Structural Decompositions of Epistemic Logic Programs

Markus Hecher,[1,2] **Michael Morak,**[3] **Stefan Woltran**[1]
[1]TU Wien, Vienna, Austria,
[2]University of Potsdam, Potsdam, Germany
[3]University of Klagenfurt, Klagenfurt, Austria
{hecher, woltran}@dbai.tuwien.ac.at, michael.morak@aau.at

Abstract

Epistemic logic programs (ELPs) are a popular generalization of standard Answer Set Programming (ASP) providing means for reasoning over answer sets within the language. This richer formalism comes at the price of higher computational complexity reaching up to the fourth level of the polynomial hierarchy. However, in contrast to standard ASP, dedicated investigations towards tractability have not been undertaken yet. In this paper, we give first results in this direction and show that central ELP problems can be solved in linear time for ELPs exhibiting structural properties in terms of bounded treewidth. We also provide a full dynamic programming algorithm that adheres to these bounds. Finally, we show that applying treewidth to a novel dependency structure—given in terms of epistemic literals—allows to bound the number of ASP solver calls in typical ELP solving procedures.

1 Introduction

Epistemic logic programs (ELPs) (Gelfond and Przymusinska 1991), also referred to as the language of Epistemic Specifications (Gelfond 1991), have received renewed attention in the research community as of late. ELPs are an extension of the language of Answer Set Programming (ASP) (Brewka, Eiter, and Truszczynski 2011; Schaub and Woltran 2018) with epistemic operators. Gelfond (1991) introduced the operators $\mathbf{K}$ and $\mathbf{M}$ in order to represent the concepts of *known to be true* and *may be true*, and defined an initial semantics. Several improvements to the semantics have since been proposed in the literature (Gelfond 2011; Truszczynski 2011; Kahl et al. 2015; Fariñas del Cerro, Herzig, and Su 2015). Shen and Eiter (2016) realized that these two operators can be represented via a single negation-type operator that they called *epistemic negation*, denoted $\mathbf{not}$, and gave a new semantics based on this operator. Morak (2019) proposed a novel characterization of the central construct of the ELP semantics: the *world view*. While a recent analysis (Cabalar, Fandinno, and Fariñas del Cerro 2019) has shown that this semantics still does not eliminate all existing flaws, we will make use of it in this paper, since no clear "winner" semantics has as of yet emerged, and our approach should be equally applicable to other existing semantics that have been proposed.

Evaluating ELPs is a computationally hard task. The central decision problem, checking whether an ELP has a world view, is Σ_3^P-complete, and problems even higher on the polynomial hierarchy exist (Shen and Eiter 2016; Morak 2019). In order to deal with this high complexity efficiently, we propose to use a method from the field of parameterized complexity, namely, investigate how the runtime behaves when looking at different structural parameters of the problem. For standard ASP, this topic has received considerable interest, (Lonc and Truszczynski 2003; Gottlob, Pichler, and Wei 2010; Fichte and Szeider 2015; Bliem, Ordyniak, and Woltran 2016; Fichte, Kronegger, and Woltran 2019; Fichte and Hecher 2019). However, the parameterized complexity of epistemic ASP has remained largely unexplored so far. From the ASP case, we see strong evidence that this will be the case for ELPs as well. In this paper, we will investigate, in particular, whether ELPs can be solved efficiently if their treewidth (i.e., a measure for the tree-likeness of graphs) is bounded.

It turns out that this question can be answered in the affirmative: the main decision problems become tractable. In practice, a dynamic programming algorithm on tree decompositions can be used to exploit this directly, similarly to what was successfully proposed for ASP and QBF solvers (Fichte et al. 2017; Bliem et al. 2017; Fichte, Hecher, and Zisser 2019; Charwat and Woltran 2019). However, we also aim to investigate a more interesting angle. Many ELP solvers today work by making (up to exponentially many) calls to an underlying ASP solving system in order to check world view existence. Being able to find a bound on the number of these ASP solver calls would be very useful. Using so-called epistemic (primal) graphs of ELPs that focus on epistemically negated literals only, we can again employ treewidth to establish such bounds. This novel use of structural decomposition intuitively works well in some interesting cases including instances of the scholarship eligibility (SE)[1] benchmark set, as provided with the system "EP-ASP" (Son et al. 2017).

Using the epistemic primal graph representation, these

[1]Problem SE was a prime motivator for ELPs (Gelfond 1991).

instances naturally decompose into their individual subproblems, that is, one sub-instance of the SE problem for each student within the original instance.

Contributions. Our contributions are summarized below:

- We investigate the complexity of the ELP world view existence problem when parameterized by the treewidth of the ELP instance. We establish that this problem is fixed-parameter tractable in this setting, and, in fact, can be solved in linear time if the treewidth is bounded from above by a constant. The same holds for the even more complex problem of world view formula evaluation.
- Then, we propose a novel graph representation of ELPs, namely, the epistemic primal graph and show how this can be exploited to bound the number of calls to an underlying ASP solver in a classical ELP solver setting. It turns out that the number of calls is bounded in case the epistemic primal graph has bounded treewidth.
- Finally, we provide a full dynamic programming algorithm that could be used in practice to directly exploit the tractability result above. We also show that the worst-case runtime of this algoritm cannot be significantly improved under reasonable complexity-theoretic assumptions.

2 Preliminaries

Answer Set Programming (ASP). A *ground logic program* with nested negation (also called answer set program, ASP program, or, simply, logic program) is a pair $\mathscr{P} = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ is a set of propositional (i.e., ground) atoms and $\mathcal{R}$ is a set of rules of the form $a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg\ell_1, \ldots, \neg\ell_n$, where the comma symbol stands for conjunction, $0 \leq l \leq m$, $0 \leq n$, $a_i \in \mathcal{A}$ for all $1 \leq i \leq m$, and each ℓ_i is a *literal*, that is, either an atom a or its (default) negation $\neg a$ for any atom $a \in \mathcal{A}$.[2] Note that, therefore, doubly negated atoms may occur. We will sometimes refer to such rules as *standard rules*. Each rule $r \in \mathcal{R}$ consists of a *head* $H(r) = \{a_1, \ldots, a_l\}$ and a *body* $B(r) = \{a_{l+1}, \ldots, a_m, \neg\ell_1, \ldots, \neg\ell_n\}$, and is alternatively denoted by $H(r) \leftarrow B(r)$. The *positive* body is given by $B^+(r) = \{a_{l+1}, \ldots, a_m\}$. Sometimes, we add a set of rules $\mathcal{R}'$ to a logic program $\mathscr{P} = (\mathcal{A}, \mathcal{R})$. By some abuse of notation, let $\mathscr{P} \cup \mathcal{R}'$ denote the logic program $(\mathcal{A} \cup \mathcal{A}', \mathcal{R} \cup \mathcal{R}')$, where $\mathcal{A}'$ is the set of atoms occurring in the rules of $\mathcal{R}'$.

An *interpretation* I (over $\mathcal{A}$) is a set of atoms, that is, $I \subseteq \mathcal{A}$. A literal ℓ is true in an interpretation $I \subseteq \mathcal{A}$, denoted $I \vDash \ell$, iff $a \in I$ and $\ell = a$, or $a \notin I$ and $\ell = \neg a$; otherwise ℓ is false in I, denoted $I \nvDash \ell$. Finally, for some literal ℓ, we define that $I \vDash \neg\ell$ if $I \nvDash \ell$. This notation naturally extends to sets of literals. An interpretation M is called a *model* of r, denoted $M \vDash r$, if, whenever $M \vDash B(r)$, it holds that $M \vDash H(r)$. We denote the set of models of r by $mods(r)$; the models of a logic program $\mathscr{P} = (\mathcal{A}, \mathcal{R})$ are given by $mods(\mathscr{P}) = \bigcap_{r \in \mathcal{R}} mods(r)$. We also write $I \vDash r$ (resp. $I \vDash \mathscr{P}$) if $I \in mods(r)$ (resp. $I \in mods(\mathscr{P})$).

The GL-reduct of a logic program $\mathscr{P} = (\mathcal{A}, \mathcal{R})$ with respect to an interpretation I is given by $\mathscr{P}^I = (\mathcal{A}, \mathcal{R}^I)$ with $\mathcal{R}^I = \{H(r) \leftarrow B^+(r) \mid r \in \mathcal{R}, \forall(\neg\ell) \in B(r) : I \nvDash \ell\}$.

[2] In this case, we say that it is a literal *over* $\mathcal{A}$.

Definition 1 ((Gelfond and Lifschitz 1988; 1991; Lifschitz, Tang, and Turner 1999))**.** *$M \subseteq \mathcal{A}$ is an* answer set *of a logic program $\mathscr{P}$ if (1) $M \in mods(\mathscr{P})$ and (2) there is no subset $M' \subset M$ such that $M' \in mods(\mathscr{P}^M)$.*

The set of answer sets of a logic program $\mathscr{P}$ is denoted by $AS(\mathscr{P})$. The *consistency problem* of ASP, that is, to decide whether for a given logic program $\mathscr{P}$ it holds that $AS(\mathscr{P}) \neq \emptyset$, is Σ^2_P-complete (Eiter and Gottlob 1995), and remains so also in the case where doubly negated atoms are allowed in rule bodies (Pearce, Tompits, and Woltran 2009).

Epistemic Logic Programs. An *epistemic literal* is a formula $\mathbf{not}\,\ell$, where ℓ is a literal and $\mathbf{not}$ is the epistemic negation operator. A *ground epistemic logic program (ELP)* is a pair $\Pi = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ is a set of atoms and $\mathcal{R}$ is a set of *ELP rules*, which are implications of the 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 from $\mathcal{A}$, each ℓ_i is a literal over $\mathcal{A}$, and each ξ_i is an *epistemic literal* of the form $\mathbf{not}\,\ell$, where ℓ is a literal over $\mathcal{A}$. Similarly to logic programs, let $H(r) = \{a_1, \ldots, a_k\}$, and let $B(r) = \{\ell_1, \ldots, \ell_m, \xi_1, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n\}$. Further, $at(r) \subseteq \mathcal{A}$ denotes the set of atoms ocurring in ELP rule r, and $atel(r) \subseteq at(r)$ denotes the set of atoms used in epistemic literals of r. These notions naturally extend to sets of rules.

In order to define the semantics of an ELP, we will use the approach by Morak (2019), which follows the semantics defined in (Shen and Eiter 2016), but uses a different formal representation. Note that, however, our results can be adapted to other "reduct-based" semantics, by changing the definition of the reduct appropriately. Given an ELP $\Pi = (\mathcal{A}, \mathcal{R})$, a *candidate world interpretation (CWI)* I for Π is a consistent subset $I \subseteq \mathcal{L}$, where $\mathcal{L}$ is the set of all literals that can be built from atoms in $\mathcal{A}$. Note that a CWI I naturally gives rise to a three-valued truth assignment to all the atoms in $\mathcal{A}$; hence, we will sometimes treat a CWI I as a triple of disjoint sets $\langle I^P, I^N, I^U \rangle$, where $I^P = \{a \mid a \in I \cap \mathcal{A}\}$, $I^N = \{a \mid \neg a \in I\}$ and $I^U = (\mathcal{A} \setminus I^P) \setminus I^N$, with the intended meaning that atoms in I^P, I^N, and I^U are "always true," "always false," and "unknown," respectively.

With the above definition in mind, we now define when a CWI is compatible with a given set of interpretations.

Definition 2. *Let $\mathcal{I}$ be a set of interpretations over a set of atoms $\mathcal{A}$. Then, a CWI I is* compatible *with $\mathcal{I}$ iff we have:*

1. *$\mathcal{I} \neq \emptyset$;*
2. *for each atom $a \in I^P$, it holds that for each $J \in \mathcal{I}$, $a \in J$;*
3. *for each atom $a \in I^N$, we have for each $J \in \mathcal{I}$, $a \notin J$;*
4. *for each atom $a \in I^U$, it holds that there are $J, J' \in \mathcal{I}$, such that $a \in J$, but $a \notin J'$.*

The *epistemic reduct* (Shen and Eiter 2016; Morak 2019) of program $\Pi = (\mathcal{A}, \mathcal{R})$ w.r.t. a CWI I, denoted Π^I, is defined as $\Pi^I = (\mathcal{A}, \{r^I \mid r \in \mathcal{R}\})$ where r^I denotes rule r where each epistemic literal $\mathbf{not}\,\ell$ is replaced by $\neg\ell$ if $\ell \in I$, and by $\top$ otherwise. Note that, hence, Π^I is a plain logic program with all occurrences of epistemic negation removed.[3]

[3] In fact, Π^I may contain triple-negated atoms $\neg\neg\neg a$. Accord-

Now, a CWI I is a *candidate world view (CWV)* of Π iff I is compatible with the set $AS(\Pi^I)$ of answer sets. The set of CWVs of an ELP Π is denoted $cwv(\Pi)$. Following the principle of knowledge minimization, furthermore I is a *world view (WV)* iff it is a CWV and there is no proper subset $J \subset I$ such that $J \in cwv(\Pi)$. The set of WVs of an ELP Π is denoted $wv(\Pi)$.

One of the main reasoning tasks regarding ELPs is the *world view existence problem*, that is, given an ELP Π, decide whether $wv(\Pi) \neq \emptyset$ (or, equivalently, whether $cwv(\Pi) \neq \emptyset$). This problem is known to be Σ_3^P-complete (Shen and Eiter 2016; Morak 2019). Another interesting reasoning task is deciding, given an ELP $\Pi = (\mathcal{A}, \mathcal{R})$ and an arbitrary propositional formula φ over $\mathcal{A}$, whether φ holds in some WV, that is, whether there exists $W \in wv(\Pi)$ such that $W \vDash \varphi$. This *formula evaluation problem* is even harder, namely Σ_4^P-complete (Shen and Eiter 2016).

Tree Decompositions and Treewidth. We assume that graphs are undirected, simple, and free of self-loops. Let $G = (V, E)$ be a graph, T a rooted tree, and χ a labeling function that maps every node t of T to a subset $\chi(t) \subseteq V$ called the *bag* of t. The pair $\mathcal{T} = (T, \chi)$ is called a *tree decomposition (TD)* (Robertson and Seymour 1984) of G iff (i) for each $v \in V$, there exists a t in T, such that $v \in \chi(t)$; (ii) for each $\{v, w\} \in E$, there exists t in T, such that $\{v, w\} \subseteq \chi(t)$; and (iii) for each r, s, t of T, such that s lies on the unique path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. For a node t of T, we say that $\mathrm{type}(t)$ is leaf if t has no children and $\chi(t) = \emptyset$; join if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; intr ("introduce") if t has a single child t', $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; rem ("removal") if t has a single child t', $\chi(t') \supseteq \chi(t)$ and $|\chi(t')| = |\chi(t)| + 1$. If for every node $t \in T$, $\mathrm{type}(t) \in \{\mathsf{leaf}, \mathsf{join}, \mathsf{intr}, \mathsf{rem}\}$, then (T, χ) is called *nice*. The *width* of a TD is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width over all TDs of G. Note that if G is a tree, then $tw(G) = 1$.

Monadic Second Order Logic and Courcelle's Theorem. Monadic Second Order logic (MSO) extends First Order logic (FO) with set variables that range over sets of domain elements. Atomic MSO-formulas over a signature σ are either (1) atoms over some predicate in σ; (2) equality atoms; or (3) atoms of the form $x \in S$, where x is a FO variable, and S is a set variable. MSO-formulas are closed under FO operators. It is convenient to use symbols like $\notin$, $\subseteq$, $\subset$, $\cap$, or $\cup$, with the obvious meanings as abbreviations for the corresponding MSO (sub-)formulas. A σ-structure $\mathfrak{A}$ is a set of atoms over predicates in σ. Let $dom(\mathfrak{A})$ denote its domain.

In order to exploit the structural information, we need to define how logical structures can be represented as graphs, and how their treewidth is then defined: Given a structure $\mathfrak{A}$ over some logical signature σ of arity at most two (sufficent for us), we say that the treewidth of $\mathfrak{A}$ equals $tw(G_{\mathfrak{A}})$, where $G_{\mathfrak{A}} = (V, E)$ is a graph with $V = dom(\mathfrak{A})$ and edge $\{a, b\} \in E$ iff $r(a, b) \in \mathfrak{A}$, where r is some relation in σ.

MSO formulas over structures of bounded treewidth are important in the context of parameterized complexity in order to establish running time bounds, as the following landmark theorem by Courcelle shows:

Theorem 3 ((Courcelle 1990))**.** *Let φ be a fixed MSO formula over signature σ and let $\mathfrak{A}$ be a σ-structure with $tw(\mathfrak{A}) \leqslant k$, for some integer k. Then, evaluating φ over $\mathfrak{A}$ can be done in time $O(f(k) \cdot |\mathfrak{A}|)$, for some function f not depending on $|\mathfrak{A}|$.*

Problems with a parameter k that can be solved in time $O(f(k) \cdot n^c)$, where c is a constant and f only depends on k, are called *fixed-parameter tractable (FPT)* (Downey and Fellows 1999).

3 An MSO Encoding for ELPs

The main objective in this section is to investigate how the semantics of ELPs can be encoded in terms of an MSO formula and thereby investigate, from a theoretical perspective, the time complexity of evaluating ELPs, specifically looking at tree-like instances.

Now, our goal will be to offer a fixed MSO encoding to exploit Theorem 3, in the spirit of (Gottlob, Pichler, and Wei 2010), which is able to solve the world view existence problem for an ELP by evaluating it over a suitable logical structure representing the ELP. In order to begin the construction of this, we first need to fix the signature over which our MSO encoding will be expressed. To this end, let signature $\sigma = \{atom, rule, h, b, b^{\neg}, b^{\mathbf{not}}, b^{\mathbf{not}\,\neg}, b^{\neg\mathbf{not}}, b^{\neg\mathbf{not}\,\neg}\}$, where $atom(a)$ and $rule(r)$ represent the fact that domain elements a and r are an atom and a rule, respectively; where $h(a, r)$ represents that atom a appears in the head of rule r; and where $b^{\square}(a, r)$, with $\square \in \{\varepsilon, \neg, \mathbf{not}\,, \neg\mathbf{not}\,, \mathbf{not}\,\neg, \neg\mathbf{not}\,\neg\}$ and ε the empty word, represents that fact that the sub-formula $\square a$, for atom a, appears in the body of rule r. Next we construct the encoding.

Lemma 4. *Consider the signature σ above. WV existence can be expressed by means of a fixed MSO formula over σ.*

Proof. Recall that, in order to check the existence of a WV, it suffices to check the existence of a CWV (since WVs are simply subset-minimal CWVs). We will construct an MSO formula $cwv(\mathbf{P}, \mathbf{N}, \mathbf{U})$ with the intended meaning that it evaluates to true iff the input set variables $\mathbf{P}$, $\mathbf{N}$, and $\mathbf{U}$ represent a CWV W with $W^P = \mathbf{P}$, $W^N = \mathbf{N}$, and $W^U = \mathbf{U}$. To this end, our formula will be of the following form:
$cwv(\mathbf{P}, \mathbf{N}, \mathbf{U}) \equiv cwi(\mathbf{P}, \mathbf{N}, \mathbf{U}) \wedge \bigwedge_{i=1}^{4} chk_i(\mathbf{P}, \mathbf{N}, \mathbf{U})$
where cwi ensures that $\mathbf{P}$, $\mathbf{N}$, and $\mathbf{U}$ indeed encode a valid CWI (i.e., a three-partition of the set of atoms stored in $atom$), and chk_i verifies that Condition i of Definition 2 holds. We will now give the construction of these checks.

First, the check for a valid CWI is expressed as follows:
$cwi(\mathbf{P}, \mathbf{N}, \mathbf{U}) \equiv \forall X\, (atom(X) \Leftrightarrow X \in \mathbf{P} \cup \mathbf{N} \cup \mathbf{U}) \wedge$
$\neg\exists X\, ((X \in \mathbf{P} \cap \mathbf{N}) \vee (X \in \mathbf{N} \cap \mathbf{U}) \vee (X \in \mathbf{P} \cap \mathbf{U}))$

ing to (Lifschitz, Tang, and Turner 1999), such formulas are equivalent to simple negated atoms $\neg a$, and we treat them as such.

The four remaining checks have a similar structure:

$$chk_1(\mathbf{P},\mathbf{N},\mathbf{U}) \equiv \exists \mathbf{X}\, as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U});$$
$$chk_2(\mathbf{P},\mathbf{N},\mathbf{U}) \equiv \forall X\,(X \in \mathbf{P} \Rightarrow \forall \mathbf{X}\,(as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \Rightarrow X \in \mathbf{X}));$$
$$chk_3(\mathbf{P},\mathbf{N},\mathbf{U}) \equiv \forall X\,(X \in \mathbf{N} \Rightarrow \forall \mathbf{X}\,(as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \Rightarrow X \notin \mathbf{X}));$$
$$chk_4(\mathbf{P},\mathbf{N},\mathbf{U}) \equiv \forall X\,(X \in \mathbf{U} \Rightarrow (\exists \mathbf{X}\,(as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \wedge X \in \mathbf{X}) \wedge \exists \mathbf{X}\,(as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \wedge X \notin \mathbf{X}))).$$

The four checks encode precisely the conditions of Definition 2, where $as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U})$ is a sub-formula, to be defined below, that expresses that $\mathbf{X}$ is an answer set of the epistemic reduct w.r.t. the CWI represented by the sets $\mathbf{P}$, $\mathbf{N}$, and $\mathbf{U}$. For example, chk_3 encodes that for each atom X that is set to "always false" in the CWI (i.e., $X \in \mathbf{N}$), it must hold that for every stable model $\mathbf{X}$ of the epistemic reduct, X must not be true in that stable model (i.e., $X \notin \mathbf{X}$).

It now remains to define the sub-formula for checking answer sets. This construction is based on the one presented in (Gottlob, Pichler, and Wei 2010, Theorem 3.5), but adapted to take the computation of the epistemic reduct into account. Firstly, a set of atoms M is an answer set if it is a model and no proper subset of M is a model of the GL-reduct w.r.t. M. This is expressed as follows (note that any model M is also a model of its GL-reduct):

$$as(\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \equiv gl(\mathbf{X},\mathbf{X},\mathbf{P},\mathbf{N},\mathbf{U}) \wedge \forall \mathbf{Y}\,(\mathbf{Y} \subset \mathbf{X} \Rightarrow \neg gl(\mathbf{X},\mathbf{Y},\mathbf{P},\mathbf{N},\mathbf{U})).$$

The intuitive meaning of $gl(\mathbf{X},\mathbf{Y},\mathbf{P},\mathbf{N},\mathbf{U})$ is that it should hold iff $\mathbf{Y}$ is a model of the GL-reduct w.r.t. $\mathbf{X}$ of the epistemic reduct w.r.t. the CWI represented by $\mathbf{P}$, $\mathbf{N}$, and $\mathbf{U}$:

$$\begin{aligned} gl(\mathbf{X},\mathbf{Y},\mathbf{P},\mathbf{N},\mathbf{U}) \equiv \forall R\,(rule(R) \Rightarrow \exists Z\,(&\\ (h(Z,R) \wedge Z \in \mathbf{Y})&\\ \vee (b(Z,R) \wedge Z \notin \mathbf{Y}) \vee (b^{\neg}(Z,R) \wedge Z \in \mathbf{X}))&\\ \vee (b^{\mathbf{not}}(Z,R) \wedge Z \in \mathbf{P} \wedge Z \in \mathbf{X})&\\ \vee (b^{\mathbf{not}\,\neg}(Z,R) \wedge Z \in \mathbf{N} \wedge Z \notin \mathbf{X})&\\ \vee (b^{\neg\mathbf{not}}(Z,R) \wedge ((Z \in \mathbf{N} \cup \mathbf{U}) \vee (Z \in \mathbf{P} \wedge Z \notin \mathbf{X})))&\\ \vee (b^{\neg\mathbf{not}\,\neg}(Z,R) \wedge ((Z \in \mathbf{P} \cup \mathbf{U}) \vee (Z \in \mathbf{N} \wedge Z \in \mathbf{X})))))& \end{aligned}$$

Note that the definition of the gl-relation is such that it precisely mirrors the definition of both the epistemic reduct and the GL-reduct. It amounts to checking that every rule in the GL-reduct is satisfied, and amounts to a large case distinction, dealing with all seven cases of how atoms can appear in a rule (that is, either in the head, or nested under six combinations of default and epistemic negation in the body). For example, in line three, the first disjunct says that rule R is satisfied if there is an atom Z in the positive body, but this atom is not present in the reduct model $\mathbf{Y}$ (satisfying rule R by not satisfying the body). Line four treats the case of an epistemically negated atom Z in the body. Such a rule is satisfied iff Z is set to "always true" in the CWI (since otherwise the epistemic literal is replaced by $\top$ in the epistemic reduct, and the rule cannot be satisfied solely by this body element in this case), and Z is false in the original model $\mathbf{X}$ (since in the epistemic reduct, the epistemic negation turns into default negation in this case, and default-negated atoms are evaluated over the original model $\mathbf{X}$).

This completes our MSO encoding. Correctness follows by construction, as explained above. In order to solve the WV existence problem via this encoding, we simply have to quantify the relevant set variables:

$$\varphi = \exists \mathbf{P} \exists \mathbf{N} \exists \mathbf{U}\; cwv(\mathbf{P},\mathbf{N},\mathbf{U}).$$

Evaluating this formula over a σ-structure $\mathfrak{P}$ that represents an ELP Π, we get that Π has a CWV iff $\mathfrak{P} \models \varphi$. □

With the above reduction in mind, lets take a closer look at what worst-case solving time guarantees we can give for solving ELPs, in particular w.r.t. structural properties. Let $\Pi = (\mathcal{A},\mathcal{R})$ be an ELP, and let $\mathfrak{P}$ be the σ-structure that represents it. Recall that $tw(\mathfrak{P}) = tw(G_{\mathfrak{P}})$. In our case, $G_{\mathfrak{P}}$ coincides with the so-called *incidence graph* of the ELP Π, a graph representation that is well-known and studied in the literature for a wide range of logic-based formalisms, and, in particular, for ASP (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017). The incidence graph of an ELP Π is the graph $G = (V,E)$ with $V = \mathcal{A} \cup \mathcal{R}$ and $\{a,r\} \in E$ iff atom $a \in \mathcal{A}$ occurs in rule $r \in \mathcal{R}$ in Π. It is not difficult to verify that the σ-structure $\mathfrak{P}$, when represented as a graph, mirrors the incidence graph of Π precisely. From this correspondence, Theorem 3, and Lemma 4, we thus obtain the following, fundamental parameterized complexity result:

Theorem 5. *Let Π be an ELP, let G be its incidence graph, and let $tw(G) \leqslant k$, for some integer k. Then, checking whether $cwv(\Pi) \neq \emptyset$ can be done in time $O(f(k) \cdot |\Pi|)$, for some function f that does not depend on $|\Pi|$.*

Using a simple extension of the MSO construction in the proof of Lemma 4, we can state a similar result for the formula evaluation problem. The MSO formula

$$\exists \mathbf{P} \exists \mathbf{N} \exists \mathbf{U}\; cwv(\mathbf{P},\mathbf{N},\mathbf{U}) \wedge entails(\mathbf{P},\mathbf{N},\mathbf{U},\varphi) \wedge \neg\,(\exists \mathbf{P}' \exists \mathbf{N}' \exists \mathbf{U}'\,(\mathbf{P}' \subset \mathbf{P} \vee \mathbf{N}' \subset \mathbf{N}) \wedge cwv(\mathbf{P}',\mathbf{N}',\mathbf{U}'))$$

checks whether there is at least one WV that satisfies formula φ, where the atom $entails(\mathbf{P},\mathbf{N},\mathbf{U},\varphi)$ encodes the check that the WV represented by $\mathbf{P}$, $\mathbf{N}$, and $\mathbf{U}$ cautiously entails formula φ, a straightforward model-checking construction left to the interested reader. We obtain the following by the same argument as the one for Theorem 5:

Theorem 6. *Let Π, G, and k be as in Theorem 5, and let φ be a propositional formula. Then, checking whether Π has a WV that cautiously entails φ can be done in time $O(f(k) \cdot |\Pi|)$, for some function f that does not depend on $|\Pi|$.*

From the above theorems we immediately obtain the fact that ELPs of bounded treewidth can be solved in linear time in the size of the ELP. We will investigate how to exploit this result and pinpoint function $f(k)$ in the next two sections.

4 Bounding Calls to Standard ASP Solvers

Before providing a concrete algorithm for the FPT result in Theorem 5 we will investigate a more abstract approach. Many ELP solvers today make use of standard ASP solvers to check the compatibility of a CWI with the set of answer sets of its epistemic reduct. However, the number of calls to such an ASP solver can be at worst exponential. In this section, we will propose an algorithm that makes use

of the structural relationships between the epistemic literals in an ELP in order to control the number of ASP solver calls needed and give finer-grained worst-case bounds on this number. In the next section, we will then extend these concepts to a full dynamic programming algorithm that exploits the result in Theorem 5.

We first need to define the structural relationship between atoms occurring in epistemic literals in an ELP. To this end, let $\Pi = (\mathcal{A}, \mathcal{R})$ be an ELP. Then, the *primal graph* $\mathbb{P}_\Pi = (V, E)$ of Π is a graph with $V = \mathcal{A}$ and $\{a, b\} \in E$ iff atoms a and b with $a \neq b$ appear together in a rule in $\mathcal{R}$, that is, iff $\exists r \in \mathcal{R} : \{a, b\} \subseteq at(r)$. Two vertices a, b in the primal graph are *non-epistemically connected* iff there is a path $\langle a, v_1, \ldots, v_n, b\rangle$ with $n \geq 0$ in $\mathbb{P}_\Pi$, such that each vertex v_i, $1 \leq i \leq n$, belongs to the set $\mathcal{A} \setminus atel(\Pi)$. Now, the *epistemic primal graph* $\mathbb{E}_\Pi = (V, E)$ of Π is a graph with the vertex set $V = atel(\Pi)$ being the set of atoms appearing in epistemic literals in Π, and an edge $\{a, b\} \in E$ iff $a \neq b$ and vertices a, b are non-epistemically connected in $\mathbb{P}_\Pi$. Intuitively, two atoms from $atel(\Pi)$ form an edge in $\mathbb{E}_\Pi$ iff they are connected in $\mathbb{P}_\Pi$ via atoms that *do not* appear in epistemic literals. The concept of epistemic primal graph is inspired by the notion of the *torso graph* (Ganian, Ramanujan, and Szeider 2017), which is used in parameterized complexity to decompose certain abstraction graphs.

Example 7. *Consider the classic* scholarship eligibility problem *encoding, first investigated by Gelfond (1991):*
$eligible(X) \leftarrow highGPA(X)$
$ineligible(X) \leftarrow lowGPA(X)$
$\bot \leftarrow eligible(X), ineligible(X)$
$interview(X) \leftarrow \mathbf{not}\ eligible(X), \mathbf{not}\ ineligible(X).$
Now, assume the above abstract (non-ground) program is instantiated with two students (assume that it is copied twice and $mike$ *and* $mark$ *are substituted for* X*), and that we add the following rules, resulting in ELP* Π*:*
$lowGPA(mike) \vee highGPA(mike)$
$lowGPA(mark) \vee highGPA(mark)$
Epistemic primal graph $\mathbb{E}_\Pi$ *contains only four nodes:* $eligible(mike)$*,* $ineligible(mike)$*, and the same two for* $mark$*. Further,* $\mathbb{E}_\Pi$ *does not have any edges except an edge between the two* $mike$*-atoms, and the same for* $mark$*. Since* $\mathbb{E}_\Pi$ *forms a forest, the treewidth of* $\mathbb{E}_\Pi$ *is* 1.

While the epistemic primal graph does not directly provide new complexity results, it will allow us to give firm guarantees on the number of ASP solver calls needed. As a side-effect, this algorithm is conceptually simpler than the one of the next section, but prepares ideas for later.

Algorithms that solve problems of bounded treewidth typically proceed by *dynamic programming (DP)*, bottom-up, along a TD where at each node t of the TD information is gathered (Bodlaender and Kloks 1996) in a table $\tau(t)$. A *table* $\tau(t)$ is a set of rows, where a *row* in $\tau(t)$ is a fixed-length sequence of elements. Tables are derived by an algorithm executed in each bag, called *bag algorithm*, which determine the actual content and meaning of the rows. Then, the DP approach DP_B for an epistemic logic program Π and a given bag algorithm B performs the following steps:

1. Construct graph representation G of Π that is used by B.
2. Heuristically compute a (nice) TD $\mathcal{T} = (T, \chi)$ of G.
3. Execute B for every node t in TD $\mathcal{T}$ in post-order. As input, B takes a node t, a bag $\chi(t)$, a *solving program* (depending on $\chi(t)$ and B), which is the part of Π currently visible in t, and the tables computed at children of t. Bag algorithm B outputs a table $\tau(t)$.
4. Print the result by interpreting the table for root n of T.

Next, we define a bag algorithm EPRIM for the epistemic primal graph representation of Π. To this end, let $\Pi = (\mathcal{A}, \mathcal{R})$ be the given input epistemic program, $\mathcal{T} = (T, \chi)$ be a nice TD of $\mathbb{E}_\Pi$, t be a node of $\mathcal{T}$, and $\prec$ be any arbitrary total ordering among the nodes in $\mathcal{T}$. To ease notation, for some set $X \subseteq atel(\mathcal{R})$, let $conn(X)$ be the set of vertices (i.e., atoms) from $\mathbb{P}_\Pi$ that lie on a path that non-epistemically connects any two vertices a and b in X.[4] We now define the *induced bag rules* for node t of $\mathcal{T}$, denoted by $\mathcal{R}_t^\mathbb{E}$, as follows. For every rule $r \in \mathcal{R}$, r is *compatible* with node t of $\mathcal{T}$ iff (a) $at(r) \cap conn(atel(\mathcal{R})) \subseteq conn(\chi(t))$, and (b) $\chi(t)$ is subset-maximal among all nodes of $\mathcal{T}$. Now, $r \in \mathcal{R}_t^\mathbb{E}$ iff t is the $\prec$-minimal node among all nodes t' in $\mathcal{T}$ with $\mathrm{type}(t') = \mathsf{intr}$ compatible with r. The *induced bag program* for node t is the ELP $\Pi_t^\mathbb{E} = (at(\mathcal{R}_t^\mathbb{E}), \mathcal{R}_t^\mathbb{E})$.

Observe that any set of vertices that form a clique within $\mathbb{E}_\Pi$ will appear together in some node t of $\mathcal{T}$. Note that for each node t of $\mathcal{T}$ that has not a non-subset-maximal bag, or has one, but is not $\prec$-minimal for any compatible $r \in \mathcal{R}$, the induced bag program is empty. Therefore, we have that for each rule $r \in \mathcal{R}$ there is exactly one node t of $\mathcal{T}$ where $r \in \mathcal{R}_t^\mathbb{E}$, and, even more stringent, that each atom $a \in at(r) \setminus atel(r)$ appears only in the induced bag program of t, but not in any other node. The bag algorithm EPRIM uses the induced bag program as its solving program, and, following the argument above, can check all rules containing atoms from $\mathcal{A} \setminus atel(\Pi)$ in a single node. Hence, during its traversal of the tree decomposition, it does not need to store anything about these atoms. Instead, every row computed as part of a table by EPRIM for a node t, called an *epistemic row*, is of the form $\langle I\rangle$, where $I \subseteq 2^{\{a, \neg a \mid a \in \chi(t)\}}$ is a *partial CWI* (that is, a CWI restricted to and defined w.r.t. $\chi(t)$)[5].

Listing 1 presents algorithm EPRIM. For the ease of presentation, it deals with nice TDs only, but can be generalized to arbitrary TDs, requiring a more involved case distinction. Intuitively, since for each leaf node t we have $\chi(t) = \emptyset$, bag algorithm EPRIM ensures in Line 1 that $\tau(t)$ consists only of the empty epistemic row. Then, when an atom a appears in bag $\chi(t)$ for a node t, but does not occur in child bags, CWI J, with either $a \in J^P$, $a \in J^N$, or $a \in J^U$, is computed in Line 3. Further, if the solving program with rules $\mathcal{R}_t^\mathbb{E}$ is not empty, i.e., t is the unique node responsible for evaluating all the rules in $\mathcal{R}_t^\mathbb{E}$, the four conditions of Definition 2 are checked in Line 4. Note that these checks can be performed by calling a black-box ASP solvers a limited number of times for each row in t:

[4] Note that we may have that $a = b$, and hence, $conn(\{a\})$ contains all those vertices from $\mathcal{A} \setminus atel(\Pi)$ connected to atom a.

[5] This also means that J^P, J^N, and J^U are defined w.r.t. $\chi(t)$.

Listing 1: Bag algorithm $\mathsf{EPRIM}(t, \chi_t, \Pi_t^{\mathbb{E}}, \langle \tau_1, \ldots \rangle)$ for nice TDs of the epistemic primal graph representation.

In: Node t, bag χ_t, induced bag program $\Pi_t^{\mathbb{E}}$, tables $\langle \tau_1, \ldots \rangle$ for child nodes $\langle t_1, \ldots \rangle$ of t. **Out:** Table $\tau(t)$.

1 **if** $\mathrm{type}(t) =$ *leaf* **then** $\tau(t) = \{\langle \emptyset \rangle\}$ `/* Abbrevs. below */`
2 **else if** $\mathrm{type}(t) =$ *intr, and $a \in \chi_t$ is introduced* **then**
3 $\quad \tau(t) = \{\langle J \rangle \mid \langle I \rangle \in \tau_1, J \in \{I_a^+, I_{\neg a}^+, I\}, \mathscr{P} = (\Pi_t^{\mathbb{E}})^J, \mathcal{R}_t^{\mathbb{E}} = \emptyset$ or
4 $\quad\quad [\, AS(\mathscr{P}) \neq \emptyset, AS(\mathscr{P} \cup \{\bot \leftarrow (J^P \cup \overline{J^N})\}) = \emptyset,$
$\quad\quad\quad$ for every $b \in J^U$: $AS(\mathscr{P} \cup \{\bot \leftarrow b\}) \neq \emptyset,$
$\quad\quad\quad\quad AS(\mathscr{P} \cup \{\bot \leftarrow \neg b\}) \neq \emptyset\,]\}$
5 **else if** $\mathrm{type}(t) =$ *rem, and $a \notin \chi_t$ is removed* **then**
6 $\quad \tau(t) = \{\langle I_a^- \rangle \mid \langle I \rangle \in \tau_1\}$
7 **else if** $\mathrm{type}(t) =$ *join* **then** $\tau(t) = \tau_1 \bowtie \tau_2$

$S_e^+ := S \cup \{e\}, S_e^- := S \setminus \{e, \neg e\}, \overline{S} := \{\neg s \mid s \in S\}$.

Proposition 8. *To compute a row in a table of* EPRIM*, an ASP solver needs to be called at most $2 + 2 \cdot |\chi(t)|$ times.*

On an abstract level, bag algorithm EPRIM hence provides a method for solving epistimic logic programs Π by means of plain ASP solvers based on the structure of the epistemic literals of Π. Whenever an epistemic atom a is removed in node t, indicating that a does not occur in any ancestor bag of t, information about the "role" of a in any CWI is not needed anymore. Finally, for join nodes, Line 7 ensures that the CWIs in $\chi(t)$ coincide with the ones that both child bags have in common. The last step is the evaluation of the root node. If in the root a non-empty table is computed by EPRIM, then the input ELP Π has a CWV.

Example 9. *The epistemic primal graph from Example 7 is a "best case"-scenario for using our TD-based approach: the TD naturally separates the ELP into one part each for the two students, and algorithm* EPRIM *would evaluate the two completely separately, which is exactly what intuition would tell us to do. However, standard ELP solvers seem to struggle in this setting when the number of students increases; cf. e.g. (Bichler, Morak, and Woltran 2018).*

From Proposition 8, and the facts that there are only linearly many TD nodes in the size of the input ELP Π, and that the number of rows per tree node is at most exponential in the treewidth of $\mathbb{E}_\Pi$, we obtain the following statement:

Theorem 10. *Given an input ELP Π of size n, algorithm $\mathsf{DP}_{\mathsf{EPRIM}}$ described above makes at most $\mathcal{O}(2^k \cdot n)$ calls to an underlying ASP solver, where $k = tw(\mathbb{E}_\Pi)$.*

Correctness of the algorithm presented above can be established along the same lines as for established TD-based dynamic programming algorithms for ASP (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017). A more formal correctness argument will be given in the next section.

5 A Full Dynamic Programming Algorithm

In this section, we will extend the EPRIM algorithm in such a way that it no longer relies on an underlying ASP solver, but solves an ELP completely on its own, using dynamic programming. This new algorithm, PRIM, will operate on the primal graph, instead of on the epistemic primal graph, and makes use of features of the entire ELP structure.

Recall that the primal graph is defined on all atoms of an ELP, instead of just on the ones appearing in epistemic literals. As a result, we need to define a different solving program for TD nodes. To this end, for the remainder of the section, assume we are a given ELP $\Pi = (\mathcal{A}, \mathcal{R})$ to solve. Further, let $\mathcal{T} = (T, \chi)$ be a nice TD of the primal graph $\mathbb{P}_\Pi$ of Π, and t a node of $\mathcal{T}$. Then, the *bag rules* for t, denoted $\mathcal{R}_t$ are defined as the set $\{r \mid r \in \mathcal{R}, at(r) \subseteq \chi(t)\}$, that is, all the rules of Π that are completely "covered" by $\chi(t)$. Further, the *bag program* of t is defined as $\Pi_t = (\mathcal{A} \cap \chi(t), \mathcal{R}_t)$.

In order to define PRIM, we need to define what a row of a table for a TD node t looks like. Since PRIM, in contrast to EPRIM, now also needs to compute the answer sets underlying a CWV, we start with the following, preliminary definition. Let $M \subseteq \chi(t)$ be an interpretation and $\mathcal{C} \subseteq 2^{\chi(t)}$ a set of interpretations w.r.t. $\chi(t)$. Then, we refer to a tuple $\langle M, \mathcal{C} \rangle$ as an *answer set tuple*. This construct, proposed in (Fichte et al. 2017), directly follows the definition of answer sets as in Definition 1, namely, (1) set M, called a *witness*, is used for storing (parts of) an answer set candidate of some ASP program, and (2) set $\mathcal{C}$, called *counterwitnesses*, holds a set of (partial) models of the GL-reduct w.r.t. M that are potential subsets of M, and hence may be counter-examples to M being extendable to an answer set. An answer set tuple with an empty set of counterwitnesses is referred to as *proving answer set tuple*, which, vaguely speaking, proves that M can be indeed extended to an answer set of some ASP program, which the tuple was constructed for. Answer set tuples are used by algorithm PRIM in order to "transport" information—in the form of parts of models restricted to the respective bags—of already evaluated rules of the ASP program from the leaves towards the root during TD traversal.

With this definition in mind, we are now ready to define a row for node t used in algorithm PRIM. Such a row, called *primal row*, is of the form $\langle I, \mathcal{M}, \mathcal{K}, \mathcal{S} \rangle$, where I corresponds to a CWI restricted to $\chi(t)$ as in EPRIM, and sets $\mathcal{M}, \mathcal{K}, \mathcal{S}$ consist of answer set tuples, where $\mathcal{M}$ represents a set of possible witness answer sets for Condition 1 of Definition 2, $\mathcal{K}$ represents possible witnesses for disproving Conditions 2 and 3, and $\mathcal{S}$ represents possible witnesses for guaranteeing Condition 4. In the root node n of the TD, a specific primal row $\vec{u} \in \tau(n)$ is required in table $\tau(n)$ to answer the question of WV existence of Π positively, and PRIM is designed to maintain primal rows accordingly. The set $\mathcal{M}$ of answer set tuples in $\vec{u}$ is used for ensuring Condition (1) of Definition 2, where a proving answer set tuple in $\mathcal{M}$ gives rise to an answer set of some ASP program $\Pi^{I'}$ of some extension $I' \supseteq I$ of I. For ensuring Conditions (2) and (3), the set $\mathcal{K}$ in $\vec{u}$ shall not contain any proving program tuples, i.e., proving program tuples of $\mathcal{K}$ are required to vanish (get "killed") during the TD traversal, otherwise Conditions (2) or (3) would be violated. Finally, $\mathcal{S}$ in $\vec{u}$ serves to establish Condition (4), where no non-proving answer set tuple is allowed, that is, each answer set tuple needs to "survive".

Definition 11. *A primal row $\langle I, \mathcal{M}, \mathcal{K}, \mathcal{S} \rangle$ is* proving *if (1) there is a $\langle M, \mathcal{C} \rangle \in \mathcal{M}$ with $\mathcal{C} = \emptyset$, (2) there is no $\langle M, \mathcal{C} \rangle \in \mathcal{K}$ with $\mathcal{C} = \emptyset$, and (3) there is no $\langle M, \mathcal{C} \rangle \in \mathcal{S}$ with $\mathcal{C} \neq \emptyset$.*

Listing 2: Bag algorithm $\mathsf{PRIM}(t, \chi_t, \Pi_t, \langle \tau_1, \ldots \rangle)$ for nice TDs of the primal graph representation.

In: Node t, bag χ_t, bag program Π_t, $\langle \tau_1, \ldots \rangle$ is the seq. of tables for child nodes $\langle t_1, \ldots \rangle$ of t. **Out:** Table $\tau(t)$.

1 **if** $\mathrm{type}(t) = \mathit{leaf}$ **then** $\tau(t) = \{\langle \emptyset, \{\langle \emptyset, \emptyset \rangle\}, \emptyset, \emptyset \rangle\}$

2 **else if** $\mathrm{type}(t) = \mathit{intr}$, $a \in \chi_t$ *introduced,* $a \notin \mathit{atel}(\Pi_t)$ **then**

3 $\quad \tau(t) = \{\langle I, \mathcal{M}', \mathcal{K}', \mathcal{S}' \rangle \mid \langle I, \mathcal{M}, \mathcal{K}, \mathcal{S} \rangle \in \tau_1, \mathscr{P} = (\Pi_t)^I, \mathcal{M}' = \mathrm{intTs}(a, \mathcal{M}, \mathscr{P}), \mathcal{K}' = \mathrm{intTs}(a, \mathcal{K}, \mathscr{P}),$

4 $\quad\quad \mathcal{S}' \in \mathrm{succS}(a, \mathcal{M}', \mathcal{S}, \mathscr{P})\}$

5 **else if** $\mathrm{type}(t) = \mathit{intr}$, $a \in \chi_t$ *introduced,* $a \in \mathit{atel}(\Pi_t)$ **then**

6 $\quad \tau(t) = \{\langle J, \mathcal{M}', \mathcal{K}' \cup \mathcal{K}'', \mathcal{S}' \cup \mathcal{S}'' \rangle \mid \langle I, \mathcal{M}, \mathcal{K}, \mathcal{S} \rangle \in \tau_1, \mathscr{P} = (\Pi_t)^J, \mathcal{M}' = \mathrm{intTs}(a, \mathcal{M}, \mathscr{P}), \mathcal{K}' = \mathrm{intTs}(a, \mathcal{K}, \mathscr{P}),$

7 $\quad\quad \mathcal{S}' \in \mathrm{succS}(a, \mathcal{M}', \mathcal{S}, \mathscr{P}), J \in \{I_a^+, I_{\neg a}^+, I\},$

8 $\quad\quad \mathcal{K}'' = \mathrm{intTs}(a, \bigcup_{S \in \mathcal{M}} \{S \mid a \in J^P\}, \mathscr{P} \cup \{\bot \leftarrow a\}) \cup \mathrm{intTs}(a, \bigcup_{S \in \mathcal{M}} \{S \mid a \in J^N\}, \mathscr{P} \cup \{\bot \leftarrow \neg a\}),$

9 $\quad\quad \{M', M''\} \subseteq \mathcal{M}', \mathcal{S}'' = \{M', M'' \mid a \in J^U\} = \{M', M'' \mid a \in J^U, M' \vDash \mathscr{P} \cup \{\bot \leftarrow a\}, M'' \vDash \mathscr{P} \cup \{\bot \leftarrow \neg a\}\}\}$

10 **else if** $\mathrm{type}(t) = \mathit{rem}$, *and* $a \notin \chi_t$ *is removed* **then**

11 $\quad \tau(t) = \{\langle I_a^-, \mathcal{M}_a^\sim, \mathcal{K}_a^\sim, \mathcal{S}_a^\sim \rangle \mid \langle I, \mathcal{M}, \mathcal{K}, \mathcal{S} \rangle \in \tau_1\}$

12 **else if** $\mathrm{type}(t) = \mathit{join}$ **then**

13 $\quad \tau(t) = \{\langle I, \mathcal{M}_1 \sqcap \mathcal{M}_2, [\mathcal{K}_1 \sqcap (\mathcal{K}_2 \cup \mathcal{M}_2)] \cup [\mathcal{K}_2 \sqcap (\mathcal{K}_1 \cup \mathcal{M}_1)], [\mathcal{S}_1 \sqcap (\mathcal{S}_2 \cup \mathcal{M}_2)] \cup [\mathcal{S}_2 \sqcap (\mathcal{S}_1 \cup \mathcal{M}_1)] \rangle \mid \langle I, \mathcal{M}_1, \mathcal{K}_1, \mathcal{S}_1 \rangle \in \tau_1, \langle I, \mathcal{M}_2, \mathcal{K}_2, \mathcal{S}_2 \rangle \in \tau_2,$

14 $\quad\quad |\mathcal{S}_1| = |\mathcal{S}_1 \sqcap (\mathcal{S}_2 \cup \mathcal{M}_2)|, |\mathcal{S}_2| = |\mathcal{S}_2 \sqcap (\mathcal{S}_1 \cup \mathcal{M}_1)|\}$

$S_e^+ := S \cup \{e\}$, $S_e^- := S \setminus \{e, \neg e\}$, $\mathcal{S}_e^\sim := \{\langle M_e^-, \{C_e^- \mid C \in \mathcal{C}\}\rangle \mid \langle M, \mathcal{C} \rangle \in S\}$, $\mathcal{M}_1 \sqcap \mathcal{M}_2 := \{\langle M, (\mathcal{C}_M^+ \cap \mathcal{D}) \cup (\mathcal{C} \cap \mathcal{D}_M^+) \rangle \mid \langle M, \mathcal{C} \rangle \in \mathcal{M}_1, \langle M, \mathcal{D} \rangle \in \mathcal{M}_2\}$.

Algorithm PRIM is designed to ensure existence of such a proving primal row in $\tau(n)$ of root node n of the TD, iff a WV exists. PRIM uses the following constructs, assuming an answer set tuple $\langle M, \mathcal{C} \rangle$, an atom $a \in \mathcal{A}$, and a program $\mathscr{P}$. For updating an answer set tuple, we let $\mathrm{updT}(M, \mathcal{C}, \mathscr{P}) = \{\langle M, \mathcal{C} \cap \mathit{mods}(\mathscr{P}^M) \rangle \mid M \vDash \mathscr{P}\}$. When some atom a is introduced in an intr-type node, we need to distinguish between a already being in the interpretation, or not. We define $\mathrm{intT}(a, M, \mathcal{C}, \mathscr{P}) = \mathrm{updT}(M_a^+, \bigcup_{C \in \mathcal{C}} \{M, C, C_a^+\}, \mathscr{P}) \cup \mathrm{updT}(M, \mathcal{C}, \mathscr{P})$, which is generalized to sets $\mathcal{M}$ of answer set tuples as follows: $\mathrm{intTs}(a, \mathcal{M}, \mathscr{P}) = \bigcup_{\langle M, \mathcal{C} \rangle \in \mathcal{M}} \mathrm{intT}(a, M, \mathcal{C}, \mathscr{P})$. Finally, to obtain good runtime bounds later and at the same time still ensure Condition (4) of Definition 2 using a set $\mathcal{S}$ of answer set tuples, we need to find, for each answer set tuple in $\mathcal{S}$, exactly one "succeeding" answer set tuple among the set $\mathcal{M}$ of answer set tuples. We formalize this by defining $\mathrm{succS}(a, \mathcal{M}, \mathcal{C}, \mathscr{P}) = \{\mathcal{S}' \mid \mathcal{S}' \subseteq \mathcal{M}, |\mathcal{S}'| = |\mathcal{S}|,$ for every $\langle M, \mathcal{C} \rangle \in \mathcal{S} : \mathrm{intT}(a, M, \mathcal{C}, \mathscr{P}) \cap \mathcal{S}' \neq \emptyset\}$.

Bag algorithm PRIM, as presented in Listing 2, again distinguishes between different types of tree nodes during the post-order traversal of $\mathcal{T}$. For leafs, Line 1 returns the primal row consisting of the empty CWI, where the second component $\mathcal{M}$ contains only the proving answer set tuple $\langle \emptyset, \emptyset \rangle$ (since $\emptyset$ is the smallest model of the empty program), and $\mathcal{K}, \mathcal{S}$ are both empty as there is no need to remove or create answer set tuples, respectively. If an atom $a \notin \mathit{atel}(\Pi)$ is introduced, Line 3 updates $\mathcal{M}$ and $\mathcal{K}$. Line 4 ensures that each answer set tuple in $\mathcal{S}$ has at least one succeeding answer set tuple in every primal row of table $\tau(t)$. If an atom $a \in \mathit{atel}(\Pi)$ is introduced, the sets $\mathcal{M}$, $\mathcal{K}$, and $\mathcal{S}$ are similarly updated in Line 6, but the three cases (true, false, and unknown) need to be considered when adding a to I. Conditions (2) and (3) are handled in Line 8, where answer set tuples in $\mathcal{M}$ that violate these two conditions (for $a \in J^P$, and $a \in J^N$, respectively) are added to $\mathcal{K}$. For Condition (4), Line 9 ensures that if $a \in J^N$, there is both a succeeding answer set tuple where a is set to true, and one where it is false. If an atom a is removed, a is removed from the primal rows in Line 11, since we have processed every part of Π where a occurs. Finally, for join nodes, we combine only "compatible" primal rows in Line 13. In particular, Line 14 ensures that no answer set tuple is lost in $\mathcal{S}_1$ or $\mathcal{S}_2$ of the child primal rows.

In the following, we briefly mention correctness and runtime bounds for bag algorithms PRIM and EPRIM.

Theorem 12 (Correctness)**.** *Let* Π *be an ELP* Π*, and* $\mathcal{T} = (T, \chi)$ *a TD of* $\mathbb{P}_\Pi$*. Then there is a proving row in table* $\tau(n)$ *obtained by* $\mathsf{DP}_{\mathsf{PRIM}}$ *for root* n *of* $\mathcal{T}$ *iff there is a WV for* Π*.*

Then, correctness of $\mathsf{DP}_{\mathsf{EPRIM}}$, cf., Sec. 4, is a special case.

Corollary 13 (Correctness of $\mathsf{DP}_{\mathsf{EPRIM}}$)**.** *Let* Π *be an ELP* Π *and* $\mathcal{T} = (T, \chi)$ *a TD of* $\mathbb{E}_\Pi$*. Then there is a row in table* $\tau(n)$ *for root* n *obtained by* $\mathsf{DP}_{\mathsf{EPRIM}}$ *iff there is a WV for* Π*.*

Proof (Idea). $\mathcal{T}$ can be turned into a TD $\mathcal{T}'$ of $\mathbb{P}_\Pi$ by adding to each $\chi(t)$ the set $\mathcal{A}$, where $(\Pi_t^{\mathbb{E}}) = (\mathcal{A}, \cdot)$. $\mathsf{DP}_{\mathsf{EPRIM}}$ on $\mathcal{T}$ is, therefore, just a simplification of $\mathsf{DP}_{\mathsf{PRIM}}$ on $\mathcal{T}'$. □

Theorem 14 (Runtime)**.** $\mathsf{DP}_{\mathsf{PRIM}}$ *runs in time* $2^{2^{2^{\mathcal{O}(k)}}} \cdot |\mathcal{A}|$ *for epistemic program* $\Pi = (\mathcal{A}, \mathcal{R})$*, and treewidth* k *of* $\mathbb{P}_\Pi$*.*

Indeed, under reasonable assumptions in computational complexity, that is, the *exponential time hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), one cannot significantly improve $\mathsf{DP}_{\mathsf{PRIM}}$ since $\mathsf{DP}_{\mathsf{PRIM}}$ is ETH-tight.

Proposition 15 (cf. (Fichte, Hecher, and Pfandler 2019), Theorem 19)**.** *Let* $\Pi = (\mathcal{A}, \cdot)$ *be an epistemic logic program with* $\mathbb{P}_\Pi$ *of treewidth* k*. Then, unless ETH fails, WV existence for* Π *cannot be decided in time* $2^{2^{2^{o(k)}}} \cdot 2^{o(|\mathcal{A}|)}$*.*

6 Conclusions

This work provides the first parameterized complexity analysis of ELP solving w.r.t. treewidth. Tree decompositions (TDs) have been successfully used in the *selp* ELP solver (Bichler, Morak, and Woltran 2018), but for a different purpose, namely that ELPs are rewritten into non-ground ASP programs with long rules, which are then split up using *rule decomposition* (Bichler, Morak, and Woltran 2016). Our approach partitions an ELP according to a TD, and then solves the entire ELP by evaluating these parts in turn. Note that this is different from (ELP) splitting (Cabalar, Fandinno, and Fariñas del Cerro 2019; Lifschitz and Turner 1994).

For future work, we aim to extend our DP algorithm to the formula evaluation problem, which, viz. Theorem 6, should work in a similar fashion to our existing algorithms, given a suitable graph representation. Furthermore, we would like to apply our approach to other ELP semantics; cf. (Gelfond 1991; 2011; Kahl et al. 2015). There, we do not anticipate large obstacles, since most semantics are reduct-based, and the reduct is an easily exchangeable part in our algorithms.

Acknowledgements

M. Hecher and S. Woltran were supported by the Austrian Science Fund (FWF) under grants Y698 and P32830.

References

Bichler, M.; Morak, M.; and Woltran, S. 2016. The power of non-ground rules in answer set programming. *TPLP* 16(5-6):552–569.

Bichler, M.; Morak, M.; and Woltran, S. 2018. Single-shot epistemic logic program solving. In *Proc. IJCAI*, 1714–1720.

Bliem, B.; Moldovan, M.; Morak, M.; and Woltran, S. 2017. The impact of treewidth on ASP grounding and solving. In *Proc. IJCAI*, 852–858.

Bliem, B.; Ordyniak, S.; and Woltran, S. 2016. Clique-width and directed width measures for answer-set programming. In *Proc. ECAI*, 1105–1113.

Bodlaender, H. L., and Kloks, T. 1996. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms* 21(2):358–402.

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

Cabalar, P.; Fandinno, J.; and Fariñas del Cerro, L. 2019. Splitting epistemic logic programs. In *Proc. LPNMR*, 120–133.

Charwat, G., and Woltran, S. 2019. Expansion-based QBF solving on tree decompositions. *Fundam. Inform.* 167(1-2):59–92.

Courcelle, B. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.* 85(1):12–75.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. Monographs in Computer Science. Springer.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3-4):289–323.

Fariñas del Cerro, L.; Herzig, A.; and Su, E. I. 2015. Epistemic equilibrium logic. In *Proc. IJCAI*, 2964–2970.

Fichte, J. K., and Hecher, M. 2019. Treewidth and counting projected answer sets. In *Proc. LPNMR*, 105–119.

Fichte, J. K., and Szeider, S. 2015. Backdoors to tractable answer set programming. *Artif. Intell.* 220:64–103.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2017. Answer set solving with bounded treewidth revisited. In *Proc. LPNMR*, 132–145.

Fichte, J. K.; Hecher, M.; and Pfandler, A. 2019. TE-ETH: Lower Bounds for QBFs of Bounded Treewidth. Preliminary version available at https://tinyurl.com/y7wnvu6w.

Fichte, J. K.; Hecher, M.; and Zisser, M. 2019. An improved GPU-based SAT model counter. In *Proc. CP*, 491–509.

Fichte, J. K.; Kronegger, M.; and Woltran, S. 2019. A multiparametric view on answer set programming. *Ann. Math. Artif. Intell.* 86(1-3):121–147.

Ganian, R.; Ramanujan, M. S.; and Szeider, S. 2017. Combining treewidth and backdoors for CSP. In *Proc. STACS*, 36:1–36:17.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. ICLP/SLP*, 1070–1080. MIT Press.

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

Gelfond, M., and Przymusinska, H. 1991. Definitions in epistemic specifications. In *Proc. LPNMR*, 245–259.

Gelfond, M. 1991. Strong introspection. In *Proc. AAAI*, 386–391. AAAI Press / The MIT Press.

Gelfond, M. 2011. New semantics for epistemic specifications. In *Proc. LPNMR*, 260–265.

Gottlob, G.; Pichler, R.; and Wei, F. 2010. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.* 174(1):105–132.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.* 63(4):512–530.

Jakl, M.; Pichler, R.; and Woltran, S. 2009. Answer-set programming with bounded treewidth. In *Proc. IJCAI*, 816–822.

Kahl, P. T.; Watson, R.; Balai, E.; Gelfond, M.; and Zhang, Y. 2015. The language of epistemic specifications (refined) including a prototype solver. *J. Log. Comput.* 25.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Proc. ICLP*, 23–37.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Ann. Math. Artif. Intell.* 25(3-4):369–389.

Lonc, Z., and Truszczynski, M. 2003. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Log.* 4(1):91–119.

Morak, M. 2019. Epistemic logic programs: A different world view. In *Proc. ICLP*, 52–64.

Pearce, D.; Tompits, H.; and Woltran, S. 2009. Characterising equilibrium logic and nested logic programs: Reductions and complexity. *TPLP* 9(5):565–616.

Robertson, N., and Seymour, P. D. 1984. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B* 36(1):49–64.

Schaub, T., and Woltran, S. 2018. Special issue on answer set programming. *KI* 32(2-3).

Shen, Y., and Eiter, T. 2016. Evaluating epistemic negation in answer set programming. *Artif. Intell.* 237:115–135.

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*.