selp*: A Single-Shot Epistemic Logic Program Solver**

MANUEL BICHLER, MICHAEL MORAK, and STEFAN WOLTRAN

TU Wien, Vienna, Austria

(*e-mails:* bichler@dbai.tuwien.ac.at, michael.morak@aau.at, woltran@dbai.tuwien.ac.at)

Abstract

Epistemic logic programs (ELPs) are an extension of answer set programming (ASP) with epistemic operators that allow for a form of meta-reasoning, that is, reasoning over multiple possible worlds. Existing ELP solving approaches generally rely on making multiple calls to an ASP solver in order to evaluate the ELP. However, in this paper, we show that there also exists a direct translation from ELPs into non-ground ASP with bounded arity. The resulting ASP program can thus be solved in a single shot. We then implement this encoding method, using recently proposed techniques to handle large, non-ground ASP rules, into the prototype ELP solving system "selp," which we present in this paper. This solver exhibits competitive performance on a set of ELP benchmark instances.

KEYWORDS: logic programming methodology and applications, knowledge representation and nonmonotonic reasoning, technical notes and rapid communications

1 Introduction

Epistemic logic programs (ELPs), as defined in Shen and Eiter (2016), are an extension of the well-established formalism of answer set programming (ASP). ASP is a generic, fully declarative logic programming language that allows us to encode problems in such a way that the resulting answers (called *answer sets*) directly correspond to solutions of the encoded problem (Brewka *et al.* 2011). Negation in ASP is generally interpreted according to the stable model semantics (Gelfond and Lifschitz 1988), that is, as *negation-as-failure* or *default negation*. The default negation $\neg a$ of an atom a is true if there is no justification for a in the same answer set, making it a "local" operator in the sense that it is defined relative to the same answer set. ELPs, on the other hand, extend ASP with the *epistemic negation* operator **not** that allows for a form of meta-reasoning, that is, reasoning over multiple answer sets. Intuitively, an epistemically negated atom **not** a expresses that a cannot be *proven* true, that is, it is false in at least one answer set. Thus, epistemic negation is defined relative to a collection of answer sets, referred to as a *world view*. The main reasoning task for ELPs, checking that a world view exists, is Σ_P^3-complete (Shen and Eiter 2016).

Epistemic negation has long been recognized as a desired construct for ASP (Gelfond 1991, 1994). In these works, Michael Gelfond introduced the modal operators **K**

("known" or "provably true") and **M** ("possible" or "not provably false"), in order to address this need. Given an atom a, **K**a and **M**a stand for $\neg$**not** a and **not** $\neg a$, respectively.

Example 1
A classical example for the use of epistemic negation is the presumption of innocence rule

$$innocent(X) \leftarrow \textbf{not} \; guilty(X),$$

namely, a person is innocent if they cannot be proven guilty.

Renewed interest in recent years has revealed several flaws in the original semantics, and several approaches (Gelfond 2011; Truszczynski 2011; Kahl 2014; del Cerro *et al.* 2015; Shen and Eiter 2016) aimed to refine them in such a way that unintended world views are eliminated. In this work, we will settle on the semantics proposed in Shen and Eiter (2016). The flurry of new research also led to the development of ELP solving systems (Kahl *et al.* 2015; Son *et al.* 2017). Such solvers employ readily available, highly efficient ASP systems like *clingo* (Gebser *et al.* 2012; 2014) and *WASP* (Alviano *et al.* 2013), especially making use of the former solver's multi-shot solving functionality (Gebser *et al.* 2019). However, these ELP solving systems rely on ground ASP programs when calling the ASP solver, which, for complexity-theoretic reasons, generally requires multiple calls in order to check for world view existence. The main aim of our paper is to present techniques and a system for solving ELPs that is able to utilize an ASP solver in such a way that the ELP can be solved in a single shot.

Contributions. Our contributions in this paper are twofold:

- We propose a novel translation from ELPs to ASP programs using large non-ground ASP rules, such that the ELP can be solved by an ASP solving system in a single shot. This is done via a recently proposed encoding technique (Bichler *et al.* 2016b) that uses large ASP rules to formulate complex checks. This technique builds on a result from (Eiter *et al.* 2007) that states that evaluating non-ground ASP programs with bounded predicate arity is Σ_P^3-complete, which matches the complexity of evaluating ELPs. Our proposed translation is therefore optimal from a complexity-theoretic point of view. From a practical point of view, such an encoding avoids multiple calls to the ASP solver. State-of-the-art systems use sophisticated heuristics and learning, and multiple calls might result in a loss of knowledge about the problem instance, which the solver has already learned.
- We further discuss how our encoding needs to be constructed in order to be useful in practice. In particular, in current ASP systems, non-ground ASP programs first need to be *grounded*, that is, all variables need to be replaced by all allowed combinations of constants. Since our encoding makes use of large non-ground rules, a naive grounding will often not terminate, since there may be hundreds or thousands of variables in a rule. However, as proposed in Bichler *et al.* (2016b), we make use of the *lpopt* rule decomposition tool (Bichler *et al.* 2016a) that splits such large rules into smaller ones that are more easily grounded, by making use of the concept of *treewidth* and *tree decompositions* (Bodlaender 1993). To use this tool to its full potential, the large rules we use in our encoding must be constructed carefully, in order for *lpopt* to split them up optimally.

- Finally, we present a prototype implementation of our ELP-to-ASP rewriting approach and combine it with the state-of-the-art ASP solving system *clingo* (Gebser *et al.* 2014) in order to evaluate its performance. We compare our system against *EP-ASP* (Son *et al.* 2017) on different benchmarks found in the literature. Our system shows competitive performance on these benchmarks, in particular on instances with good structural properties.

The remainder of the paper is structured as follows: in Section 2, we introduce the formal background of ASP, ELPs, and tree decompositions; Section 3 states our reduction from ELPs to ASP, including practical considerations and a discussion of related work; Section 4 presents how quantified boolean formula (QBF) formulas can be encoded as ELP programs; Section 5 introduces our ELP solver; Section 6 presents our benchmark results, making use of results from Section 4; and finally Section 7 closes with some concluding remarks.

This paper is extended versions of Bichler *et al.* (2018a, 2018b). Additional material includes a full correctness proof for our reduction in Section 3 and a formalized and detailed description of the adaptations needed to make the reduction workable in practice. Furthermore, Section 4 describes in detail how our QBF benchmarks, used in Section 6, are constructed.

2 Preliminaries

Answer set programming (ASP). A *ground logic program* (also called answer set program, ASP program, or simply program) is a pair $\Pi = (\mathscr{A}, \mathscr{R})$, where $\mathscr{A}$ is a set of propositional (i.e., ground) atoms and $\mathscr{R}$ is a set of rules of the form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n; \tag{1}$$

where the comma symbol stands for conjunction, $n \geq m \geq l \geq 0$ and $a_i \in \mathscr{A}$ for all $1 \leq i \leq n$. Each rule $r \in \mathscr{R}$ of form (1) consists of a head $H(r) = \{a_1, \ldots, a_l\}$ and a body given by $B^+(r) = \{a_{l+1}, \ldots, a_m\}$ and $B^-(r) = \{a_{m+1}, \ldots, a_n\}$. A *literal* ℓ is either an atom a or its (default) negation $\neg a$. A literal ℓ is true in a set of atoms $I \subseteq \mathscr{A}$ if $\ell = a$ and $a \in I$, or $\ell = \neg a$ and $a \notin I$; otherwise ℓ is false in I. A set $M \subseteq \mathscr{A}$ is a called a *model* of r if $B^+(r) \subseteq M$ together with $B^-(r) \cap M = \emptyset$ implies that $H(r) \cap M \neq \emptyset$. We denote the set of models of r by $models(r)$, and the models of a logic program $\Pi = (\mathscr{A}, \mathscr{R})$ are given by $models(\Pi) = \bigcap_{r \in \mathscr{R}} models(r)$. The GL-reduct Π^I of a ground logic program Π with respect to a set of atoms $I \subseteq \mathscr{A}$ is the program $\Pi^I = (\mathscr{A}, \{H(r) \leftarrow B^+(r) \mid r \in \mathscr{R}, B^-(r) \cap I = \emptyset\})$.

Definition 2
Gelfond and Lifschitz (1988, 1991) $M \subseteq \mathscr{A}$ is an *answer set* of a program Π if (1) $M \in models(\Pi)$ and (2) there is no subset $N \subset M$ such that $N \in models(\Pi^M)$.

The set of answer sets of a program Π is denoted $AS(\Pi)$. The *consistency problem* of ASP (decide whether, given Π, $AS(\Pi) \neq \emptyset$) is Σ_P^2-complete (Eiter and Gottlob 1995).

General *non-ground logic programs* differ from ground logic programs in that variables may occur in rules. Such rules are $\forall$-quantified first-order implications of the form $H_1 \vee \cdots \vee H_k \leftarrow P_1, \ldots, P_n, \neg N_1, \ldots, \neg N_m$, where H_i, P_i, and N_i are (non-ground) atoms. A *non-ground atom* A is of the form $p(\mathbf{t})$ and consists of a predicate name p, as well as a sequence of *terms* $\mathbf{t}$, where each term $t \in \mathbf{t}$ is either a *variable* or a *constant* from a domain

Δ, with $|\mathbf{t}|$ being the arity of p. Let $var(A)$ denote the set of variables $\mathbf{X}$ in a non-ground atom A. This notation naturally extends to sets. We will denote variables by capital letters, constants, and predicates by lower-case words. A non-ground rule can be seen as an abbreviation for all possible instantiations of the variables with domain elements from Δ. This step is usually explicitly performed by a *grounder* that transforms a (non-ground) logic program into a set of ground rules of the form (1). Note that, in general, such a ground program may be exponential in the size of the non-ground program. For non-ground programs of bounded arity, the consistency problem is Σ_P^3-complete (Eiter *et al.* 2007).

Epistemic logic programs. A *ground ELP* is a pair $\Pi = (\mathscr{A}, \mathscr{R})$, where $\mathscr{A}$ is a set or 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 each a_i is an atom, each ℓ_i is a *literal*, and each ξ_i is an *epistemic literal*, that is, a formula $\mathbf{not}\,\ell$, where $\mathbf{not}$ is the epistemic negation operator, and ℓ is a literal. W.l.o.g. we assume that no atom appears twice in a rule[1]. Let $elit(r)$ denote the set of all epistemic literals occurring in a rule $r \in \mathscr{R}$. This notation naturally extends to programs. Let $H(r) = \{a_1, \ldots, a_k\}$. 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.

In order to define the main reasoning tasks for ELPs, we recall the notion of the epistemic reduct (Shen and Eiter 2016). Let $\Phi \subseteq elit(\Pi)$ (called a *guess*). The *epistemic reduct* Π^Φ of the program $\Pi = (\mathscr{A}, \mathscr{R})$ w.r.t. Φ consists of the rules $\{r^\neg \mid r \in \mathscr{R}\}$, where $r^\neg$ is defined as the rule r with all epistemic literals $\mathbf{not}\,\ell$ in Φ (resp. in $elit(\Pi) \setminus \Phi$) replaced by $\top$ (respectively, $\neg\ell$). Note that Π^Φ is a logic program without epistemic negation[2]. This leads to the following central definition.

Definition 3
Let Φ be a guess. The set $\mathscr{M} = AS(\Pi^\Phi)$ is called a *candidate world view* of Π iff

1. $\mathscr{M} \neq \emptyset$,
2. for each epistemic literal $\mathbf{not}\,\ell \in \Phi$, there exists an answer set $M \in \mathscr{M}$ wherein ℓ is false, and
3. for each epistemic literal $\mathbf{not}\,\ell \in elit(\Pi) \setminus \Phi$, it holds that ℓ is true in each answer set $M \in \mathscr{M}$.

Example 4
Let Π be the following ELP, with $\mathscr{R} = \{r_1, r_2\}$:

$$r_1 : p \leftarrow \mathbf{not}\,q$$

$$r_2 : q \leftarrow \mathbf{not}\,p$$

ELP Π has two candidate world views: (1) $\Phi = \{\mathbf{not}\,q\}$ with $AS(\Pi^\Phi) = \{\{p\}\}$; (2) $\Phi = \{\mathbf{not}\,p\}$ with $AS(\Pi^\Phi) = \{\{q\}\}$.

---

[1] This can be achieved by introducing auxiliary atoms whenever an atom appears twice in a rule, and add two rules that ensure that the original and auxiliary atom must be equivalent.
[2] We interpret double negation according to Faber *et al.* (2011).

The main reasoning task we treat in this paper is the *world view existence problem* (or *ELP consistency*), that is, given an ELP Π, decide whether a candidate world view exists. This problem is known to be Σ_3^P-complete (Shen and Eiter 2016).

Tree Decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathscr{T} = (T, \chi)$, where T is a rooted tree and χ is a labelling function over nodes t, with $\chi(t) \subseteq V$, such that the following holds: (i) for each $v \in V$ there is a node t in T such that $v \in \chi(t)$; (ii) for each $\{v, w\} \in E$ there is a node t in T such that $\{v, w\} \subseteq \chi(t)$; and (iii) for all nodes r, s, and t in T, where s lies on the path from r to t, $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The *width* of a tree decomposition $\mathscr{T}$ is defined as the maximum cardinality of $\chi(t)$ minus one, over all nodes t of $\mathscr{T}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. Trees have treewidth 1, and cliques of size k have treewidth k. Finding a tree decomposition of minimal width is non-deterministic polynomial time (NP)-hard in general.

3 Single-Shot ELP solving

In this section, we provide our novel translation for solving ELPs via a single call to an ASP solving system. The goal is to transform a given ELP Π to a non-ground ASP program Π' with predicates of bounded arity, such that Π is consistent (i.e., it has a candidate world view) iff Π' has at least one answer set. A standard ASP solver can then decide the consistency problem for the ELP Π in a single call, by solving Π'.

3.1 Reducing ELPs to ASP programs

The reduction is based on an encoding technique proposed in Bichler *et al.* (2016b), which uses large, non-ground rules. Given an ELP Π, the ASP program Π' will roughly be constructed as follows. Π' contains a guess part that chooses a set of epistemic literals from $elit(\Pi)$, representing a guess Φ for Π. Then, the check part verifies that, for Φ, a candidate world exists. In all, the ASP program Π' consists of five parts:

$$\Pi' = \Pi'_{facts} \cup \Pi'_{guess} \cup \Pi'_{check_1} \cup \Pi'_{check_2} \cup \Pi'_{check_3},$$

where the subprogram Π'_{facts} is a set of facts representing the ELP Π, and Π'_{check_i} represents the part of the program that checks Condition i of Definition 3. We now proceed to the construction of the program Π'. Let $\Pi = (\mathscr{A}, \mathscr{R})$ be the ELP to reduce from. To ease notation, let $\mathscr{A} = \{a_1, \ldots, a_n\}$.

The set of facts Π'_{facts}. Π'_{facts} represents basic knowledge about the ELP Π, plus some auxiliary facts and is given as:

- $atom(a)$, for each atom $a \in \mathscr{A}$;
- $elit(\ell)$, for each epistemic literal **not** $\ell \in elit(\Pi)$[3];
- $leq(0,0)$, $leq(0,1)$, and $leq(1,1)$, representing the less or equal relation for Boolean values; and
- $or(0,0,0)$, $or(0,1,1)$, $or(1,0,1)$, and $or(1,1,1)$, representing the Boolean relation *or*.

---

[3] Note that we use the literal ℓ as an ASP constant.

Subprogram Π'_{guess}. This part of the program consists of a single, non-ground rule that guesses a subset of the epistemic literals (stored in predicate g) as follows:

$$g(L, 1) \vee g(L, 0) \leftarrow elit(L).$$

Shorthands. Before defining the three check parts of the program, we will introduce some useful shorthands which will be used at several occasions. To this end, we use a context identifier $\mathscr{C}$. We first define the following:

$$H_{val}^{\mathscr{C}}(A) \equiv v_{\mathscr{C}}(A, 1) \vee v_{\mathscr{C}}(A, 0),$$

that is, $H_{val}^{\mathscr{C}}(A)$ guesses a truth assignment for some variable A and stores it in relation $v_{\mathscr{C}}$. We will often use variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ or $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$ to represent a subset M of $\mathscr{A}$, where assigning X_i to 1 characterizes $a_i \in M$, and $X_i = 0$ otherwise. Let

$$B_{val}^{\mathscr{C}}(\mathbf{X}) \equiv \bigwedge_{a_i \in \mathscr{A}} v_{\mathscr{C}}(a_i, X_i),$$

that is, $B_{val}^{\mathscr{C}}(\mathbf{X})$ extracts the truth assignment from relation $v_{\mathscr{C}}$ into the variables $\mathbf{X}$ as described above. Finally, for some rule r in Π, we define a formula $B_{sat}^r(\mathbf{X}, \mathbf{Y}, S)$ that checks whether the rule r is satisfied in the epistemic reduct Π^{Φ} w.r.t. the guess Φ encoded in the relation g, when the negative body (respectively, positive body and head) is evaluated over the set of atoms encoded by $\mathbf{X}$ (respectively, $\mathbf{Y}$). If the rule is satisfied, $B_{sat}^r(\mathbf{X}, \mathbf{Y}, 1)$ should hold, and $B_{sat}^r(\mathbf{X}, \mathbf{Y}, 0)$ otherwise. This is done as follows. Let r contain the atoms $\{a_{i_1}, \ldots, a_{i_m}\}$ (recall that no atom appears twice in a rule), where $i_1, \ldots, i_m \in \{1, \ldots, n\}$. For ease of notation, we will use a four-ary *or* relation, which can easily be split into two of our three-ary *or* atoms using a helper variable T:

$$or(W, X, Y, Z) \leftarrow or(W, X, T), or(T, Y, Z).$$

The following is the central building block of our reduction:

$$B_{sat}^r(\mathbf{X}, \mathbf{Y}, R_m) \equiv R_0 = 0, \bigwedge_{a_{i_j} \in H(r)} or(R_{j-1}, Y_{i_j}, R_j),$$

$$\bigwedge_{a_{i_j} \in B(r)} or(R_{j-1}, 1 - Y_{i_j}, R_j), \bigwedge_{\neg a_{i_j} \in B(r)} or(R_{j-1}, X_{i_j}, R_j),$$

$$\bigwedge_{\mathbf{not}\, a_{i_j} \in B(r)} g(a_{i_j}, N_j), or(N_j, 1 - X_{i_j}, T_j), or(R_{j-1}, 1 - T_j, R_j),$$

$$\bigwedge_{\mathbf{not}\, \neg a_{i_j} \in B(r)} g(\neg a_{i_j}, N_j), or(N_j, Y_{i_j}, T_j), or(R_{j-1}, 1 - T_j, R_j),$$

$$\bigwedge_{\neg \mathbf{not}\, a_{i_j} \in B(r)} g(a_{i_j}, N_j), or(R_{j-1}, N_j, 1 - Y_{i_j}, R_j),$$

$$\bigwedge_{\neg \mathbf{not}\, \neg a_{i_j} \in B(r)} g(\neg a_{i_j}, N_j), or(R_{j-1}, N_j, X_{i_j}, R_j).$$

For a rule r, each big conjunction in the above formula encodes a reason for r to be satisfied. For example, the fifth line encodes the fact that rule r is true if the disjunct $\neg\mathbf{not}\,a_{i_j}$ is not satisfied, that is, if the epistemic literal $\mathbf{not}\,a_{i_j}$ is part of the guess Φ, or the atom a_{i_j} is false (represented by $1-Y_{i_j}$). Each disjunct of rule r is evaluated in this way, and the results are connected via the *or* relation (with the result of the first i disjuncts stored in variable R_i). Therefore, R_m will be 1 if r is satisfied, and 0 otherwise, as desired (recall that r has m disjuncts). The following example illustrates how this shorthand is constructed for a concrete input program.

Example 5

Recall program $\Pi = (\mathscr{A}, \mathscr{R})$ from Example 4. Let $\mathscr{A} = \{a_1, a_2\}$, where $a_1 = p$ and $a_2 = q$. Let rule $r_2 \in \mathscr{R}$ contain the atoms $\{a_{i_1}, a_{i_2}\}$, where $i_1 = 2$ and $i_2 = 1$. We give the core construct, $B_{sat}^r(\cdot, \cdot, \cdot)$ for rule r_2:

$$B_{sat}^{r_2}(X_1, X_2, Y_1, Y_2, R_2) \equiv or(0, Y_2, R_1), g(p, N_2), or(N_2, 1{-}X_1, T_2), or(R_1, 1{-}T_2, R_2).$$

Finally, we define $B_{ss}(\mathbf{X}, \mathbf{Y})$, which makes sure that the variables $\mathbf{Y}$ identify a strict subset of the atoms identified by $\mathbf{X}$. Let $B_{ss}(\mathbf{X}, \mathbf{Y}) \equiv$

$$N_0 = 0, N_n = 1, \bigwedge_{a_i \in \mathscr{A}} leq(Y_i, X_i), or(N_{i-1}, X_i{-}Y_i, N_i).$$

We can now proceed with the remainder of our reduction.

Subprogram Π'_{check_1}. This part of the program needs to check that, given the guess Φ made in Π'_{guess}, there exists at least one answer set of the epistemic reduct Π^Φ, as per Definition 3(1). Therefore, according to Definition 2, we need to find a set $M \subseteq \mathscr{A}$, such that (1) M is a model of Π^Φ, and (2) there is no proper subset of M that is a model of the GL-reduct $(\Pi^\Phi)^M$. Π'_{check_1} contains the following rules:

- $H_{val}^{check_1}(A) \leftarrow atom(A);$
- $\bot \leftarrow B_{val}^{check_1}(\mathbf{X}), B_{sat}^r(\mathbf{X}, \mathbf{X}, 0),$ for each $r \in \mathscr{R}$; and
- $\bot \leftarrow B_{red}^{check_1}.$

The first rule guesses a truth assignment for all atoms. The second rule verifies that there is no rule in Π^Φ that is violated by the candidate answer set M, represented by the variables $\mathbf{X}$, and guessed by the first rule. $B_{red}^{\mathscr{C}}$ checks whether a subset of M is a model of the GL-reduct $(\Pi^\Phi)^M$. To this end, let

$$B_{red}^{\mathscr{C}} \equiv B_{val}^{\mathscr{C}}(\mathbf{X}), B_{ss}(\mathbf{X}, \mathbf{Y}), \bigwedge_{r \in \mathscr{R}} B_{sat}^r(\mathbf{X}, \mathbf{Y}, 1).$$

The last big conjunction in $B_{red}^{\mathscr{C}}$ makes sure that the subset $N \subset M$ identified by the variables $\mathbf{Y}$ is indeed a model of every rule in the GL-reduct $(\Pi^\Phi)^M$. This completes Π'_{check_1}.

Subprogram Π'_{check_2}. This part needs to check that, for every epistemic literal $\mathbf{not}\,\ell \in \Phi$, the epistemic reduct Π^Φ has some answer set wherein ℓ is false. Π'_{check_2} contains the

following rules and facts, for each epistemic literal $\mathbf{not}\,\ell \in elit(\Pi)$ (used as the context $\mathscr{C}$ so guesses are independent):

- $H_{val}^{\ell}(A) \leftarrow atom(A), g(\ell, 1);$
- $v_{\ell}(a, \eta)$, where $\eta = 1$ if $\ell = \neg a$, or $\eta = 0$ if $\ell = a$;
- $\bot \leftarrow B_{val}^{\ell}(\mathbf{X}), B_{sat}^{r}(\mathbf{X}, \mathbf{X}, 0)$, for each $r \in \mathscr{R}$; and
- $\bot \leftarrow B_{red}^{\ell}.$

These rules guess, for each epistemic literal $\mathbf{not}\,\ell \in \Phi$, a candidate answer set M wherein ℓ is false and then verify that M is indeed an answer set, using the same technique as in Π'_{check_1}. This ensures Condition 2 of Definition 3.

Subprogram Π'_{check_3}. Finally, this part needs to check that, for every epistemic literal $\mathbf{not}\,\ell \in elit(\Pi) \setminus \Phi$, every answer set of Π^{Φ} satisfies ℓ. The construction makes use of the technique of saturation (Eiter and Gottlob 1995):

- $H_{val}^{check_3}(A) \leftarrow atom(A);$
- $v_{check_3}(A, 0) \leftarrow sat, atom(A);$
- $v_{check_3}(A, 1) \leftarrow sat, atom(A);$ and
- $\bot \leftarrow \neg sat.$

This setup checks that, for every candidate answer set M guessed in the first rule, the atom sat is derived. Since we are only interested in answer sets, we first check that M is indeed one, using the following rules, similarly to Π'_{check_1}:

- $sat \leftarrow B_{val}^{check_3}(\mathbf{X}), B_{sat}^{r}(\mathbf{X}, \mathbf{X}, 0)$, for each $r \in \mathscr{R}$; and
- $sat \leftarrow B_{red}^{check_3}.$

It now remains to check that in each answer set M (that is, where sat has not been derived yet) all epistemic literals $\mathbf{not}\,\ell$ are either in Φ, or otherwise ℓ is true in M. This is done by adding the following rule to Π'_{check_3}:

$$sat \leftarrow \bigwedge_{\mathbf{not}\,a \in elit(\Pi)} g(a, N_a), v_{check_3}(a, X_a), or(N_a, X_a, 1),$$
$$\bigwedge_{\mathbf{not}\,\neg a \in elit(\Pi)} g(\neg a, N_a^{\neg}), v_{check_3}(a, X_a^{\neg}), or(N_a^{\neg}, 1 - X_a^{\neg}, 1).$$

This completes the reduction. We will now show that this reduction indeed accomplishes our goals. The correctness of our reduction can be intuitively seen from the observation that each of the three check parts of the constructed ASP program Π' ensures precisely one of the three conditions that define a candidate world view. Each answer set A of Π' is a witness for the fact that a guess $\Phi \subseteq elit(\Pi)$ encoded in A indeed gives rise to a candidate world view. The next theorem formally states that our reduction is correct.

Theorem 6
Let $\Pi = (\mathscr{A}, \mathscr{R})$ be an ELP and let Π' be the ASP program obtained from Π via the above reduction. Then, Π has a candidate world view if and only if Π' has an answer set.

Proof

We will begin with the "if" direction. To this end, assume that there is a guess $\Phi \subseteq elit(\Pi)$ for Π that gives rise to a candidate world view $\mathscr{M} = AS(\Pi^\Phi)$. We will show that Π' has an answer set M. Clearly, M contains all the facts from Π'_{facts}. Furthermore, let M contain the fact $g(\ell, 1)$ for each epistemic literal $\mathbf{not}\ \ell \in \Phi$ and the fact $g(\ell, 0)$ for each epistemic literal $\mathbf{not}\ \ell \in elit(\Pi) \setminus \Phi$. This clearly satisfies subprogram Π'_{guess}.

Now, let $M' \in \mathscr{M}$ be any answer set of Π^Φ (such an answer set exists, since, by assumption, $\mathscr{M}$ is a candidate world view for Φ and by Definition 3, $\mathscr{M}$ is non-empty). Let M contain the fact $v_{check_1}(a, 1)$ for each $a \in M'$ and the fact $v_{check_1}(a, 0)$ for each $a \in \mathscr{A} \setminus M'$. This satisfies subprogram Π'_{check_1} as follows. Clearly, M satisfies the first line of the subprogram. Since the atoms with relation v_{check_1} encode precisely the answer set M' of Π^Φ, and since M' is a model of Π^Φ, also the second line of the subprogram is satisfied, which, by construction, checks that the assignment encoded in relation v_{check_1} satisfies all the rules of Π^Φ. Finally, the third line, by construction, checks that the same assignment is also minimal w.r.t. the GL-reduct $[\Pi^\Phi]^{M'}$. Since M' is an answer set of Π^Φ, also this line of the subprogram Π'_{check_1} is satisfied.

The argument for satisfaction of Π'_{check_2} is similar to the one for Π'_{check_1}. Since $\mathscr{M}$ is a candidate world view for guess Φ, it contains, for each epistemic literal $\mathbf{not}\ \ell \in \Phi$, an answer set $M_\ell \in \mathscr{M}$ such that ℓ is false in M_ℓ. Thus, the argument for subprogram Π'_{check_1} can be analogously applied for each $\mathbf{not}\ \ell \in \Phi$, taking the answer set M_ℓ instead of M'.

Finally, we need to verify that M also satisfies the rules in Π'_{check_3}. To this end, let M contain the facts M_{check_3} consisting of the fact sat, as well as the fact $v_{check_3}(a, b)$ for each $a \in \mathscr{A}$ and $b \in \{0, 1\}$. It is easy to verify that all the rules in Π'_{check_3} are classically satisfied. However, since the negative literal $\neg sat$ appears in line 4, in order to verify that M is indeed an answer set, we also need to look at minimality w.r.t. GL-reduct. Since line 4 is removed in the GL-reduct of Π'_{check_3}, it may be the case that some subset of $M_{check_3} \setminus \{sat\}$ may indeed satisfy the GL-reduct. However, we will show that every such subset requires sat to be true via lines 5, 6, or 7 of Π'_{check_3} (i.e., those rules of Π'_{check_3} with atom sat in the head) and can therefore not exist. Indeed, every subset of $M_{check_3} \setminus \{sat\}$ that does not encode an answer set of Π^Φ in relation v_{check_3} derives sat via lines 5 or 6, by construction (the argument to see this is analogous to the one for the previous two subprograms). It remains to show that all other remaining subsets (i.e., the subsets of $M_{check_3} \setminus \{sat\}$ that encode answer sets of Π^Φ) also derive sat. However, since every answer set $M' \in \mathscr{M}$, by Definition 3, has to satisfy precisely the condition encoded by line 7 of Π'_{check_3}, this is easy to see. We thus have that M, as constructed above, is indeed an answer set of Π'.

The "only if" direction can be seen via similar arguments to the above. By construction, any answer set M of Π' will encode a guess Φ for Π. Since any such answer set M, to be an answer set, must satisfy the three check subprograms of Π' in the way described above, and these three check subprograms, by construction, correspond directly to the three conditions of Definition 3, we have that M encodes a guess Φ for Π that leads to a candidate world view. $\square$

As we have seen, our reduction works as intended: the ASP program Π' derived from the input ELP Π has an answer set precisely when Π has a candidate world view. The next interesting observation is that our reduction is, in fact, a polynomial-time reduction, as stated below.

Theorem 7
Given an ELP Π, the reduction above runs in time $O(e \cdot n)$, where n is the size of Π and $e = |elit(\Pi)|$, and uses predicates of arity at most three.

Proof
Predicates of arity at most three are used if the four-ary *or* relation is not materialized as an actual relation in ASP but viewed as a shorthand for two connected ternary *or* relations (cf. the paragraph on shorthands of our reduction). The reduction's runtime (and output size) can be seen to be in $O(e \cdot n)$ by noting the fact that the construct B_{red} is of size linear in n (it precisely encodes each rule using the *or* predicates). B_{red} is then used once for each epistemic literal in Π (cf. Π'_{check_2}). □

Note that the above theorem shows that our reduction is indeed worst-case optimal as claimed in Section 1: checking consistency of non-ground, fixed-arity ASP programs is Σ_P^3-complete, as is checking world view existence for ELPs.

3.2 Using the reduction in practice

As we have seen, using the construction in the previous subsection, we can solve the consistency problem for a given ELP via a single call to an ASP solving system. However, when trying this in practice, the performance is less than optimal, mainly for the following reason. At several points in the construction, large non-ground rules are used (i.e., where $B_{red}^{\mathscr{E}}$ appears in a rule body). As noted in Section 2, these rules need to be grounded but may contain hundreds or thousands of variables, which need to be replaced by all possible combinations of constants; a hopeless task for ASP grounders, as the resulting ground program, is exponential in the number of variables.

However, as noted in Bichler *et al.* (2016b), such large rules can often be decomposed into smaller, more manageable rules, using the *lpopt* tool (Bichler *et al.* 2016a). This tool roughly works as follows: (1) compute a *rule graph* G_r for each non-ground rule r, where there is a vertex for each variable V in r, and there is an edge between V_1 and V_2, if the two variables appear together in an atom of r; then (2) compute a tree decomposition of G_r of minimal width; and finally, (3) in a bottom-up manner, output a rule for each node in the tree decomposition. The resulting rules each contain only as many variables as the treewidth of G_r (plus one), and, together, are equivalent to the original rule r. After this rule decomposition step, grounding now becomes much easier, since the number of variables in each rule is reduced. Note that, since finding optimal tree decompositions is NP-hard, *lpopt* employs heuristics to find good decompositions.

In our construction, $B_{red}^{\mathscr{E}}$ stands for a long rule body that effectively encodes the entire input ELP Π. Each atom a_i in Π is represented by the two variables X_i and Y_i. If we represent Π as a graph G_Π, where each atom a_i is a vertex, and there is an edge between two atoms if they appear together in a rule in Π, then this graph structure can be found (as a minor) in the rule graph of $B_{red}^{\mathscr{E}}$. However, in addition, $B_{red}^{\mathscr{E}}$ also adds a series of
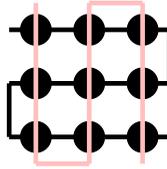
Fig. 1. Creating a grid from chains.

$or(\cdot, \cdot, \cdot)$ atoms (via $B_{ss}(\mathbf{X}, \mathbf{Y})$) that introduce additional connections in the rule graph of $B_{red}^{\mathscr{C}}$. These connections may increase the treewidth substantially. In fact, even if G_{Π} has a treewidth of 1, by introducing the additional connections in a bad way, the treewidth may increase arbitrarily: imagine that G_{Π} is a chain, depicted in black in Figure 1, and imagine the $or(\cdot, \cdot, \cdot)$-chain from $B_{ss}(\mathbf{X}, \mathbf{Y})$ is inserted into G_{Π}, illustrated in pink. The treewidth now depends on the chain's length (and thereby on the size of Π), and *lpopt* can no longer split the rule well.

   In the following, we will formalize the problem described above and present an extension to our reduction presented in the previous subsection that will alleviate the problem. First, we define the primal graph of an ELP Π, a standard notion in topics of satisfiability, constraint programming, and logic programming; cf. standard textbooks, for example, (Ebbinghaus and Flum 1995).

Definition 8
The *primal graph* of an ELP $\Pi = (\mathscr{A}, \mathscr{R})$ is the graph $G_{\Pi} = (V, E)$, where $V = \mathscr{A}$, and there is an edge $(a_i, a_j) \in E$ iff the atoms a_i and a_j occur together in a rule in $\mathscr{R}$.

   Then, we define the rule graph for a non-ground ASP rule r.

Definition 9
The *rule graph* of a non-ground ASP rule r is the graph $G_r = (V, E)$, such that $V = var(r)$, and there is an edge between two variables X and Y in E iff X and Y occur together in an atom in r.

   From the construction, it is not difficult to see that any rule r containing $B_{red}^{\mathscr{C}}$ reflects the structure of the input ELP Π, or, more formally, the graph G_{Π} is contained (as a minor) in the graph G_r. Thus, by well-known graph-theoretic results, the treewidth of G_r is at least the treewidth of G_{Π}. Since this is an integral part of our construction, we cannot hope for *lpopt* to split up rule r any better than the structure of Π allows. However, as noted in the intuitive problem description above, G_r contains additional connections between variables. These are introduced by the subformula $B_{ss}(\cdot, \cdot)$ that effectively links all the variables in a rule r into a chain in G_r. In the worst case, as illustrated in Figure 1, these additional connections in G_r may increase the treewidth arbitrarily, making it almost impossible for *lpopt* to split up the rule well. It is therefore important to introduce these additional connections carefully. We will now introduce a more involved construction of $B_{ss}(\cdot, \cdot)$ that preserves the treewidth of G_{Π} in G_r (i.e., does not arbitrarily increase it). In this modified version, $B_{ss}(\cdot, \cdot)$ is constructed as follows:

   1. First, compute a tree decomposition $\mathscr{T}_{\Pi}$ of G_{Π} with minimal width.
   2. Secondly, construct $B_{ss}(\cdot, \cdot)$ in a bottom-up (i.e., post-order traversal) fashion along this tree decomposition in the following way, for each node type. To this end, let

$\mathscr{A} = \{a_1, \ldots, a_n\}$, and, for a node t of $\mathscr{T}_\Pi$, let $\chi(t)$ contain the set of atoms $\{a_{i_1}, \ldots, a_{i_m}\}$, with $i_j \in \{1, \ldots, n\}$.

Leaf node t: For a leaf node t of $\mathscr{T}_\Pi$, let $B_{ss}(\mathbf{X}, \mathbf{Y})$ contain the following conjunction of atoms:

$$N_0^t = 0, N^t = N_m^t, \bigwedge_{a_{i_j} \in \chi(t)} leq(Y_{i_j}, X_{i_j}), or(N_{j-1}^t, X_{i_j} - Y_{i_j}, N_j^t),$$

that is, N^t contains 1, if the proper subset condition between $\mathbf{X}$ and $\mathbf{Y}$ is already fulfilled in node t, and 0 otherwise.

Inner node t: For an inner node t of $\mathscr{T}_\Pi$ with children $t_1, \ldots, t_k$, let $B_{ss}(\mathbf{X}, \mathbf{Y})$ contain the same conjunction as for a leaf node, but where the equality atom $N^t = N_m^t$ is replaced by the following disjunction:

$$or(N_m^t, N^{t_1}, \ldots, N^{t_k}, N^t),$$

where the $k + 2$-ary *or* atom can be split into 3-ary *or* atoms in the same way as with the 4-ary *or* atom in our main construction. Intuitively, we now have that N^t is set to 1 if the proper subset condition is already fulfilled somewhere in the subtree rooted at t.

Root node t_{root}: Finally, for the root node t_{root}, we add the same conjunction of atoms to $B_{ss}(\mathbf{X}, \mathbf{Y})$ as for an inner node, but, in addition, we add the final condition $N^{t_{root}} = 1$, that makes sure that, at the root node, the proper subset condition is fulfilled.

If constructed in the way described above, it is not difficult to see that $B_{ss}(\mathbf{X}, \mathbf{Y})$ still ensures the same condition as in our original construction from Section 3.1, namely, that the variables $\mathbf{Y}$ identify a proper subset of the atoms identified by the variables $\mathbf{X}$. However, the treewidth of a rule containing $B_{ss}(\cdot, \cdot)$ is now not increased arbitrarily. In fact, it can be verified that the treewidth of G_Π is preserved up to a constant additive factor, for any rule containing $B_{red}^{\mathscr{C}}$, when using the alternative construction for $B_{ss}(\cdot, \cdot)$ provided above. In practice, this means that *lpopt* is able to split the rule up as well as possible, that is, as well as the structure of Π allows.

3.3 Discussion and related work

As we have seen, the reduction proposed above allows us to solve ELPs via a single call to an ASP solving system. However, our encoding also has several other interesting practical properties, which make it very flexible for use with, for example, different ASP semantics, or harder problems. A brief discussion follows.

Other ASP semantics. Apart from the original semantics for ASP (called *stable model semantics*, (Gelfond and Lifschitz 1988,1991), several different semantics have been proposed that investigate how to interpret more advanced constructs in ASP, like double negation, aggregates, optimization, and more (Lifschitz *et al.* 1999; Pearce 2006; Pelov *et al.* 2007; Ferraris *et al.* 2011; Faber *et al.* 2011; Shen *et al.* 2014). Epistemic reducts may contain double negation, and we have opted to use the FLP semantics of (Faber *et al.* 2011), as used in Shen and Eiter (2016), to interpret this. The actual interpretation

of double negation is encoded in the $B^r_{sat}(\cdot, \cdot)$ shorthand defined in our reduction. This construction is very flexible and can easily be modified to use different ASP semantics (e.g., (Lifschitz *et al.* 1999)).

Enumeration of world views. Modern ASP systems like *clasp* (Gebser *et al.* 2012) contain several useful features not included in the ASP base language. One such feature is an advanced implementation of projection, as presented in Gebser *et al.* (2009): given a set of atoms (or relations), the solver will output answer sets where all other atoms are projected away and will also guarantee that there are no repetitions (even if multiple answer sets with the same assignment on the projected atoms exist), while still maintaining efficiency. This can be used to enumerate candidate world views by projecting away all relations in our encoding, except for $g(\cdot)$ and $v_{check_1}(\cdot)$. When enumerating all projected answer sets in this way, our encoding yields all guesses together with their candidate world views (when grouped by $g(\cdot)$).

Comparison to related work. Classic ELP solvers generally work by first establishing a candidate epistemic guess Φ and then use an answer set solver to verify that the epistemic guess indeed yields an epistemic reduct whose answer sets form a candidate world view of the original ELP w.r.t. Φ. Different approaches are used to find promising epistemic guesses and also to verify that they lead to candidate world views, but, generally, these systems have in common that an underlying ASP solver is used, and called multiple times, to solve the ELP. Notable recent ELP solvers that follow this approach include *EP-ASP* (Son *et al.* 2017), *GISolver* (Zhang *et al.* 2015) and a later, probabilistic, variant called *PelpSolver*, and *ELPsolve* (Kahl *et al.* 2016). A comprehensive survey of recent ELP solving systems (including the one presented in the present paper) can be found in Leclerc and Kahl (2018).

We are not aware of another single-shot ELP solver that only needs to call an underlying ASP system once. However, the idea of our approach is similar to the one used in Bichler *et al.* (2016b), where a single-shot ASP encoding for disjunctive ASP, which is rewritten into non-ground normal ASP with fixed arity, is presented. That is, a solving system for normal ASP would be able to solve a disjunctive ASP program in a single call. However, this approach was not implemented and only presented as an example to show how long non-ground rules with fixed arity can be used to solve hard problems. In order to use such encodings (including our own presented herein), (Bichler *et al.* 2016b) make use of *rule decomposition*, where large non-ground ASP rules are split up into smaller parts based on tree decompositions (Morak and Woltran 2012). This rule decomposition approach was implemented as a stand-alone tool called *lpopt* (Bichler *et al.* 2016a) but has also recently been integrated into ASP solving systems like *I-DLV* (Calimeri *et al.* 2017).

4 Application: QBF solving

In this section, we illustrate the power of ELPs by illustrating a way to solve QBF formulas with at most three quantifier alternations (3-QBF) by encoding them as ELPs. This provides an alternative way to show the Σ^3_P lower bound for ELP consistency but relies on the existence of a reduction from the so-called *restricted 3-QBF formulas*. Shen and Eiter (2016) present such a reduction from restricted 3-QBF formulas to ELP world

view existence. Our aim is to generalize this by presenting a reduction from (general) 3-QBF formulas to restricted 3-QBF formulas. We will use this result to benchmark our ELP solver presented in Section 5. Let us begin by first recalling the definition of such 3-QBF formulas.

Definition 10
A *3,∃-QBF in conjunctive normal form (CNF) form* (or *QBF*, for short) is a formula of the form

$$\exists \mathbf{X} \forall \mathbf{Y} \exists \mathbf{Z} \varphi$$

where $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are sets (or sequences) of distinct (propositional) atoms (also called variables), and $\varphi = \bigwedge_{i=1}^{k} C_i$ is a CNF over the atoms $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$, that is, $C_i = \bigvee_{j=0}^{k_i} L_{i,j}$ is a clause of size k_i and $L_{i,j}$ is either an atom a or its negation $\neg a$.

W.l.o.g. we can assume that the clause size $k_i = 3$ for each $0 < i \le k$, that is, that φ is given in 3-CNF form, where each clause has at most three elements. In Shen and Eiter (2016), the authors make use of a version of QBFs called *restricted QBFs*. These are QBFs that evaluate to true under all interpretations of the existentially quantified variables if all universally quantified variables are replaced by $\top$ (i.e., if they are set to true).

Definition 11
A *restricted QBF* is a QBF where $\varphi[y/\top \mid y \in \mathbf{Y}]$ is a tautology.

The hardness proof of Theorem 5 of Shen and Eiter (2016) is a reduction from the validity problem of restricted QBFs to the consistency problem of ELPs. While the actual construction of the reduction is not needed for our purposes in this section, we nevertheless report it here, for completeness sake.

Proposition 12
(Shen and Eiter 2016, Proof of Theorem 5) Let $\Theta = \exists \mathbf{X} \forall \mathbf{Y} \exists \mathbf{Z} \varphi$ be a restricted QBF. Then, there exists an ELP Π such that Π has a candidate world view iff Θ is satisfiable.

Proof
The ELP Π consists of the following rules:

- For each variable $X \in \mathbf{X}$:

$$X \leftarrow \mathbf{not}\, \overline{X},$$
$$\overline{X} \leftarrow \mathbf{not}\, X.$$

- For each variable $Y \in \mathbf{Y}$:

$$Y \leftarrow \neg \overline{Y},$$
$$\overline{Y} \leftarrow \neg Y.$$

- For each variable $Z \in \mathbf{Z}$:

$$Z \vee \overline{Z}.$$

- For each clause C_i, $0 < i \le k$:

$$U \leftarrow L_{i,1}^*, L_{i,2}^*, L_{i,3}^*,$$

where $*$ is an operator that converts a positive literal W into $\overline{W}$, and a negative literal $\neg W$ into W.

- For each $Z \in \mathbf{Z}$:

$$Z \leftarrow U,$$

$$\overline{Z} \leftarrow U.$$

- And, finally, the rule

$$V \leftarrow \mathbf{not}\, V, \mathbf{not}\, \neg U.$$

As reported in Shen and Eiter (2016), the above ELP has a (candidate) world view iff $\exists \mathbf{X} \forall \mathbf{Y} \exists \mathbf{Z}\, \varphi$ is satisfiable. $\qquad\square$

We now show a more general reduction that also works for the nonrestricted case. To this end, we will combine the (Shen and Eiter 2016) reduction with our own reduction of QBF formulas to restricted QBF formulas. To achieve our goal, we are going to introduce one new atom v_i in each clause C_i and $\forall$-quantify these new atoms together with the $\mathbf{Y}$ atoms.

Definition 13
Given a QBF $\Theta = \exists \mathbf{X} \forall \mathbf{Y} \exists \mathbf{Z}\varphi$ with φ being constructed as in Definition 10, let its *extension*, denoted $\Theta^\uparrow$, be the QBF

$$\Theta^\uparrow = \exists \mathbf{X} \forall (\mathbf{Y} \cup \mathbf{V}) \exists \mathbf{Z}\varphi',$$

where

$$\varphi' = \bigwedge_{i=1}^{k} \left(v_i \vee \bigvee_{j=0}^{k_i} L_{i,j} \right),$$

and $\mathbf{V} = \{v_1, \dots, v_k\}$ is a list of fresh atoms.

It is easy to see that any extension $\Theta^\uparrow$ of a QBF Θ is a restricted QBF.

Proposition 14
Let Θ be a QBF. Its extension $\Theta^\uparrow$ is a restricted QBF.

We will now show that validity-equivalence between a QBF and its extension is preserved. For the proof, we establish the following terminology: given a subset of atoms $\sigma \subseteq \mathbf{S}$, we define its *out-set* $\overline{\sigma} = \{\neg a \mid a \in \mathbf{S} \setminus \sigma\}$ and its *literal-set* $\widehat{\sigma} = \sigma \cup \overline{\sigma}$.

Proposition 15
Θ and $\Theta^\uparrow$ are validity-equivalent.

Proof
$(\Rightarrow)$ Assume Θ is valid, that is, there exists an interpretation $\sigma_X \subseteq \mathbf{X}$, such that for any interpretation $\sigma_Y \subseteq \mathbf{Y}$, there exists an interpretation $\sigma_Z \subseteq \mathbf{Z}$ such that $(\widehat{\sigma_X} \cup \widehat{\sigma_Y} \cup \widehat{\sigma_Z}) \cap C_i \neq \emptyset$ for all $i \in \{1, \dots, k\}$. By monotonicity of non-emptiness of set intersections, also $(\widehat{\sigma_X} \cup \widehat{\sigma_Y} \cup \widehat{\sigma_V} \cup \widehat{\sigma_Z}) \cap (C_i \cup \{v_i\}) \neq \emptyset$ for any interpretation $\sigma_V \subseteq \mathbf{V}$ of a list of new atoms $\mathbf{V} = \{v_1, \dots, v_k\}$. But this is proof of the validity of $\Theta^\uparrow$.

$(\Leftarrow)$ For the other direction, assume $\Theta^\uparrow$ is valid, that is, there exists an interpretation $\sigma_X \subseteq \mathbf{X}$ such that for any interpretations $\sigma_Y \subseteq \mathbf{Y}$ and $\sigma_V \subseteq \mathbf{V}$, there exists an inter-

pretation $\sigma_Z \subseteq \mathbf{Z}$ such that $(\widehat{\sigma_X} \cup \widehat{\sigma_Y} \cup \widehat{\sigma_V} \cup \widehat{\sigma_Z}) \cap (C_i \cup \{v_i\}) \neq \emptyset$ for all $i \in \{1, \ldots, k\}$.
By setting $\sigma_V = \emptyset$, we especially get that there exists an interpretation $\sigma_X \subseteq \mathbf{X}$ such that for any interpretation $\sigma_Y \subseteq \mathbf{Y}$ there exists an interpretation $\sigma_Z \subseteq \mathbf{Z}$ such that $(\widehat{\sigma_X} \cup \widehat{\sigma_Y} \cup \{\neg v_1, \ldots, \neg v_k\} \cup \widehat{\sigma_Z}) \cap (C_i \cup \{v_i\}) \neq \emptyset$ for all $i \in \{1, \ldots, k\}$. Since the only literals containing a v_i variable on the left-hand side of the $\cap$ are negative and the only ones on the right-hand side are positive, we get $(\widehat{\sigma_X} \cup \widehat{\sigma_Y} \cup \widehat{\sigma_Z}) \cap C_i \neq \emptyset$ for all $i \in \{1, \ldots, k\}$, which establishes validity of Θ. $\qquad\square$

Now it is straightforward to generalize the reduction from Shen and Eiter (2016): let Θ be a QBF and apply the reduction from Shen and Eiter (2016) to the restricted QBF $\Theta^\uparrow$.

Theorem 16
Let Θ be a QBF. Let the ELP Π_Θ be obtained by applying the reduction from Shen and Eiter (2016) to the restricted QBF $\Theta^\uparrow$. It holds that Θ is valid iff Π_Θ is consistent, that is, Π_Θ has at least one candidate world view.

Correctness of this theorem follows from immediately from Propositions 12 and 15.

5 The *selp* system

We implemented the reduction in Section 3 as part of the single-shot ELP solving toolbox *selp*, available at `https://dbai.tuwien.ac.at/proj/selp`. In addition, the toolbox features a grounder for ELPs and a grouping script which groups answer sets of the reduction into candidate world views (allowing for enumeration). The tools are implemented in *python* and depend on the parser generator $LARK$[4], the rule decomposition tool *lpopt* (Bichler *et al.* 2016a), the tree decomposition tool *htd_main* (Abseher *et al.* 2017), and the ASP grounder *gringo* (Gebser *et al.* 2011).

Input formats. The *selp* solver reads the *EASP-not* file format, which is a restriction of the ASP input language of *gringo* to plain ground logic programs as defined in Section 2, extended with the `$not$` operator for epistemic negation. This allows us to encode ELPs as defined in Section 2. *selp* also supports *EASP-KM*, defined by adding the operators `K$` and `M$` instead of `$not$`. By allowing variables in body elements, both formats also have a *non-ground* version. The toolbox offers scripts to translate between the two formats.

Toolbox. We briefly present the main building blocks of *selp*.

easpGrounder.py takes as input a *non-ground EASP-not* program and outputs its equivalent ground form by rewriting it into an ASP program that the *gringo* grounder can understand and ground. This is done by encoding epistemic negation as predicate names and, after grounding, re-introducing epistemic negation where a placeholder predicate appears. Our grounding component, *easpGrounder.py*, supports arithmetics and the *sorts* format (Kahl *et al.* 2015) as input.

easp2asp.py is *selp*'s key component. It takes a ground *EASP-not* program, performs the reduction given in Section 3.1 (with some modifications to account for the extended language of ASP used by today's ASP systems and some straightforward optimizations),

---

[4] Available here: `https://github.com/erezsh/lark`

also adhering to the practical considerations presented in Section 3.2, and it finally outputs the resulting non-ground logic program in the syntax of *gringo*. Optionally, additional *clasp* directives are generated to allow for enumeration (cf. Section 3.3). For concrete implementation details, please consult the freely available source code at https://dbai.tuwien.ac.at/proj/selp.

groupWorldViews.py takes *clasp*'s output in JSON format, groups the answer sets into candidate world views according to their $g(\cdot)$ atoms, and outputs them in a human-readable format.

Usage. As a typical use case, suppose the file `problem.easp` contains a non-ground ELP encoding of a problem of interest and the file `instance.easp` contains a problem instance. In order to output all candidate world views, one would use the following command: flags `-pas` and `--project` enable projection of answer sets onto relevant predicates only. `-n0` tells *clasp* to compute all answer sets, and `--outf=2` to print in JSON format. *lpopt* is used to decompose long rule bodies. The `--sat-prepro=3` flag is recommended by *lpopt*):

```
cat problem.easp instance.easp |
easpGrounder.py -sELP | easp2asp.py -pas |
lpopt | gringo | clasp -n0 --outf=2 --project --sat-prepro=3 |
groupWorldViews.py
```

6 Experimental evaluation

We tested our system *selp* against the state-of-the-art ELP solver, *EP-ASP* (Son *et al.* 2017), using three test sets. For every test set, we measured the time it took to solve the consistency problem. For *selp*, *clasp* was stopped after finding the first answer set. For *EP-ASP*, search was terminated after finding the first candidate world view[5]. Note that a single answer set of the *selp* system is enough to establish consistency of an input ELP. *EP-ASP* needs to compute a full candidate world view to be able to prove consistency.

Experiments were run on a 2.1GHz AMD Opteron 6272 system with 224 GB of memory. Each process was assigned a maximum of 14 GB of RAM. For *EP-ASP*, we used the required *clingo* 4.5.3, since newer versions are incompatible with the solver. For *selp*, we used *clingo* 5.2.2, *htd_main* 1.2.0, and *lpopt* 2.2. The time it took *EP-ASP* to rewrite the input to its own internal format was not measured. *EP-ASP* was called with the preprocessing option for brave and cautious consequences on, since it always ran faster this way. The *selp* time is the sum of running times of its components.

Benchmark instances. We used three types of benchmarks, two coming from the ELP literature and one from the quantified boolean satisfiability domain that contains structures of low treewidth[6].

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

---

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

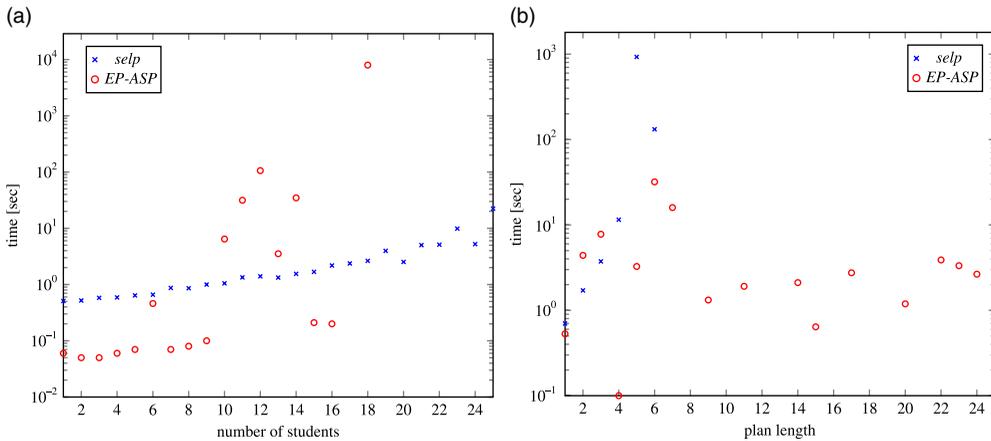[6] Benchmark archive: https://dbai.tuwien.ac.at/proj/selp

Fig. 2. Benchmark results. Missing points indicate timeouts.

Yale shooting (YS). This test set consists of 25 non-ground ELP programs encoding a simple version of the YS 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 Kahl *et al.* (2015).

Tree QBFs (TQ). The hardness proof for ELP consistency (Shen and Eiter 2016) 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 as discussed in Section 4. We apply this extended reduction to the 14 "Tree" instances of QBFEVAL'16 (Pulina 2016), available at `http://www.qbflib.org/family_detail.php?idFamily=56`, splitting each instance's variables into three random quantifier blocks.

Results. The results for the first two sets are shown in Figure 2. *selp* solves all instances from (SE) within 30 s, while *EP-ASP* only solves 17 within the time limit of 8 h. For YS, on the other hand, *selp* is able to solve only 6 instances within the time limit of 30 min, whereas *EP-ASP* can solve 17. Finally, for (TQ), *selp* can solve 6 of the 14 instances within the time limit of 12 h, whereas *EP-ASP* was unable to solve any instances at all.

These results confirm that *selp* is highly competitive on well-structured problems: in the SE instances, knowledge about students is not interrelated, and hence the graph G_Π of the ground ELP Π consists of one component for each student, thus having constant treewidth. The TQ instances keep their constant treewidth thanks to the fact that both the reductions from QBF to ELP (cf. Section 4) and from ELP to non-ground ASP (cf. Section 3.2) preserve the low treewidth of the original QBF instance. Different from *selp*, *EP-ASP* is not designed to exploit such structural information of ELPs and, consequently, performs worse than *selp* in these benchmarks. On the other hand, YS contains instances of high treewidth, even though it does not depend on the horizon. *EP-ASP* is therefore able to outperform *selp* on such instances. A similar observation can be made for the "Bomb in the Toilet" problem, as benchmarked in Son *et al.* (2017), which inherently

contains a huge clique structure. *selp* is not designed to solve such instances and is therefore most suited to solve ELPs of low treewidth, where it is able to efficiently exploit the problem structure.

7 Conclusions

In this paper, we have seen that ELPs can be encoded into ASP programs using long non-ground rules, such that a single call to an ASP solver is sufficient to evaluate them. A prototype ELP solver implementation, *selp*, performs particularly well on problems whose internal structure is of low treewidth. A combined solver that either calls *selp* or another state-of-the-art solver based on the treewidth of the input may therefore lead to even better overall performance.

Another topic for future work is that, under the FLP semantics, checking whether a given atom a is true in all candidate world views with a *subset-maximal* guess Φ is known to be Σ_P^4-complete (Shen and Eiter 2016). To solve this problem, advanced optimization features of state-of-the-art ASP solvers could allow us to encode this subset-maximality condition, while leaving the core of our encoding unchanged.

Finally, an interesting question is program optimization. Recently, a practical, easily applicable notion of strong equivalence for ELPs has been defined Faber *et al.* (2019). It would be interesting to investigate if and how parts of ELPs can be replaced in such a way that the solving performance of *selp* improves, seeing that *selp* is sensitive to treewidth. This could lead to an encoding technique for ELPs that tries to minimize the treewidth, similar to the class of connection-guarded ASP (Bliem *et al.* 2017), which was recently proposed in order to write ASP programs in such a way as to keep the treewidth of the resulting ground program low.

References

ABSEHER, M., MUSLIU, N. AND WOLTRAN, S. 2017. htd - A free, open-source framework for (customized) tree decompositions and beyond. In *Proc. CPAIOR,* 376–386.

ALVIANO, M., DODARO, C., FABER, W., LEONE, N. AND RICCA, F. 2013. WASP: A native ASP solver based on constraint learning. In *Proc. LPNMR*, 54–66.

BICHLER, M., MORAK, M. AND WOLTRAN, S. 2016a. lpopt: A rule optimization tool for answer set programming. In *Proc. LOPSTR*, 114–130.

BICHLER, M., MORAK, M. AND WOLTRAN, S. 2016b. The power of non-ground rules in answer set programming. *TPLP 16,* 5–6, 552–569.

BICHLER, M., MORAK, M. AND WOLTRAN, S. 2018a. Single-shot epistemic logic program solving. In *Proceedings of IJCAI,* 1714–1720.

BICHLER, M., MORAK, M. AND WOLTRAN, S. 2018b. Single-shot epistemic logic program solving. In *Proceedings of ASPOCP*.

BLIEM, B., MOLDOVAN, M., MORAK, M. AND WOLTRAN, S. 2017. The impact of treewidth on ASP grounding and solving. In *Proceedings of IJCAI*, 852–858.

BODLAENDER, H. L. 1993. A tourist guide through treewidth. *Acta Cybernetica 11,* 1–2, 1–21.

BREWKA, G., EITER, T. AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

CALIMERI, F., FUSCÀ, D., PERRI, S. AND ZANGARI, J. 2017. I-DLV: The new intelligent grounder of DLV. *Intelligenza Artificiale 11,* 1, 5–20.

DEL CERRO, L. F., HERZIG, A. AND SU, E. I. 2015. Epistemic equilibrium logic. In *Proceedings of IJCAI*, 2964–2970.

EBBINGHAUS, H.-D. AND FLUM, J. 1995. *Finite Model Theory*. Springer Monographs in Mathematics. Springer, Berlin, Heidelberg.

EITER, T., FABER, W., FINK, M. AND WOLTRAN, S. 2007. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence 51,* 2–4, 123–165.

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

FABER, W., MORAK, M. AND WOLTRAN, S. 2019. Strong equivalence for epistemic logic programs made easy. In *Proceedings of AAAI*.

FABER, W., PFEIFER, G. AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1, 278–298.

FERRARIS, P., LEE, J. AND LIFSCHITZ, V. 2011. Stable models and circumscription. *Artificial Intelligence 175,* 1, 236–263.

GEBSER, M., KAMINSKI, R., KAUFMANN, B. AND SCHAUB, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool.

GEBSER, M., KAMINSKI, R., KAUFMANN, B. AND SCHAUB, T. 2014. *clingo* = ASP + control: Preliminary report. In *ICLP Technical Communications*

GEBSER, M., KAMINSKI, R., KAUFMANN, B. AND SCHAUB, T. 2019. Multi-shot ASP solving with clingo. *TPLP 19,* 1, 27–82.

GEBSER, M., KAMINSKI, R., KÖNIG, A. AND SCHAUB, T. 2011. Advances in *gringo* series 3. In *Proceedings of LPNMR,* 345–351.

GEBSER, M., KAUFMANN, B. AND SCHAUB, T. 2009. Solution enumeration for projected boolean search problems. In *Proceedings of CPAIOR*, 71–86.

GEBSER, M., KAUFMANN, B. AND SCHAUB, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187*, 52–89.

GELFOND, M. 1991. Strong introspection. In *Proc. AAAI, Vol. 1*, 386–391.

GELFOND, M. 1994. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence 12,* 1–2, 89–116.

GELFOND, M. 2011. New semantics for epistemic specifications. In *Proceedings of LPNMR*, 260–265.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP*, 1070–1080.

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

KAHL, P. T. 2014. Refining the semantics for epistemic logic programs. Ph.D. thesis, Texas Tech University, Texas, USA.

KAHL, P. T., LECLERC, A. P. AND SON, T. C. 2016. A parallel memory-efficient epistemic logic program solver: Harder, better, faster. In *Proceedings of ASPOCP*.

KAHL, P. T., WATSON, R., BALAI, E., GELFOND, M. AND ZHANG, Y. 2015. The language of epistemic specifications (refined) including a prototype solver. *Journal of Logic and Computation*, exv065.

LECLERC, A. P. AND KAHL, P. T. 2018. A survey of advances in epistemic logic program solvers. *CoRR abs/1809.07141*.

LIFSCHITZ, V., TANG, L. R. AND TURNER, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 369–389.

MORAK, M. AND WOLTRAN, S. 2012. Preprocessing of complex non-ground rules in answer set programming. In *ICLP Technical Communications*, 247–258.

PEARCE, D. 2006. Equilibrium logic. *Annals of Mathematics and Artificial Intelligence 47,* 1–2, 3–41.

PELOV, N., DENECKER, M. AND BRUYNOOGHE, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *TPLP 7,* 3, 301–353.

PULINA, L. 2016. The ninth QBF solvers evaluation – preliminary report. In *Proc. QBF*. CEUR Workshop Proceedings, vol. 1719. CEUR-WS.org, 1–13.

SHEN, Y. AND EITER, T. 2016. Evaluating epistemic negation in answer set programming. *Artificial Intelligence 237*, 115–135.

SHEN, Y., WANG, K., EITER, T., FINK, M., REDL, C., KRENNWALLNER, T. AND DENG, J. 2014. FLP answer set semantics without circular justifications for general logic programs. *Artificial Intelligence 213*, 1–41.

SON, T. C., LE, T., KAHL, P. T. AND LECLERC, A. P. 2017. On computing world views of epistemic logic programs. In *Proc. IJCAI*, 1269–1275.

TRUSZCZYNSKI, M. 2011. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, 315–333.

ZHANG, Z., WANG, B. AND ZHANG, S. 2015. Logic programming with graded introspection. In *Proceedings of ASPOCP*.