epistemic Logic Programs via Answer Set Programming with Quantifiers

Wolfgang Faber[1], Michael Morak[1]

[1]University of Klagenfurt, Klagenfurt, Austria

Abstract
In this paper we introduce a simple way to evaluate epistemic logic programs by means of answer set programming with quantifiers, a recently proposed extension of answer set programming. The method can easily be adapted for most of the many semantics that were proposed for epistemic logic programs. We evaluate the proposed transformation on existing benchmarks using a recently proposed solver for answer set programming with quantifiers, which relies on QBF solvers.

Keywords
Epistemic Logic Programs, Answer Set Programming with Quantifiers, Answer Set Programming

1. Introduction

Answer Set Programming (ASP) is a generic, fully declarative logic programming language that allows users to encode problems such that the resulting output of the program (called *answer sets*) directly corresponds to solutions of the original problem [1, 2, 3, 4].

Epistemic Logic Programs (ELPs) are an extension of ASP with epistemic operators. Originally introduced as the two modal operators **K** ("known" or "provably true") and **M** ("possible" or "not provably false") by Gelfond [5, 6], epistemic extensions of ASP have received renewed interest (c.f. e.g. [7, 8, 9, 10, 11, 12, 13, 14, 15]), with refinements of the semantics and proposals of new language features. Further, the development of efficient solving systems is underway with several working systems now available [16, 12, 17].

Recently, another language extension for ASP has been proposed, named ASP with Quantifiers, or ASP(Q) [18], which introduces an explicit way to express quantifiers and quantifier alternations in ASP, in a similar way as in Quantified Boolean Formulas (QBFs). It has the form

$$\Box_1 P_1 \Box_2 P_2 \cdots \Box_n P_n : C,$$

where the P_1, P_2, ...are classical ASP programs, $\Box_1$, $\Box_2$, ...are quantifiers, and C expresses a set of constraints in ASP.

Example 1. *The intuitive reading of the ASP(Q) program*

$$\exists^{st} P_1 \forall^{st} P_2 \cdots P_n : C$$

RCRA'22
wolfgang.faber@aau.at (W. Faber); michael.morak@aau.at (M. Morak)
https://www.aau.at/team/faber-wolfgang/ (W. Faber); https://www.aau.at/team/morak-michael/ (M. Morak)
0000-0002-0330-5868 (W. Faber); 0000-0001-7116-9338 (M. Morak)

is that there exists an answer set A_1 of ASP program P_1 such that for each answer set A_2 of the ASP program P_2 combined with A_1 ... such that the ASP program C combined with A_n is coherent (has an answer set).

Since evaluating ASP(Q) programs is PSPACE-complete in general [18], this formalisms forms an interesting target for rewriting ELPs. In this paper, we propose such a rewriting, where, given an ELP Π, we create an ASP(Q) program that is consistent if and only if Π has a world view. This happens by using the ASP(Q) quantifiers to directly encode the semantics of world views of ELPs, and, in turn, the existence the relevant stable models inside of that world view. We then experimentally verify that this encoding, together with a QBF solver-based ASP(Q) solver, indeed performs well, compared to current ELP solvers.

Contributions. The results and contributions presented in this paper can be summarized as follows.

- We specify a rewriting from ELPs to ASP(Q) programs in such a way that preserves consistency. This rewriting also allows us to extract information about the world views of the original ELPs by evaluating the outermost quantor of the obtained ASP(Q) program.
- We implement a rewriting tool that performs our rewriting on real-world ELP instances.
- We compare the performance of evaluating ELPs via our rewriting tool and state-of-the-art ASP(Q) solvers versus several existing ELP solvers. We observe that, indeed, our ASP(Q) rewriting approach shows competitive performance.

Related Work. Most ELP solvers build upon an underlying ASP solver that is called multiple times in a procedural, sequential manner [19]. The *selp* system [17] follows a similar approach as the one proposed in this paper, since it tries to rewrite an ELP into a non-ground ASP program with fixed arity in order to then use a single call to an ASP solver to establish the consistency of the input ELP. In this work, we try to follow a similar approach by rewriting ELPs to the language of ASP(Q). This is due to the fact that ASP(Q) allows one to easily express the quantification that is needed to write an intuitive encoding of the ELP semantics. However, other target languages that follow a similar idea would be available, including the stable-unstable semantics [20], for which a solver implementation has recently been proposed [21], but also the language of Quantified ASP [22], which follows a similar approach as ASP(Q), but does not quantify over answer sets but over atoms. We found the language of ASP(Q) to be very intuitive to use in practice, as well as having several solver implementations available, and hence chose this as our target language in this paper.

Structure. The remainder of the paper is structured as follows. In Section 2, we provide an overview of ELPs and ASP(Q). In Section 3, we present our rewriting of ELPs to ASP(Q). We then perform an experimental evaluation in Section 4. We conclude with a summary in Section 5.

2. Preliminaries

Answer Set Programming (ASP). We assume the reader is familiar with ASP and will only give a very brief overview of the core language. For more information, we refer to standard literature [3, 23, 24], and, in our case, the ASP-Core-2 input language format [25].

Briefly, ASP programs consist of sets of *rules* of the form

$$a_1 \mid \cdots \mid a_n \leftarrow b_1, \dots, b_\ell, \neg b_{\ell+1}, \dots, \neg b_m.$$

In these rules, all a_i and b_i are *atoms* of the form $p(t_1, \dots, t_n)$, where p is a predicate name, and $t_1, \dots, t_n$ are terms, that is, either variables or constants. A *literal* is either an atom or a negated atom. The domain of constants in an ASP program P is given implicitly by the set of all constants that appear in it. Generally, before evaluating an ASP program, variables are removed by a process called *grounding*, that is, for every rule, each variable is replaced by all possible combination of constants, and appropriate ground copies of the rule are added to the resulting program $ground(P)$. In practice, several optimizations have been implemented in state-of-the-art grounders that try to minimize the size of the grounding.

The semantics of a (ground) ASP program P is as follows [2]. An *interpretation* I (i.e., a set of ground atoms appearing in P) is called a *model* of P iff it satisfies all the rules in P in the sense of classical logic. It is further called an *answer set* of P iff there is no proper subset $I' \subset I$ that is a model of the so-called reduct P^I of P w.r.t. I. P^I is defined as the set of rules obtained from P where all negated atoms on the right-hand side of the rules are evaluated over I and replaced by $\top$ or $\bot$ accordingly. The main decision problem for ASP is deciding whether a program has at least one answer set. This has been shown to be Σ_P^2-complete [26]. The set of answer sets of an ASP program P is denoted $AS(P)$.

In this paper, we will make limited use of choice rules as defined in [25] that look like $\{a\}$. and mean that atom a can be assumed to be true or not.

Answer Set Programming with Quantifiers (ASP(Q)). An extension of ASP, referred to as ASP(Q), has been proposed in [18], providing a formalism reminiscent of Quantified Boolean Formulas, but based on ASP. An ASP(Q) program is of the form

$$\Box_1 P_1 \Box_2 P_2 \cdots \Box_n P_n \, : \, C,$$

where, for each $i \in \{1, \dots, n\}$, $\Box_i \in \{\exists^{st}, \forall^{st}\}$, P_i is an ASP program, and C is a stratified normal ASP program (this is, as intended by the ASP(Q) authors, a "check" in the sense of constraints). $\exists^{st}$ and $\forall^{st}$ are called *existential* and *universal answer set quantifiers*, respectively.

The intuitive reading of an ASP(Q) program $\exists^{st} P_1 \forall^{st} P_2 \cdots P_n \, : \, C$ is that there exists an answer set A_1 of P_1 such that for each answer set A_2 of P_2 extended by A_1 ... such that C extended by A_n is coherent (has an answer set).

Let us be more precise about a program P being extended by an answer set, or rather interpretation I: For an interpretation I, let $f_P(I)$ be the ASP program that contains all true atoms in I as facts and all false atoms in I w.r.t. the Herbrand base of ASP program P as constraints (i.e. rules of the form $\bot \leftarrow a$, for some atom a). Furthermore, for a program P and an interpretation I, let $f_P(\Pi, I)$ be the ASP(Q) program obtained from an ASP(Q) Π by replacing the first program P_1 in Π with $P_1 \cup f_P(I)$. *Coherence* of an ASP(Q) program is then defined inductively:

- $\exists^{st} P : C$ is coherent if there exists an answer set M of P such that $C \cup f_P(M)$ has at least one answer set.
- $\forall^{st} P : C$ is coherent if for all answer sets M of P it holds that $C \cup f_P(M)$ has at least one answer set.
- $\exists^{st} P \Pi$ is coherent if there exists an answer set M of P such that $f_P(\Pi, M)$ is coherent.
- $\forall^{st} P \Pi$ is coherent if for all answer sets M of P it holds that $f_P(\Pi, M)$ is coherent.

In addition, for an existential ASP(Q) program Π (one that starts with $\exists^{st}$), the witnessing answer sets of the first ASP program P_1 are referred to as *quantified answer sets*.

In general, deciding coherence for an ASP(Q) program is known to be PSpace-complete [18, Theorem 2], and on the n-th level of the polynomial hierarchy for programs with n quantifier alternations [18, Theorem 3].

Epistemic Logic Programming (ELP). An *epistemic literal* is a formula $\mathbf{K}\,\ell$ or $\mathbf{M}\,\ell$, where ℓ is a literal and $\mathbf{K}$ and $\mathbf{M}$ are the epistemic operators of *certainty* ("known" or "provably true") and *possibility* ("maybe," "possible," or "not provably false"). An *epistemic logic program (ELP)* is a tuple $\Pi = (\mathscr{A}, \mathscr{R})$, where $\mathscr{A}$ is a set of propositional atoms and $\mathscr{R}$ is a set of rules of the following form:

$$a_1 \vee \cdots \vee a_k \leftarrow \ell_1, \ldots, \ell_m, \xi_1, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n,$$

where $k \geqslant 0$, $m \geqslant 0$, $n \geqslant j \geqslant 0$, each $a_i \in \mathscr{A}$ is an atom, each ℓ_i is a literal, and each ξ_i is an epistemic literal, where the latter two each use an atom from $\mathscr{A}$. Such rules are also called *ELP rules*. Similar to ASP, we consider an ELP *ground*, if no variables appear in it, and treat programs with variables as an abbreviation of the ground program, where each variable is replaced with every possible constant from the program.

Let $H(r) = \{a_1, \ldots, a_k\}$ denote the head elements of an ELP rule, and let $B(r) = \{\ell_1, \ldots, \ell_m, \xi_1, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n\}$, that is, the set of elements appearing in the rule body. The *union* (or *combination*) of two ELPs $\Pi_1 = (\mathscr{A}_1, \mathscr{R}_1)$ and $\Pi_2 = (\mathscr{A}_2, \mathscr{R}_2)$ is the ELP $\Pi_1 \cup \Pi_2 = (\mathscr{A}_1 \cup \mathscr{A}_2, \mathscr{R}_1 \cup \mathscr{R}_2)$. The set of epistemic literals occurring in an ELP Π is denoted $elit(\Pi)$.

The semantics of ELPs are given via *world views*. A world view of an ELP Π has originally [5, 6] been defined as a set of (ASP) interpretations $\mathscr{I} = \{I_1, \ldots I_n\}$, such that $\mathscr{I} = AS(\Pi^{\mathscr{I}})$, where $\Pi^{\mathscr{I}}$ is called the *epistemic reduct* of Π w.r.t. $\mathscr{I}$, and is obtained from Π by (1) removing all rules from Π that contain an epistemic literal which is negated but true in $\mathscr{I}$, or unnegated and false in $\mathscr{I}$; and (2) removing all epistemic literals from the remaining rules. Here, $\mathbf{K}\,\ell$ is true in $\mathscr{I}$, if and only if ℓ is true in all interpretations in $\mathscr{I}$, and $\mathbf{M}\,\ell$ is true in $\mathscr{I}$, if and only if ℓ is true in at least one interpretation in $\mathscr{I}$.

Various other semantics have been defined over the years and most of them can be characterized by reducts, either over sets of interpretations (as above) or over the set of $elit(\Pi)$ that are satisfied by $\mathscr{I}$. A concise summary can be found in [19].

The main reasoning task for ELPs is checking whether they are *consistent*, that is, whether they have a world view. This problem is also referred to as *world view existence problem* and is known to be Σ_3^P-complete [11].

3. Transformation

The basic idea of our transformation is to "guess" a set of epistemic literals that are true and represent them as standard atoms, and verify that a world view exists that is defined precisely by this guess. This can be done since a world view is uniquely determined by the set of epistemic literals that it entails [15].

In order to verify that a world view for the guessed epistemic literals actually exists, several existential and universal ASP(Q) sub-programs, corresponding to the epistemic reduct, are created that check (a) for epistemic literals ξ of type $\mathbf{M}\, a$ and $\neg\mathbf{K}\, a$ the existence of an answer set verifying the truth of ξ via an existential program for each such ξ and (b) for epistemic literals ξ of type $\mathbf{K}\, a$ and $\neg\mathbf{M}\, a$ the non-existence of an answer set contradicting the truth of ξ via a single universal program. Finally, a set of constraints check that each of these witness answer sets is consistent with the guessed set of epistemic literals.

We first illustrate this idea with an example.

Example 2. *The ELP program*

$$a \leftarrow \mathbf{K}\,\neg b$$
$$b \leftarrow \mathbf{K}\,\neg a$$

has two world views $\{\{a\}\}$ *and* $\{\{b\}\}$. *An ASP(Q) program that implements the idea outlined above would be* $\Pi = \exists^{st} P_1 \exists^{st} P_2 \forall^{st} P_3 \ : \ C$ *with* P_1 *being*

$$\{kna\}.$$
$$\{knb\}.$$

P_2 *being*

$$a' \leftarrow knb.$$
$$b' \leftarrow kna.$$

P_3 *being*

$$a'' \leftarrow knb.$$
$$b'' \leftarrow kna.$$

and C *consisting of*

$$\leftarrow \neg kna, \neg a'.$$
$$\leftarrow \neg knb, \neg b'.$$
$$\leftarrow kna, a''.$$
$$\leftarrow knb, b''.$$

Here, P_1 *has four answer sets, which serve as representations of the potential world views.* P_2 *and* P_3 *are used to determine answer sets of the epistemic reduct, and* C *makes sure that these answer sets are consistent with the potential world view of* P_1. *In the example, there are two quantified answer sets of* Π, $\{kna\}$ *and* $\{knb\}$.

There is one issue that does not become apparent in this example, though: in general, different epistemic literals may require different existential witnesses. This can best seen by another example.

Example 3. *Consider the ELP program*

$$a \mid b$$
$$c \leftarrow \mathbf{M}\, a, \mathbf{M}\, b$$

This program has one world view $\{\{a, c\}, \{b, c\}\}$*. Following the pattern of the previous example, one would come up with* $\Pi' = \exists^{st} P_1' \exists^{st} P_2' \forall^{st} P_3' : C'$ *with* P_1' *being*

$$\{ma\}.$$
$$\{mb\}.$$

P_2' *being*

$$a' \mid b'.$$
$$c' \leftarrow ma, mb.$$

P_3' *being*

$$a'' \mid b''.$$
$$c'' \leftarrow ma, mb.$$

and C' *consisting of*

$$\leftarrow ma, \neg a'.$$
$$\leftarrow mb, \neg b'.$$
$$\leftarrow \neg ma, a''.$$
$$\leftarrow \neg mb, b''.$$

But Π' *is not coherent, so there are no quantified answer sets. The reason is that only one answer set of* P_2' *will be considered (so either* a' *is true and* b' *is false or* a' *is false and* b' *is true), while for* ma *we need the answer set with* a' *being true and for* mb *we need the answer set with* b' *being true.*

To get this behaviour, we can introduce two copies of P_2'*, getting* $\Pi'' = \exists^{st} P_1' \exists^{st} P_2^{ma} \exists^{st} P_2^{mb} \forall^{st} P_3' :$ C'' *with* P_1' *and* P_3' *remaining unchanged and* P_2^{ma} *being*

$$a^{ma} \mid b^{ma}.$$
$$c^{ma} \leftarrow ma, mb.$$

and P_2^{mb} *being*

$$a^{mb} \mid b^{mb}.$$
$$c^{mb} \leftarrow ma, mb.$$

and C'' *consisting of*

$$\leftarrow ma, \neg a^{ma}.$$
$$\leftarrow mb, \neg b^{mb}.$$
$$\leftarrow \neg ma, a''.$$
$$\leftarrow \neg mb, b''.$$

It can be checked that Π'' *has one quantified answer set* $\{ma, mb\}$*.*

3.1. Formalization

Given an ELP Π, we create an ASP(Q) program $\Pi^\tau = \exists^{st} P_1 \exists^{st} P_2^{s_1} \cdots \exists^{st} P_2^{s_n} \forall^{st} P_3 \; : \; C$, where P_1 consists of one choice rule containing one fresh atom $\tau(s_i)$ for each element of $elit(\Pi) = \{s_1, \dots, s_n\}$, $P_2^{s_1}$ to $P_2^{s_n}$ are copies of Π, one for each epistemic literal, where all non-epistemic literals are annotated with $\tau(s_i)$ for the respective s_i and all epistemic literals ℓ are replaced by $\tau(\ell)$, and P_3 is yet another copy of Π annotated with a special symbol *all*. C consists of two constraints for each $s \in elit(\Pi)$, for instance for $\mathbf{K}\, a$ these will be $\leftarrow \tau(\mathbf{K}\, a), \neg a^{all}$ and $\leftarrow \neg\tau(\mathbf{K}\, a), a^{\tau(\mathbf{K}\, a)}$.

The transformation can be quite easily adapted for several other semantics by leveraging different reducts within the programs P_2. However, for the semantics provided by Shen and Eiter [11] more care would have to be taken, as an additional minimization over the epistemic literals chosen in P_1 is required.

4. Experimental Evaluation

We tested the rewriting approach described in Section 3, by benchmarking it against existing ELP solvers. We will refer to our rewriting tool as *elp2qasp*. To compare, we chose the state-of-the-art ELP solver *EP-ASP* [12] and the *selp* solver [17] based on a rewriting to plain ASP. For our rewriting, we used two ASP(Q) solvers as backends: the *qasp* solver [27], as well as the *q_asp* solver [28].

We use the same three test sets proposed in [17]. For every test set, we measured the time it took to solve the consistency problem. For *selp*, the underlying ASP solver *clingo* [29] was stopped after finding the first answer set. For *EP-ASP*, search was terminated after finding the first candidate world view[1]. For *qasp* and *q_asp*, the output of our ELP to ASP(Q) rewriting directly tells us whether the ELP is consistent or not, depending on whether the ASP(Q) program is consistent.

Experiments were run on an AMD EPYC 7601 system (2.2GHz base clock speed) with 500 GiB of memory. Each process was assigned a maximum of 14 GB of RAM, which was never exceeded by any of the solvers tested. A time limit of 900 seconds was used for each benchmark set. For *EP-ASP*, we made trivial modifications to the python code in order for it to run with *clingo* 5.4.1. For *selp* and *qasp*, the same version of *clingo* was used. For *selp*, in addition, we used the *htd* library, version 1.2.0, and *lpopt* 2.2. We used *qasp* 1.1.0 and *q_asp* 0.1.2 as the backend solvers for our ASP(Q) rewriting generated by *elp2qasp*. The time it took to convert input ELP programs into the specific input formats of the various tools we used (e.g. the input format for *selp* or *EP-ASP*) was not measured, since we did not want the input format conversion to influence the benchmark results. *EP-ASP* was called with the preprocessing option for brave and cautious consequences on, since it always ran faster this way. The time for *selp*, *qasp*, and *q_asp* is the sum of the time it took to run all required components. For *selp*, *clingo* was always called with SAT preprocessing enabled, as is recommended by the *lpopt* tool.

[1]Note that to have a fair comparison we disabled the subset-maximality check on the guess that *EP-ASP* performs by default.

4.1. Benchmark Instances

We used three types of benchmarks, two coming from the ELP literature and one from the QSAT domain [2]. This is the same benchmark set as used and published by the authors of the *selp* solver, which they used in the associated conference publication [17]. We briefly describe the benchmark set below.

Scholarship Eligibility. This set of non-ground ELP programs is shipped together with *EP-ASP*. Its instances encode the scholarship eligibility problem for 1 to 25 students.

Yale Shooting. This test set consists of 25 non-ground ELP programs encoding a simple version of the Yale Shooting Problem, a conformant planning problem: the only uncertainty is whether the gun is initially loaded or not, and the only fluents are the gun's load state and whether the turkey is alive. Instances differ in the time horizon. We follow the ELP encoding from [16].

Tree QBFs. The hardness proof for ELP consistency [11] relies on a reduction from the validity problem for restricted quantified boolean formulas with three quantifier blocks (i.e. 3-QBFs), which can be generalized to arbitrary 3-QBFs [17]. In that publication, the reduction is applied to the 14 "Tree" instances of QBFEVAL'16 [30], available at http://www.qbflib.org/family_detail.php?idFamily=56.

4.2. Results

The results for the first two sets are shown in Figure 1 and Figure 2, respectively. For the Scholarship Eligibility Problem, we can observe, confirming the observations by Bichler et al. [17], that *EP-ASP* can solve 16 instances, while *selp* is able to solve all instances, independent of time, within 5 seconds. Interestingly, the same holds true for our *elp2qasp* rewriting, when using *q_asp* as a backend. This combination can solve all instances within 13 seconds. Interestingly, the *qasp* backend is only able to solve three instances successfully within the time limit. The difference in performance between the two tools may be due to the fact that *q_asp* uses a QBF solver, while *qasp* delegates the solving work to the ASP solver *clingo* or *wasp* [31]. In all our benchmarks we use the latter option.

For the Yale Shooting Problem, we can see that both *EP-ASP* and *selp* are unable to solve all instances. Note that all instances of this problem are inconsistent, which sometimes allows *EP-ASP* to realize this fairly quickly. However, in seven cases, we dont get any answers from *EP-ASP* within the time limit. On the other hand, our *elp2qasp* approach with the *q_asp* backend is able to solve all instances of this problem within 70 seconds, with the solving time increasing moderately with the increase in the time horizon. For the *qasp* backend, we unfortunately encountered a problem in the implementation, which lead to an internal error message for all instances of the Yale Shooting Problem.

Finally, for the Tree QBF Problem, we confirmed the results of Bichler et al. [17]: *selp* was able to solve 4 of the 14 instances within the time limit of 900 seconds, whereas both *EP-ASP*

[2]https://drive.google.com/file/d/12lAzaM1wTXomqTniT75C7lWrh5EMJn6r

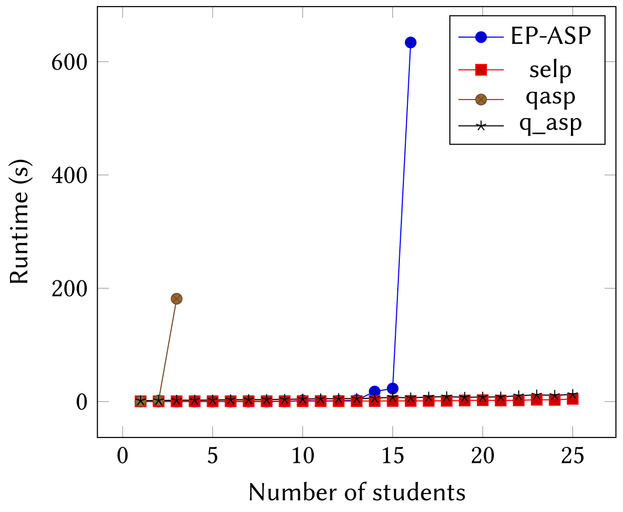

Figure 1: Scholarship Eligibility

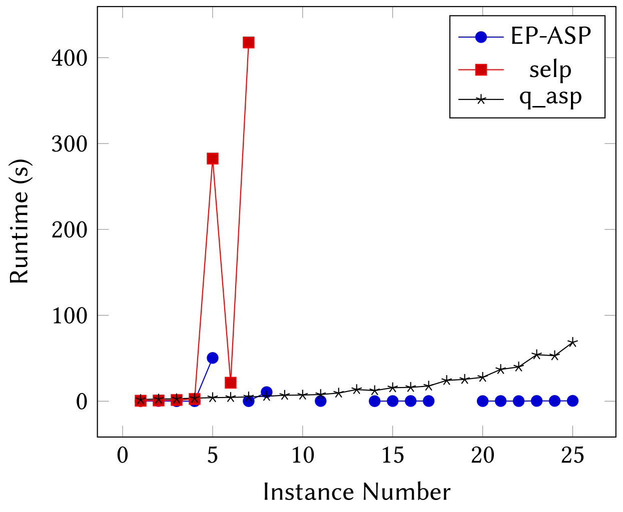

Figure 2: Yale Shooting Problem

and *elp2qasp* with the *qasp* backend were unable to solve any instances at all. For the *q_asp* backend, this class of problems is unsolvable, since the resulting ASP(Q) rewriting contains rules that are not head-cycle free and are hence not treatable by *q_asp*. Since *selp* was the only solver able to successfully solve instances of this problem, we omit a dedicated figure for this problem here.

These results confirm that *elp2qasp* is competitive for solving ELP programs, especially when paired with the *q_asp* solver for ASP(Q), which, in turn, is based on an internal QBF solver. On the other hand, the ASP(Q) solver *qasp* does not seem to match this success, but this may be an inherent limitation, since it internally relies on the ASP solver *clingo* or *wasp* to solve ASP(Q) instances, which may lead to an exponential number of internal calls to that ASP solver.

5. Conclusions

In this paper, we proposed a rewriting that transforms epistemic logic programs (ELPs) into programs for answer set programming with quantifiers (ASP(Q)). It does this by faithfully mimicking the semantics of ELPs and formulating them directly in ASP(Q), which is possible because of the explicit support for quantification that ASP(Q) provides.

We then implement our approach and, using state-of-the-art ASP(Q) solvers as a backend, test our rewriting approach against existing solvers for ELPs. We show that, for several problem domains, our rewriting approach offers competitive performance when compared to existing solvers, especially in the case where the *q_asp* ASP(Q) solver [28] is used, which internally uses a QBF solver to evaluate the given ASP(Q) program.

Future work includes further refining and optimizing the rewriting, as well as adapting it for the various other semantics that exist for evaluating ELPs [9, 10, 11, 12, 13, 14, 15]

References

[1] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. ICLP/SLP, 1988, pp. 1070–1080.

[2] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386. doi:10.1007/BF03037169.

[3] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[4] T. Schaub, S. Woltran, Special issue on answer set programming, Künstliche Intell. 32 (2018) 101–103. doi:10.1007/s13218-018-0554-8.

[5] M. Gelfond, Strong introspection, in: T. L. Dean, K. R. McKeown (Eds.), Proc. AAAI, AAAI Press / The MIT Press, 1991, pp. 386–391. URL: http://www.aaai.org/Library/AAAI/1991/aaai91-060.php.

[6] M. Gelfond, Logic programming and reasoning with incomplete information, Ann. Math. Artif. Intell. 12 (1994) 89–116.

[7] M. Gelfond, New semantics for epistemic specifications, in: Proc. LPNMR, 2011, pp. 260–265.

[8] M. Truszczynski, Revisiting epistemic specifications, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, volume 6565 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 315–333. doi:10.1007/978-3-642-20832-4_20.

[9] P. T. Kahl, Refining the Semantics for Epistemic Logic Programs, Ph.D. thesis, Texas Tech University, Texas, USA, 2014.

[10] L. F. del Cerro, A. Herzig, E. I. Su, Epistemic equilibrium logic, in: Proc. IJCAI, 2015, pp. 2964–2970.

[11] Y. Shen, T. Eiter, Evaluating epistemic negation in answer set programming, Artif. Intell. 237 (2016) 115–135.
[12] T. C. Son, T. Le, P. T. Kahl, A. P. Leclerc, On computing world views of epistemic logic programs, in: Proc. IJCAI, 2017, pp. 1269–1275.
[13] P. T. Kahl, A. P. Leclerc, Epistemic logic programs with world view constraints, in: A. D. Palù, P. Tarau, N. Saeedloei, P. Fodor (Eds.), Proc. ICLP, volume 64 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 1:1–1:17. doi:10.4230/OASIcs.ICLP.2018.1.
[14] W. Faber, M. Morak, S. Woltran, Strong equivalence for epistemic logic programs made easy, in: Proc. AAAI, AAAI Press, 2019, pp. 2809–2816. doi:10.1609/aaai.v33i01.33012809.
[15] M. Morak, Epistemic logic programs: A different world view, in: Proc. ICLP, Technical Communications, volume 306 of *EPTCS*, 2019, pp. 52–64. doi:10.4204/EPTCS.306.11.
[16] P. T. Kahl, R. Watson, E. Balai, M. Gelfond, Y. Zhang, The language of epistemic specifications (refined) including a prototype solver, J. Log. Comput. (2015) exv065. doi:https://doi.org/10.1093/logcom/exv065.
[17] M. Bichler, M. Morak, S. Woltran, selp: A single-shot epistemic logic program solver, Theory Pract. Log. Program. 20 (2020) 435–455. doi:10.1017/S1471068420000022.
[18] G. Amendola, F. Ricca, M. Truszczynski, Beyond NP: quantifying over answer sets, Theory Pract. Log. Program. 19 (2019) 705–721. doi:10.1017/S1471068419000140.
[19] A. P. Leclerc, P. T. Kahl, A survey of advances in epistemic logic program solvers, CoRR abs/1809.07141 (2018). URL: http://arxiv.org/abs/1809.07141. arXiv:1809.07141, also in Proc. of ASPOCP 2018.
[20] B. Bogaerts, T. Janhunen, S. Tasharrofi, Stable-unstable semantics: Beyond NP with normal logic programs, Theory Pract. Log. Program. 16 (2016) 570–586. doi:10.1017/S1471068416000387.
[21] T. Janhunen, Implementing stable-unstable semantics with ASPTOOLS and clingo, in: J. Cheney, S. Perri (Eds.), Proc. PADL, volume 13165 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 135–153. doi:10.1007/978-3-030-94479-7_9.
[22] J. Fandinno, F. Laferrière, J. Romero, T. Schaub, T. C. Son, Planning with incomplete information in quantified answer set programming, Theory Pract. Log. Program. 21 (2021) 663–679. doi:10.1017/S1471068421000259.
[23] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012. doi:10.2200/S00457ED1V01Y201211AIM019.
[24] V. Lifschitz, Answer Set Programming, Springer, 2019. URL: https://doi.org/10.1007/978-3-030-24658-7. doi:10.1007/978-3-030-24658-7.
[25] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. doi:10.1017/S1471068419000450.
[26] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (1995) 289–323. URL: https://doi.org/10.1007/BF01536399. doi:10.1007/BF01536399.
[27] A. Natale, Design and implementation of qasp solver, 2021. doi:10.5281/zenodo.5425783.
[28] G. Amendola, B. Cuteri, F. Ricca, M. Truszczynski, Quantified asp solver q_asp, 2022. URL:

https://github.com/bernardocuteri/q_asp.

[29] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, TPLP 19 (2019) 27–82.

[30] L. Pulina, The ninth QBF solvers evaluation - preliminary report, in: Proc. QBF, 2016, pp. 1–13. URL: http://ceur-ws.org/Vol-1719/paper0.pdf.

[31] M. Alviano, C. Dodaro, N. Leone, F. Ricca, Advances in WASP, in: F. Calimeri, G. Ianni, M. Truszczynski (Eds.), Proc. LPNMR, volume 9345 of *LNCS*, Springer, 2015, pp. 40–54. doi:10.1007/978-3-319-23264-5_5.