Informatics

Evaluation of Epistemic Logic Programs Based on External Atoms

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Anton Strasser, BSc
Matrikelnummer 00527527

an der Fakultät für Informatik der
Technischen Universität Wien

Betreuung: Prof. Dr. Thomas Eiter
Mitwirkung: Dr. Christoph Redl

Wien, 2022-09-16

Anton Strasser	Thomas Eiter

TU WIEN Informatics

Evaluation of Epistemic Logic Programs Based on External Atoms

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Anton Strasser, BSc
Registration Number 00527527

to the Faculty of Informatics
at the TU Wien

Advisor: Prof. Dr. Thomas Eiter
Assistance: Dr. Christoph Redl

Vienna, 2022-09-16 ....
Anton Strasser Thomas Eiter

Zusammenfassung

Answer Set Programming (ASP) mit epistemischen Spezifikationen erweitert die Sprache der disjunktiven Logikprogramme (DLP) um die epistemischen Modaloperatoren K und M, die ausdrücken, dass eine Formel in allen Answer Sets bzw. in mindestens einem Answer Set wahr ist. Da die Erfüllbarkeit unter Modaloperatoren ein Konzept ist, das über eine Sammlung von Answer Sets, die *World View* genannt wird, definiert ist, kann die Berechnung der Answer Sets eines Programms mit epistemischen Spezifikationen nicht direkt auf die gleiche Aufgabe reduziert werden wie für ein gewöhnliches DLP-Programm. Stattdessen generieren aktuelle Algorithmen eine Obermenge von Answer Sets, indem sie ein spezielles Programm auswerten, um mögliche Lösungen zu generieren und falsche Lösungen in einer Nachprüfung auszusortieren. In dieser Arbeit wird ein neuartiger Ansatz zur Auswertung epistemischer Logikprogramme (ELP) vorgestellt, der auf HEX-Programmen basiert. Letztere sind eine weitere Erweiterung von DLP in Richtung externer Rechenquellen. Konkret basiert unser Algorithmus zur Auswertung von ELP-Programmen auf HEX-Programmen, die deklarative Prüfregeln mit externen Atomen enthalten. Darüber hinaus stellen wir eine Implementierung des Algorithmus und seine experimentelle Auswertung vor, die deutliche Verbesserungen gegenüber einer Referenzimplementierung zeigt.

Abstract

Answer set programming (ASP) with epistemic specifications extends the language of disjunctive logic programs (DLP) to include the epistemic modal operators K and M, which express that a formula is true in all answer sets and at least one answer set, respectively. Since satisfiability under modal operators is a concept defined over a collection of answer sets called *world view*, computing the answer sets of a program with epistemic specifications cannot be directly reduced to the same task as for an ordinary DLP program. Instead, current algorithms generate a superset of answer sets by evaluating a special program to generate possible solutions and sorting out incorrect solutions in a post-check. In this thesis, we present a novel approach to evaluating epistemic logic programs (ELP) based on HEX programs. The latter are a further extension of DLP towards external computational sources. Specifically, our algorithm for evaluating ELP programs is based on HEX programs that contain declarative checking rules with external atoms. Moreover, we present an implementation of the algorithm and its experimental evaluation, which shows significant improvements over a reference implementation.

Contents

Introduction

1

In this work, we evaluate *epistemic logic programs* that extend ordinary logic programs with modal operators as in modal logic. Epistemology is the branch of philosophy concerned with the nature of knowledge and belief. Modal logic extends classical logic with modal operators to qualify statements with *modalities*, e.g., the traditional modalities of truth include *possibility* ("It is possible that φ") and *necessity* ("It is necessary that φ"). The modalities in *epistemic* modal logic capture the semantics of knowledge (Hintikka, 1962), e.g., "It is known that φ". Epistemic logic programs – which extend ordinary logic programs with epistemic modal operators – can thus be seen as an application of epistemic modal logic in answer set programming.

1.1 Motivation and Problem Statement

Answer set programming (ASP) is a well-established logic programming paradigm in the style of declarative programming. It has its roots in logic programming and nonmonotonic reasoning (Gelfond and Leone, 2002; Baral, 2003; Eiter et al., 2009). In ASP, the semantics of a logic program is defined over its stable models (answer sets) (Gelfond and Lifschitz, 1991). Search problems are reduced to computing answer sets for logic programs. An ASP solver performs the search, which generates answer sets that represent solutions.

A logic program consists of a set of rules that are expressed in a formal language. Such languages differ in expressivity and reasoning complexity. In this work we evaluate *epistemic logic programs* expressed in the ELP language. This language extends the language of disjunctive logic programs (DLP) (Gelfond and Lifschitz, 1991) with epistemic modal operators K and M to express that a formula is true in all or at least one answer set, respectively (Gelfond, 1991).

1.1.1 *A Declarative Checking Approach*

As satisfiability of a program under modal operators is a concept defined over a collection of answer sets, called *world view*, computing a world view of an ELP program is not directly reducible to computing answer sets of an ordinary DLP program. Instead, current algorithms transform an ELP program into a DLP program whose answer sets represent candidate solutions that are checked in a second computation step. The language of higher-order logic programs with external atoms (HEX) is another extension of the DLP language, where the evaluation of external atoms is delegated to an external computation resource (Eiter et al., 2005). This motivates a novel approach in which the checking part is also expressed in a declarative way, using HEX programs containing external atoms. The answer sets of these programs, which represent world views, are computed with a HEX solver, in particular the *dlvhex* system (Eiter et al., 2015).

1.2 State of the Art

In the following we give a state-of-the-art overview of ELP semantics and ELP solvers and then state our contributions. We assume that the reader is familiar with the basic concepts of answer set programming and the usual notation.[1]

Programs with epistemic specifications were first introduced in *Deep Introspection* by Gelfond (1991). In this article, he introduces the epistemic modal operator K, which allows to express that something is *known* to a reasoner. The dual modal operator M, which expresses that something *may be believed*, is then defined in terms of the K operator, i.e., given a formula φ, he defines $M\,\varphi$ as semantically equivalent to $\neg K \neg \varphi$.[2]

Example 1. (Edible fruits) To give an example using the modal operator K, the rule

$$\neg\mathrm{edible}(X) \leftarrow \mathit{not}\, K\, \mathrm{edible}(X), \mathrm{fruit}(X),$$

expresses the closed world assumption[3] (CWA) with regards to the edibility of fruits. Intuitively, this rule determines that a fruit is not edible if we do not know if it is edible.

Epistemic specifications play an important role in knowledge representation and nonmonotonic reasoning. The initial approach (G91 semantics, Gelfond (1991)) suffers from the problem of so-called *unintended world views* due to recursion trough the modal operators K and M. Other attempts to avoid unintended world views, such as the G94 semantics (Gelfond, 1994), the G11 semantics (Gelfond, 2011), and the K14 semantics (Kahl, 2014), were still not satisfactory in this regard. To illustrative unintended world views, we borrow the following example from (Kahl, 2014) .

Example 2 (Unintended World Views)**.** Under G91 semantics, unintended world views may occur due to recursion through the modal operator M. Consider the program $\Pi_1 = \{a \leftarrow M\,a\}$ containing some atom a, which states that a is true provided a is possible. Under G11 semantics, this program has the two world views $\mathcal{A}_1 = \{\{a\}\}$ and $\mathcal{A}_2 = \{\varnothing\}$. While an agent may accept the one or the other collection

[1] For an introduction, see, e.g., (Eiter et al., 2009).

[2] In later semantics (Gelfond, 2011), $M\,\varphi$ is defined as $\neg K\, \mathit{not}\, \varphi$.

[3] We informally state Reiter's (1978) closed world assumption as "if no proof of a positive ground literal exists, then the negation of that literal is assumed true".

as a world view of Π_1, the agent should not accept both collections as world view. Intuitively, the agent can see the program Π_1 as in Murphy's Law "Anything that can go wrong will go wrong" and go for $\mathcal{A}_1$. Or he sees the program as a tautology and accept $\mathcal{A}_2$. But if Π_1 is seen as a tautology, then it seems not reasonable to see it as Murphy's Law at the same time. In contrast, the program $\{a \leftarrow K\,a\}$ is clearly a tautology.

A satisfactory solution to the problem of unintended world views was proposed in the seminal article *Evaluating epistemic negation in answer set programming* by Shen and Eiter (2016). This paper also adopts a completely different modal operator **not** in the style of negation operators to expresses *epistemic negation*. An epistemic negation **not** φ expresses the lack of evidence *proving* that φ is true, e.g., the rule of Example 1 would be written as $\neg\text{edible}(X) \leftarrow$ **not** $\text{edible}(X), \text{fruit}(X)$.

1.2.1 *Program Transformations and Semantics*

Logic programs are often evaluated based on an *assumption* that leads to a simpler program that is free of negation and modal literals. Solutions of the simpler program then represent solutions of the initial program if they are consistent with the assumption. This program transformation is commonly referred to as the *reduct* of a program relative to an assumption. For instance, the FLP reduct $f\Pi^M$ (Faber et al., 2010) of a program Π relative to an interpretation M consists of the rules of Π whose body is satisfied by M. The assumption here is that M is a subset minimal model of $f\Pi^M$, and if the assumption holds, then M is an answer set of Π.

In a similar way, Gelfond (1991, 2011), Truszczynski (2011) and Kahl (2014) defined different reducts of epistemic logic program Π relative to a collection $\mathcal{A}$ of interpretations. These reducts are obtained from Π by removing or replacing modal literals in Π depending on their truth in $\mathcal{A}$.

We take a closer look at Kahl's *modal reduct* $\Pi^{\mathcal{A}}$. In particular, for modal literals of the form $M\,a$ that are false in $\mathcal{A}$ there is a rule to replace them with doubly negated literals of the form *not not a* in the reduct. Then, to test whether $\mathcal{A}$ is a world view of Π, the modal reduct $\Pi^{\mathcal{A}}$ is evaluated under the answer set semantics defined in Lifschitz et al. (1999), under which double negation does not cancel each other out. Recall that the program $\Pi_1 = \{a \leftarrow M\,a\}$ from Example 2 has two conflicting world views $\{\{a\}\}$ and $\{\varnothing\}$ under G11 semantics. Under Kahl's new semantics, the collection $\mathcal{A} = \{\varnothing\}$ is no longer a world view of Π_1, since $\mathcal{A}$ is not the collection $\{\{a\}, \varnothing\}$ of *all* answer sets of the modal reduct $\Pi_1^{\mathcal{A}} = \{a \leftarrow \textit{not not } a\}$ of Π_1.[4] However, three shortcomings of the K14 semantics were later identified in the article *Evaluating epistemic negation in answer set programming* by Shen and Eiter (2016):

1. Lack of a deeper discussion or justification for the replacements of modal

[4] Note that, e.g., under FLP answer set semantics, double negation cancels each other out resulting in the singleton answer set $\varnothing$ of $\Pi_1^{\mathcal{A}}$.

literals in program transformations

2. The choice of a semantics for nested expressions with double nonmonotonic negation that suffers from circular justifications
3. For some programs, unintended world views due to recursion through the modal operator M may still occur

In this article, Shen and Eiter introduce the *general epistemic answer set semantics* (SE16$_{\mathcal{X}}$semantics) for general logic programs. The new semantics is defined via the so-called *epistemic reduct* under $\mathcal{X}$ answer set semantics. By comparison with previous reducts, the epistemic reduct is obtained relative to an *epistemic guess* rather than a collection of interpretations. The basic idea is to assume an epistemic negation **not** φ in a program to be true whenever possible, which is referred to as *knowledge minimization with epistemic negation* in the article. Put roughly, an epistemic guess of an epistemic logic program Π is defined as a subset of the epistemic negations obtained from Π. Any epistemic guess that leads to a world view is by definition i) *consistent* with the world view and ii) *maximal* with respect to subset inclusion.

We can apply the new semantics to programs expressed in the traditional syntax, since we can convert all expressions that use the modal operators K and M into equivalent expressions that use the epistemic negation operator **not**. In this thesis, we will employ a variant of the traditional syntax and define the epistemic reduct accordingly.

1.2.2 *Algorithms and Solvers*

To date, a number of solvers for epistemic logic programs (ELP solvers) have been developed (Leclerc and Kahl, 2018). The solvers shown in Table 1.1 have in common that they offload search tasks to an ASP solver by generating logic programs that simulate some sort of reduct. But they implement different semantics and prune the search space in different ways.

Solver	Semantics	ASP Backend	Reference
Wviews	G94	*dlv*	Kelly (2007)
ESmodels	G11	*clasp*	Zhang et al. (2013)
ELPS	K14	*clingo*	Balai and Kahl (2014)
ELPsolve	K14, K16	*clingo*	Kahl et al. (2016)
EP-ASP	K14, K16	*clingo*	Son et al. (2017)
selp	SE16$_{\text{FLP}}$	*clingo*	Bichler et al. (2018)

Table 1.1: Overview of ELP solvers and supported semantics.

The first general ELP solver *Wviews* (Kelly, 2007) implements the G94 semantics. The underlying algorithm tries one epistemic guess (truth assignment of modal literals) at a time and uses the ASP solver *dlv* (Citrigno et al., 2004). Since the number of guesses to try is exponential in the number of modal literals that can be guessed and no pruning of the search space is involved, larger problems are not feasible with this approach.

The ELP solver *ESmodels* (Zhang et al., 2013) implements the G11 semantics, which prevents unintended world views from recursion through the modal operator K and uses an algorithm that calls a preprocessing routine for each guess before handing the problem over to the ASP solver *clasp*. In (Leclerc and Kahl, 2018) it was reported that *ESmodels* works reasonably well for a small number of modal literals, but that runtime errors or incomplete results sometimes occurred for larger problems.

In his PhD thesis, Kahl (2014) proposes a refined semantics (K14 semantics) that addresses unintended world views in the G11 semantics through recursion through the modal operator M. The ELP solver *ELPS* (Balai and Kahl, 2014) is the first solver to implement an algorithm (Kahl et al., 2015) with K14 semantics. In contrast to previous solver implementations, *ELPS* creates all candidate solutions to the search problem in a single call to the ASP solver *clingo*. In the worst case however, the solver generates a number of solutions that is exponential in the number of modal literals that need to be guessed, which may require more memory than is available to the solver process.

For the algorithm implemented by the ELP solver *ELPsolve*, Kahl et al. (2016) adopted the new definition of world views from Shen and Eiter (2016), with *ELPsolve* still supporting K14 semantics. We refer to the resulting semantics, which imposes maximality requirements on epistemic guesses, as the K16 semantics. As stated in (Leclerc and Kahl, 2018), the primary efficiency goals of *ELPsolve* were i) to avoid the large memory requirements of *ELPS*, and ii) to take advantage of multi-core processors. To this end, the algorithm splits the search space by guesses, i.e., guesses are grouped by their cardinality and those groups are split into smaller groups that contain at most n guesses, where n is the number of guesses per ASP solver call. Because any two groups containing guesses of the same cardinality are pairwise disjoint, *ELPsolve* is able to execute ASP solver calls in parallel. Grouping guesses by their cardinality also allows for pruning of guesses that do not meet the maximality requirements of the K16 semantics. The answer sets of the individual ASP solver calls represent candidate solutions that are checked by *ELPsolve* for consistency with their respective guesses.

EP-ASP (Son et al., 2017) is another ELP solver that supports both the K14 and the K16 semantics but uses a different algorithm. Instead of generating answer sets of candidate solutions for multiple guesses at once, *EP-ASP* uses the ASP solver *clingo* with a special transformation Γ of the input program to iteratively select guesses such that the epistemic reduct with respect to this guess is consistent. Each guess is tested to see whether it leads to a world view; only after the test passes, the solver generates the answer sets of the corresponding epistemic reduct. After this test, the program Γ is extend with constraints such that the guess is excluded from selection in subsequent iterations. The maximality condition of the K16 semantics is implemented by additionally requiring that it is impossible to select a superset of the selected guess relative to the program Γ. To our knowledge, this is the first ELP solver to employ brave and cautious reasoning for pruning the search space.

In (Bichler et al., 2018) the authors show that the consistency problem for an ELP program Π is reducible to the consistency problem of an ordinary ASP program Π' in the sense that Π has a candidate world view[5] exactly when Π' is consistent. They refer to this method in their paper as single-shot ELP solving because it requires only a single ASP solver call. The associated *selp*[6] system implements this reduction, which is a collection of tools that can be used to ground and transform an ELP program, and to group the answer sets obtained from the ASP solver call into candidate world views. The *selp* system works with $\text{SE16}_{\mathcal{X}}$semantics, where $\mathcal{X}$ is the FLP answer set semantics.

1.2.3 *Our Contributions*

In this thesis, we present a novel algorithm based on higher-order programs with external atoms (HEX programs). A HEX program uses so-called *external atoms* to access an external computation resource, e.g., an external database interface. But also another instance of a HEX solver can be an external computation resource. With this it becomes possible to reason over the answer sets of a subprogram within the main program through external atoms. Such a program that evaluates a subprogram through external atoms is called a *nested HEX program* (Eiter et al., 2013). With nested HEX programs we can check truth assignments of epistemic formulas in a declarative way, by reasoning over the answer sets of a specific subprogram through external atoms. Our algorithm transforms an ELP program into a collection of intermediate HEX programs containing external atoms. The answer sets of these programs then encode world views of the input program.

In the following list we summarize our contributions:

1. A syntax that is suitable for defining the language of epistemic logic programs as an extension of the language of disjunctive logic programs. In this syntax, we prefer the traditional modal operators K and M over the epistemic negation operator **not** to qualify atoms or negated atoms with epistemic modalities.
2. The notion of weak and strong modal literals and related notions. In our syntax, an expression that starts with the modal operator M is a weak modal literal, and an expression that starts with the modal operator K is a strong modal literal.
3. An algorithm that outputs solutions to the world view enumeration problem as it finds them.
4. Optimizations of the algorithm with the aim of reducing the search space.

[5]As defined in (Shen and Eiter, 2016).
[6]`https://dbai.tuwien.ac.at/proj/selp/`

5. An implementation of the algorithm in form of a configurable prototype solver for ELP programs.
6. An experimental evaluation of the algorithm using our prototype solver with different configurations compared to the reference solver *ELPsolve*.

1.3 Structure of this Thesis

This thesis consists of six chapters. The first two chapters are the introduction and the preliminaries. The main chapters cover the introduction, the implementation, and the evaluation of the EHEX algorithm. The final chapter is the conclusion.

In Chapter 2, we introduce the syntax and the semantics of epistemic logic programs and define when a program is satisfied. For this we also introduce the notion of *weak* and *strong modal literals*. The *epistemic answer set semantics* of ELP programs is then defined in terms of the answer sets of the *epistemic reduct* of the program relative to a set of modal literals.

In Chapter 3, we present the EHEX algorithm. This algorithm takes ELP programs as input and outputs their world views, in case some world view exists. It is based on *HEX programs* that use *brave* and *cautious external atoms*. These programs are composed out of smaller programs, which we call *building blocks*. We also show that the algorithm is correct and that it can be optimized.

In Chapter 4, we present a prototype implementation of the EHEX algorithm. The resulting solver is called *ehex*. Under the hood, *ehex* uses the *dlvhex* system, which is also described in this chapter.

In Chapter 5 we present results from the experimental evaluation of the *ehex* solver. The running times are compared with the running times of the reference solver *ELPsolve* on a set of standard problems.

Finally, in Chapter 6, we summarize our work and discuss how different epistemic semantics emerge from different program transformations. This chapter concludes with an outlook and open issues.

Preliminaries

2

In the following, we give an intuition for modal operators and define the language of epistemic logic programs (ELP) and its world view semantics.

2.1 Epistemic Logic Programs

An epistemic logic program is a set of rules that allow to express problems including statements that are qualified with epistemic modalities. The *modal operators* K and M are commonly used to qualify expressions with "it is known that" or "it may be known that", respectively, e.g., (Gelfond, 1991; Kahl, 2014). But recently, an alternative operator, the *epistemic negation operator* **not**, has been introduced to qualify statements with "there is no evidence proving that" (Shen and Eiter, 2016).

To avoid confusion with the default negation operator *not* and for convenience, from now on we denote the epistemic negation operator by the capital letter N and use the modal operators M and K as shorthand for semantically equivalent expressions with the epistemic negation operator, where

$$\begin{aligned}&M\,\varphi \text{ is equivalent to } N\,\mathit{not}\,\varphi\text{, and}\\ &K\,\varphi \text{ is equivalent to } \mathit{not}\,N\,\varphi.\end{aligned}$$

Also, we abstain from using negation in front of the modal operators K and M to give greater emphasis to the modalities they express.

2.1.1 *Syntax*

For simplicity but without loss of generality, our input language ELP is based on a subset of the ASP-Core syntax (Calimeri et al., 2012). In particular our syntax does not include classic negation, aggregates, or built-in atoms.

By convention, a *constant* is either a string starting with a lowercase letter, a quoted string, or an integer; a *variable* is a string starting with an uppercase letter. A *term* is either a constant, a variable, or a functional term. Given a function name f and a list of terms $t_1, \ldots, t_n$, the expression $f(t_1, \ldots, t_n)$ denotes a *functional term* if $n > 0$ or a constant term if $n = 0$.

An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ with predicate name p and terms $t_1, \ldots, t_n$, where $n \geq 0$ is the arity of the atom. A *standard literal* is an atom or an atom preceded by the default negation operator *not*. The expressions $K\,\ell$ and $M\,\ell$ denote *modal literals*, where ℓ is a standard literal and K and M are modal operators. A *literal* is either a standard literal or a modal literal.

The standard literals a and $not\,a$ over an atom a are are said to be the *opposite* of each other. Modal literals also have opposites, as shown in Table 2.1. For a literal φ, we denotes its opposite by $\overline{\varphi}$.

Literal φ	Opposite $\overline{\varphi}$
a	$not\,a$
$not\,a$	a
$K\,a$	$M\,not\,a$
$K\,not\,a$	$M\,a$
$M\,a$	$K\,not\,a$
$M\,not\,a$	$K\,a$

Table 2.1: Opposites of each other cannot both be true or both be false at the same time

Definition 1 (ELP programs)**.** A *disjunctive epistemic logic program* (ELP program for short) is a set of rules of the form

$$a_1 \vee \ldots \vee a_m \leftarrow b_1, \ldots, b_n, \tag{2.1}$$

where $a_1, \ldots, a_m$ are atoms, $b_1, \ldots b_m$ are standard literals or modal literals, and $m \geq 0, n \geq 0, m + n > 0$.

For a rule r of the form (2.1), the *head* of r is $\text{Head}(r) := \{a_1, \ldots, a_m\}$, and the *body* of r is $\text{Body}(r) := \{b_1, \ldots, b_n\}$.

A structure (term, atom, literal, rule or program) is called *ground* if it contains no variables. A *fact* is a ground rule of the form $a \leftarrow$ (or a for short) with an empty body and a single disjunct in the head, and a *constraint* is a rule of the form $\leftarrow b_1, \ldots, b_n$ with an empty head.

An ELP program without modal literals is a *disjunctive logic program* (DLP program for short). Unless otherwise stated, a *program* is an ELP program.

Remark 1 (Strong negation)*.* The strong (classical) negation $\neg$ of atoms can be seen as a special syntax to make programs easier to read. A program containing strongly negated atoms can be rewritten into an equivalent program without strong negation:

1. Each strongly negated atom $\neg a$ is viewed as an atom with the fresh predicate symbol $\neg a$
2. For each $\neg a$ the constraint $\leftarrow a, \neg a$ is added to prevent that a and $\neg a$ are both true in the same model

To give an intuition, a rule of the form $\psi \leftarrow \varphi$ can be read as an if-then statements in the type of "if φ holds then infer ψ" or as the material implication $\varphi \supset \psi$. However, in general, rules in logic programs have a different meaning. For example, in classical logic, the formula $\varphi \supset \psi$ is logically equivalent to its contrapositive $\neg\psi \supset \neg\varphi$. In some applications, this expectation may be translated into a logic program by stating both directions $\psi \leftarrow \varphi$ and $\neg\varphi \leftarrow \neg\psi$ explicitly. Nevertheless, a formula like $\neg\varphi \supset \varphi$ is equivalent to φ in classical logic, but the rule $\varphi \leftarrow \neg\varphi$ is not equivalent to the fact $\varphi \leftarrow$ in a logic program (Shen et al., 2014).

2.1.2 *Satisfaction and Models*

The truth value of a ground atom is relative to an *interpretation*, which is a set of ground atoms. A ground atom is true if it is contained in the interpretation and false otherwise.

For an ELP program Π, the *Herbrand universe* of Π, in symbols U_Π, is the set of all ground terms of Π. In case that no constant appears in Π, an arbitrary new constant is added to U_Π. The *Herbrand base* of Π, in symbols B_Π, is the set of ground atoms $p(t_1, \dots, t_n)$, where p occurs in Π and each t_1 is in U_Π. A *Herbrand interpretation* I of Π is any subset I of the Herbrand base B_Π. Unless otherwise stated, an interpretation of a program is a Herbrand interpretation.

A *variable substitution* is a mapping from a set V of variables to the Herbrand universe U_Π of a program Π. A *ground instance* of a rule r of the form (2.1) is any rule r' obtained from r by applying a variable substitution to the variables in r. For any rule r, we denote by $\mathrm{ground}(r)$ the set of all possible ground instances of r, and for any program Π we let

$$\mathrm{ground}(\Pi) \coloneqq \bigcup_{r \in \Pi} \mathrm{ground}(r) \tag{2.2}$$

be the *grounding* of Π.

Definition 2 (Satisfaction of standard literals)**.** A ground atom a is true under an interpretation I, denoted $I \models a$, if $a \in I$; otherwise a is false under I. A default negated ground atom *not* a is true under an interpretation I, in symbols $I \models \textit{not}\, a$, if $a \notin I$; otherwise *not* a is false under I, in symbols $I \not\models \textit{not}\, a$.

By contrast, the truth value of modal literals depends on a nonempty collection of interpretations.

Definition 3 (Satisfaction of modal literals)**.** Let $\mathcal{A}$ be a nonempty collection of interpretations and let ℓ be a ground standard literal. Then

1. $K\,\ell$ is true in $\mathcal{A}$, if $I \models \ell$ for all $I \in \mathcal{A}$,
2. $M\,\ell$ is true in $\mathcal{A}$, if $I \models \ell$ for some $I \in \mathcal{A}$.

A modal literal φ is satisfied by $\mathcal{A}$, in symbols $\mathcal{A} \models \varphi$, if φ is true in $\mathcal{A}$.

Intuitively, a strong modal literal $K\,\ell$ is true if all interpretations satisfy ℓ, and a weak modal literal $M\,\ell$ is true if some interpretation satisfies ℓ. Checking the satisfaction of weak and strong modal literals is related to brave and cautious reasoning, as we will see in Section 3.2.

In Figure 2.1 we illustrate important relations between modal literals that are immediately evident from Definition 3.

$$\begin{tikzcd}
K\,a \arrow[rr, leftrightarrow, "\text{either is false}"] \arrow[dd, "\text{implies}"] \arrow[ddrr, leftrightarrow, "\text{opposite}"] & & K\,\mathit{not}\,a \arrow[dd, "\text{implies}"] \arrow[ddll, leftrightarrow] \\
 & & \\
M\,a \arrow[rr, leftrightarrow, "\text{either is true}"] & & M\,\mathit{not}\,a
\end{tikzcd}$$

Figure 2.1: Relations between pairs of modal literals over an atom a relative to a collection of interpretations.

In principle, a *model* of a program is an interpretation that satisfies all rules of a program. But the satisfaction of modal literal depends on a collection of interpretations, which leads to the following definition.

Definition 4 (Models of a program)**.** Let $\mathcal{A}$ be a collection of interpretations and let $I \in \mathcal{A}$. Then I satisfies a ground rule r relative to $\mathcal{A}$, in symbols $I \models_{\mathcal{A}} r$, if either i) $I \models a$ for some atom $a \in \mathrm{Head}(r)$, ii) $I \not\models \ell$ for some standard literal $\ell \in \mathrm{Body}(r)$, or iii) $\mathcal{A} \not\models \varphi$ for some modal literal $\varphi \in \mathrm{Body}(r)$. We say I is a *model* of Π relative to $\mathcal{A}$ if I satisfies all rules in $\mathrm{ground}(\Pi)$ relative to $\mathcal{A}$.

2.2 Program Transformations

For logic programs that cannot be evaluated in a deterministic way, it is common to assume the truth value of some parts of the program and then create a *reduct* of the program relative to that assumption. The reduct is typically a simpler logic program with well known semantics for which a solver is readily available. Solutions of the reduct must then be checked for consistency with the initial assumption.

For example, the well known GL reduct (Gelfond and Lifschitz, 1991) for disjunctive logic programs or the more general FLP reduct (Faber et al., 2010) are program transformations relative to a fixed interpretation. The GL reduct is a program that is free of negation, which results from the original program by applying transformation rules that depend on the interpretation. The FLP reduct is a program consisting exactly of those rules of the original program whose bodies are satisfied by the interpretation. In both examples, the interpretation is an answer set of the original program if it is a subset minimal model of the reduct.

In this section, we define the *epistemic reduct* (Shen and Eiter, 2016) relative to an *epistemic guess*. An epistemic guess of a program represents a truth assignment of the modal literals occurring in the program.

2.2.1 *Truth Assignments of Modal Literals*

An expression of the form $M\,\ell$ over a standard literal ℓ is weaker than an expression of the form $K\,\ell$ over the same literal in the sense that $M\,\ell$ is true whenever $K\,\ell$ is true but not the other way round.

Definition 5 (Weak and strong modal literals)**.** We call a modal literal over a standard literal ℓ *weak* if it has the form $M\,\ell$, and *strong* if it has the form $K\,\ell$. By $\mathrm{weak}(\varphi)$ we denote the mapping of modal literals φ to their *weak form*, where

$$\mathrm{weak}(\varphi) := \begin{cases} \varphi & \text{if } \varphi \text{ is weak,} \\ \overline{\varphi} & \text{if } \varphi \text{ is strong.} \end{cases} \tag{2.3}$$

We already know that a strong modal literal $K\,\ell$ and its weak form $M\,\bar{\ell}$ cannot both be true and cannot both be false at the same time, since they are opposites of each other. This allows us to represent the set of all modal literals that occur in a program in a canonical way. The following mapping Ep[1] defines the set of weak modal literals of a program by mapping all modal literals to their weak form.

Definition 6 (Set of weak modal literals)**.** Let Π be a program. We denote by

$$\mathrm{Ep}(\Pi) := \{\, \mathrm{weak}(\varphi) \mid \varphi \text{ occurs in } \mathrm{ground}(\Pi) \,\} \tag{2.4}$$

the *set of weak modal literals* ρ *obtained from* Π such that ρ or its opposite occurs in $\mathrm{ground}(\Pi)$.

Remark 2. Disjunctive epistemic logic programs, as defined above, are a special class of general logic programs, as defined in (Shen and Eiter, 2016). We view weak modal literals $M\,\ell$ over a standard literal ℓ as shorthand for epistemic negations $N\,\mathit{not}\,\ell$. Thus, each set of weak modal literals corresponds to a set of epistemic negations.

Now any subset of $\mathrm{Ep}(\Pi)$ represents a truth assignment of the modal literals occurring in Π, which leads to the following definition.

[1]The symbol Ep is borrowed from (Shen and Eiter, 2016) in which $\mathrm{Ep}(\Pi)$ maps a program Π to the set of epistemic negations occurring in Π.

Definition 7 (Epistemic guess)**.** An *epistemic guess on the truth of weak modal literals of Π* (guess of Π for short) is a subset Φ of $\mathrm{Ep}(\Pi)$.

With a guess Φ of a ground program Π, we can determine the truth value of any modal literal occurring in Π:

1. If $M\,\ell \in \Phi$, then $M\,\ell$ is true and $K\,\overline{\ell}$ is false
2. If $M\,\ell \in \mathrm{Ep}(\Pi) \setminus \Phi$, then $K\,\overline{\ell}$ is true and $M\,\ell$ is false

Example 3 (Guesses on truth)**.** For the epistemic logic program

$$\Pi_2: \qquad \begin{array}{rcll} \mathrm{a} & \leftarrow & K\,\mathit{not}\,\mathrm{b}. & r_1 \\ \mathrm{b} & \leftarrow & K\,\mathit{not}\,\mathrm{a}. & r_2 \\ \mathrm{c} & \leftarrow & M\,\mathit{not}\,\mathrm{a}. & r_3 \end{array}$$

the set of weak modal literals $\mathrm{Ep}(\Pi_2)$ is $\{M\,\mathrm{a}, M\,\mathrm{b}, M\,\mathit{not}\,\mathrm{a}\}$. There are eight possible subsets of $\mathrm{Ep}(\Pi_2)$ each representing a guess of Π_2 (Figure 2.2).

$$\begin{tikzcd}
L_3 & & \{M\,\mathrm{a}, M\,\mathrm{b}, M\,\mathit{not}\,\mathrm{a}\} & \\
L_2 & \{M\,\mathrm{a}, M\,\mathrm{b}\} \arrow[ur] & \{M\,\mathrm{a}, M\,\mathit{not}\,\mathrm{a}\} \arrow[u] & \{M\,\mathrm{b}, M\,\mathit{not}\,\mathrm{a}\} \arrow[ul] \\
L_1 & \{M\,\mathrm{a}\} \arrow[u] \arrow[ur] & \{M\,\mathrm{b}\} \arrow[ul] \arrow[ur] & \{M\,\mathit{not}\,\mathrm{a}\} \arrow[u] \arrow[ul] \\
L_0 & & \varnothing \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.2: The partial order of guesses of Π_2 defined by the subset relation is shown in form of a Hasse diagram. Note that at each level L_k guesses have cardinality k.

2.2.2 *The Epistemic Reduct*

In previous works, reducts of programs with epistemic specifications are often defined relative to a collection of interpretations, see, e.g., (Gelfond, 1991, 2011; Truszczynski, 2011; Kahl, 2014). The epistemic reduct of (Shen and Eiter, 2016) follows a different approach as it is defined relative to an epistemic guess. Below we have adapted its definition to the syntax of ELP programs.

Definition 8 (Epistemic reduct)**.** Let Π be an ELP program and let $\Phi \subseteq \mathrm{Ep}(\Pi)$ be a guess of Π. The *epistemic reduct* Π^{Φ} of Π with respect to Φ is obtained from ground(Π) by applying the transformation rules of Table 2.2 for each pair (r, φ), where r is a rule in ground(Π) and φ is a modal literal occurring in r.

Modal literal φ	$\rho = \mathrm{weak}(\varphi)$	If $\rho \in \Phi$	If $\rho \in \mathrm{Ep}(\Pi) \setminus \Phi$
$M\,\ell$	$M\,\ell$	remove φ	replace φ with ℓ
$K\,\ell$	$M\,\bar{\ell}$	remove r	replace φ with ℓ

Table 2.2: Transformation rules of the epistemic reduct

Remark 3. The epistemic reduct of a program Π, as defined here, is a disjunctive logic program without modal literals and without nested expressions. This differs from the original definition of the epistemic reduct in (Shen and Eiter, 2016). As an example, suppose $M\,a \in \mathrm{Ep}(\Pi) \setminus \Phi$ is false w.r.t. some guess Φ of Π. By the original definition, $M\,a$, which is seen as a shorthand for the epistemic negation $N\,not\,a$, is replaced with the doubly negated expression *not not a* wherever it occurs in ground(Π). In our definition, we replace $M\,a$ with a instead, which is semantically equivalent to *not not a* in a program under FLP answer set semantics.

A program Π with $n = |\mathrm{Ep}(\Pi)|$ has 2^n possible epistemic guesses with respect to which we can create epistemic reducts.

Example 4. We can transform the following program into four different reducts:

$$\Pi_3: \qquad \mathrm{a} \leftarrow K\,not\,\mathrm{b}. \qquad r_1$$
$$\mathrm{b} \leftarrow K\,not\,\mathrm{a}. \qquad r_2$$

The first rule r_1 contains the strong modal literal $\varphi_1 = K\,not\,\mathrm{b}$ and the second rule r_2 contains the strong modal literal $\varphi_2 = K\,not\,\mathrm{a}$. Let $\rho_1 = M\,\mathrm{b}$ and $\rho_2 = M\,\mathrm{a}$ be the two weak modal literals obtained from Π_3, i.e., $\mathrm{Ep}(\Pi_3) = \{\rho_1, \rho_2\}$. We transform Π_3 into all possible epistemic reducts $\Pi_3^{\Phi_i}$ by applying the transformation rules in Definition 8 relative to each $\Phi_i \subseteq \{\rho_1, \rho_2\}$ for the pairs (r_1, φ_1) and (r_2, φ_2):

$$\begin{array}{ll}
\Phi_1 = \{\rho_1, \rho_2\} & \Pi_3^{\Phi_1} = \emptyset \\
\Phi_2 = \{\rho_2\} & \Pi_3^{\Phi_2} = \{\mathrm{a} \leftarrow not\,\mathrm{b}\} \\
\Phi_3 = \{\rho_1\} & \Pi_3^{\Phi_3} = \{\mathrm{b} \leftarrow not\,\mathrm{a}\} \\
\Phi_4 = \emptyset & \Pi_3^{\Phi_4} = \{\mathrm{a} \leftarrow not\,\mathrm{b}; \mathrm{b} \leftarrow not\,\mathrm{a}\}
\end{array}$$

2.3 Epistemic Answer Set Semantics

The general epistemic semantics in (Shen et al., 2014) is defined in terms of the answer sets of the epistemic reduct under some answer set semantics $\mathcal{X}$ for logic programs without epistemic specifications. However, as noted in Remark 3, doubly negated expressions may still occur when applying the original definition of the epistemic reduct to ELP programs. If $\mathcal{X}$ is the FLP semantics (Faber et al., 2010), then double negation cancels each other out. Under this assumption, our definition of the epistemic reduct, which does not contain nested expressions, is semantically equivalent to the original epistemic reduct. For further discussion, see Section 6.2.

Definition 9 (Answer sets of DLP programs)**.** Let the *FLP reduct* of a disjunctive logic program Π with respect to an interpretation I be the set

$$f\Pi^I := \{\, r \in \Pi \mid I \models b \text{ for all } b \in \mathrm{Body}(r)\,\} \tag{2.5}$$

of all rules whose body is satisfied by I. Then I is an answer set of Π, if I is a subset-minimal model of $f\Pi^I$.

In the remainder of this work, we will assume FLP answer set semantics for programs Π without modal literals. By $\mathrm{AS}(\Pi)$ we denote the collection of all answer sets of Π.

Example 5. Let Π be the program $\Pi_3^{\Phi_4} = \{\mathrm{a} \leftarrow \mathit{not}\,\mathrm{b}; \mathrm{b} \leftarrow \mathit{not}\,\mathrm{a}\}$ from Example 4. We transform Π into all possible FLP reducts Π^{I_i} with respect to each $I_i \subseteq \{\mathrm{a},\mathrm{b}\}$:

$$\begin{aligned} I_1 &= \{\mathrm{a},\mathrm{b}\} & f\Pi^{I_1} &= \emptyset \\ I_2 &= \{\mathrm{a}\} & f\Pi^{I_2} &= \{\mathrm{a} \leftarrow \mathit{not}\,\mathrm{b}\} \\ I_3 &= \{\mathrm{b}\} & f\Pi^{I_3} &= \{\mathrm{b} \leftarrow \mathit{not}\,\mathrm{a}\} \\ I_4 &= \emptyset & f\Pi^{I_4} &= \{\mathrm{a} \leftarrow \mathit{not}\,\mathrm{b}; \mathrm{b} \leftarrow \mathit{not}\,\mathrm{a}\} \end{aligned}$$

The interpretation I_1 is a model of $f\Pi^{I_1}$ but not a subset-minimal one, since every subset of I_1 is also a model of $f\Pi^{I_1}$. The interpretations I_2 and I_3 are subset-minimal models of $f\Pi^{I_2}$ and $f\Pi^{I_3}$, respectively. The interpretation I_4 is not a model of $f\Pi^{I_4}$. So $\mathrm{AS}(\Pi) = \{\{\mathrm{a}\},\{\mathrm{b}\}\}$.

By replacing $\mathcal{X}$ with FLP in Definition 8 (General epistemic semantics) of (Shen and Eiter, 2016), we obtain the epistemic FLP answer set semantics (EFLP or $\mathrm{SE16}_{\mathrm{FLP}}$ semantics for short).

Definition 10 (EFLP semantics)**.** Let Π be an ELP program, let Φ be a guess of Π. The collection $\mathcal{A}$ of all answer sets of the epistemic reduct Π^Φ under FLP answer set semantics is a *candidate world view* of Π with respect to Φ, if

i) $\mathcal{A} \neq \emptyset$, i.e., Π^Φ is consistent,
ii) every $\rho \in \Phi$ is true in $\mathcal{A}$, and
iii) every $\rho \in \mathrm{Ep}(\Pi) \setminus \Phi$ is false in $\mathcal{A}$.

A candidate world view $\mathcal{A}$ with respect to Φ is a *world view* if Φ is maximal, i.e., there is no other candidate world view with respect to a guess $\Phi' \supset \Phi$.

Example 6 (Presumption of innocence)**.** The well-known presumption of innocence can be expressed as the following ELP program[2]:

$$\begin{array}{llr} \Pi_4: & \text{innocent(“John”)} \vee \text{guilty(“John”)}. & r_1 \\ & \text{innocent(“John”)} \leftarrow M\,\textit{not}\,\text{guilty(“John”)}. & r_2 \end{array}$$

Here, the weak modal literal $\rho = M\,\textit{not}\,\text{guilty(“John”)}$ expresses that John *might not* be guilty, which is the opposite of saying that it is *known* that John is guilty.

Now, from the set of weak modal literals $\text{Ep}(\Pi_4) = \{\rho\}$ we get two possible guesses $\Phi_1 = \{\rho\}$ and $\Phi_2 = \emptyset$. To satisfy the maximality condition, we first try the largest guess Φ_1 and check if there is a world view with respect to Φ_1. The epistemic reduct of Π_4 with respect to Φ_1 is the program

$$\begin{array}{llr} \Pi_4^{\Phi_1}: & \text{innocent(“John”)} \vee \text{guilty(“John”)}. & r_1 \\ & \text{innocent(“John”)}. & r_2' \end{array}$$

Under FLP semantics, the program $\Pi_4^{\Phi_1}$ is consistent and has the unique answer set $I = \{\text{innocent(“John”)}\}$. Let $\mathcal{A} = \{I\}$. As guilty(“John”) is false in I, it follows that $M\,\textit{not}\,\text{guilty(“John”)}$ is true in $\mathcal{A}$, thus $\mathcal{A}$ is a candidate world view of Π_4. As Φ_1 is the largest possible guess, $\mathcal{A}$ is a world view of Π_4 with respect to Φ_1. As $\Phi_2 \subset \Phi_1$, there is no world view with respect to Φ_2. Hence, $\mathcal{A} = \{\{\text{innocent(“John”)}\}\}$ is the only world view of Π_4. Therefore, the atom innocent(“John”) is true in Π_4, which means that John is innocent, as expected.

[2]This example is borrowed from (Shen and Eiter, 2016).

Evaluating Epistemic Logic Programs

3

Solving the *world view enumeration problem* of an epistemic logic program means computing all its world views. A common approach, followed by (e.g.) the solvers *Wviews*, *ESmodels*, *ELPS*, and *ELPsolve* (cf. Table 1.1), is to use an ASP solver to compute candidate solutions with a single ore multiple solver calls. The problem associated with this approach is that there may be exponentially many candidate solutions in the number of modal literals (epistemic negations) of a program, even if no world view exists. To avoid this problem, epistemic guesses of programs that do not lead to a solution must be filtered out in the first place. For example, the solver *ELP-ASP* eliminates in advance all epistemic guesses that would not lead to a world view. This is done by iteratively evaluating checking programs relative to the answer set of a guessing program (Son et al., 2017). In our approach, the input program is transformed into a series of HEX programs containing declarative guessing and checking rules, so that the answer sets of these HEX program encode world views of the input program. The HEX programs contain brave or cautious atoms, which are external atoms with access to an ASP solver under brave or cautious reasoning mode.

With modularity in mind, we will compose these HEX programs from smaller programs, namely the *building blocks* presented in Section 3.3. These building blocks consist of: the generic epistemic reduct, the guessing program, the consistency checking program, the cardinality checking program, and the subset checking program.

The EHEX algorithm presented in Section 3.4 performs an ordered search, ordered by the cardinality of epistemic guesses of the input program, to satisfy the maximality condition of world views. Thus, when the algorithm finds a guess Φ representing a world view, it then adds constraints that prevent it from generating new guesses that are a subset of Φ in the next iteration.

In Section 3.5 we show the correctness of the algorithm, i.e., its soundness and completeness.

We close this chapter with Section 3.6 proposing some possible optimization that aim at pruning the search space.

3.1 Logic Programs with External Atoms

Higher-order logic programs with external atoms (HEX programs, Eiter et al. (2005)) extend disjunctive logic programs with *higher-order atoms* as well as *external atoms*. In this work, we use only external atoms, through which it is possible for HEX programs to access any external source of computation with respect to an interpretation. In practice, access to an external source of computation depends on whether all possible combinations of inputs and outputs to and from external atoms can be interpreted in a meaningful way. Such interpretations of inputs and outputs are formally defined via a Boolean-valued oracle function.

Later in this section we will introduce external atoms with access to an ASP solver in brave or in cautious reasoning mode. Those atoms take as input a DLP program extended with a dynamically generated set of facts and a query, and output the result of the query. But first, we define HEX programs in general.

3.1.1 *HEX Syntax*

The syntax of HEX programs is analogous to the syntax of ELP programs, except that HEX programs do not contain modal literals, but *external atoms* of the form

$$\&g[p_1, \dots, p_n](c_1, \dots, c_m). \tag{3.1}$$

Here, $\&g$ is an external predicate name, $p_1, \dots, p_n$ is a list of terms or predicate symbols, called *input list*, and $c_1, \dots, c_m$ are constant terms, called *output list*. Each external predicate $\&g$ is assumed to have a fixed input arity $n \geq 0$ and a fixed output arity $m \geq 0$. A *HEX atom* is either an atom or an external atom. A *HEX literal* is of the form a or *not* a where a is a HEX atom and *not* is the default negation operator. If clear from the context, we refer to a HEX literal as literal.

Definition 11 (HEX programs)**.** A *HEX program* is a set of rules of the form

$$a_1 \lor \ldots \lor a_m \leftarrow b_1, \dots, b_n, \tag{3.2}$$

where each a_i is an atom and each b_i is a HEX literal. The notions of the head and the body of a rule, as well as as the notion of a ground program (rule, literal, atom, term) and the notions of facts and constraints, are defined as for ELP programs.

Additionally, for a rule r of the form (3.2), the *positive body* of r is

$$\text{Body}^+(r) := \{a \mid a \in \text{Body}(r), \text{ where } a \text{ is an atom}\}, \tag{3.3}$$

and the *negative body* of r is

$$\text{Body}^-(r) := \{a \mid \textit{not}\, a \in \text{Body}(r), \text{ where } a \text{ is an atom}\}. \tag{3.4}$$

As in ELP programs, we consider strong (classical) negation to be a special syntax to make programs easier to read (see Remark 1).

3.1.2 *HEX Semantics*

The semantics of HEX programs is a proper generalization of the answer set semantics by Gelfond and Lifschitz (1991), but using the *FLP reduct* (Faber et al., 2010);

while the reduct is equivalent to the Gelfond-Lifschitz reduct reduct for ordinary DLP programs, the FLP reduct provides a more natural semantics for extensions like aggregate atoms or HEX atoms.

The semantics of a ground external atom $\&g[\boldsymbol{p}](\boldsymbol{c})$, where $\boldsymbol{p} = p_1, \dots, p_n$ and $\boldsymbol{c} = c_1, \dots, c_m$, with respect to an interpretation I is given by the value of a $(1+n+m)$-ary *Boolean-valued oracle function* $f_{\&g}$ that is defined for all possible values of I, $\boldsymbol{p}$, and $\boldsymbol{c}$. An external atom $\&g[\boldsymbol{p}](\boldsymbol{c})$ is true relative to I if and only if $f_{\&g}(I, \boldsymbol{p}, \boldsymbol{c}) = 1$.

Given the satisfaction of external atoms, the satisfaction of ground rules in a HEX program is defined as usual.

Definition 12 (Answer sets of a HEX program)**.** Let the *FLP reduct* of a HEX program Π with respect to an interpretation I be the set

$$f\Pi^I := \{\, r \in \Pi \mid I \models b \text{ for all } b \in \mathrm{Body}(r) \,\} \tag{3.5}$$

of all rules whose body is satisfied by I. Then I is an answer set of Π, if I is a subset-minimal model of $f\Pi^I$.

Example 7. Consider the HEX program $\Pi_5 = \{\mathrm{p} \leftarrow \&\mathrm{id}[\mathrm{p}]\}$, where $\&\mathrm{id}[\mathrm{p}]$ is true under an interpretation I exactly when p is true under I. It has the single answer set $I = \varnothing$, which is indeed a subset-minimal model of $f\Pi_5^I = \varnothing$.

3.2 Brave and Cautious Atoms

Since disjunctive logic programs Π have multiple answer sets in general, inference from Π is often defined in two dual reasoning modes. A ground atom a can be inferred from Π in i) brave reasoning mode if a is contained in some answer sets of Π, and in ii) cautious reasoning mode if a is contained in all answer sets of Π. We express this in symbols as $\Pi \models_{\mathrm{b}} a$ and $\Pi \models_{\mathrm{c}} a$, respectively. Clearly, the set of all brave consequences of Π is the union of all answer sets of Π and the set of all cautious consequences of Π is the intersection of all answer sets of Π.

If $\mathcal{A}$ is the collection of all answer sets of a consistent program Π, i.e., $\mathcal{A} = \mathrm{AS}(\Pi)$ and $\mathcal{A} \neq \varnothing$, then we can reduce checking the satisfaction of ground modal literals by $\mathcal{A}$ to reasoning over Π under brave or cautious reasoning mode. The next proposition follows from Definition 3, where *interpretations* is replaced with *all answer set of Π*.

Proposition 1. *Let Π be a consistent logic program under FLP answer set semantics and let a be a ground atom. Furthermore, let $\mathcal{A}$ be the nonempty collection of all answer sets of Π. Then the strong modal literal $\mathrm{K}\,a$ is true in $\mathcal{A}$ if and only if a is a cautious consequence of Π, and the weak modal literal $\mathrm{M}\,a$ is true in $\mathcal{A}$ if and only if a is a brave* *consequence of Π; in symbols:*

$$\mathcal{A} \models K\,a \quad \textit{iff} \quad \Pi \models_c a \tag{3.6}$$
$$\mathcal{A} \models M\,a \quad \textit{iff} \quad \Pi \models_b a \tag{3.7}$$

Checking the satisfaction of modals of the two other forms $K\,not\,a$ and $M\,not\,a$ can be reduced to $\Pi \not\models_b a$ and $\Pi \not\models_c a$, respectively:

$$\mathcal{A} \models K\,not\,a \quad \text{iff} \quad \Pi \not\models_b a \tag{3.8}$$
$$\mathcal{A} \models M\,not\,a \quad \text{iff} \quad \Pi \not\models_c a \tag{3.9}$$

We now have all the necessary ingredients to introduce external atoms for checking epistemic guesses. As mentioned earlier, HEX programs allow bidirectional communication between a program and an external sources of computation via external atoms. If we have access to an external source that can enumerate all answer sets of a given program, i.e., access to another instance of a HEX solver, then we can express brave and cautious queries over a subprogram through dedicated external atoms. The general concept of *nested HEX programs*, i.e., HEX programs that communicate with each other via external atoms, was introduced in (Eiter et al., 2013).

3.2.1 *Syntax of Brave and Cautious Atoms*

We use a syntax for external atoms similar to that of the *Nested HEX Plugin*,[1] but adapted for brevity. We revisit this plugin in Chapter 4.

Definition 13. A *brave atom* or a *cautious atom* is an external atom of the form

$$\&b[\Pi, p, q](t_1, \dots, t_n) \quad \text{or} \quad \&c[\Pi, p, q](t_1, \dots, t_n), \tag{3.10}$$

respectively, where $\&b$ and $\&c$ are uniquely associated external predicate names, the input list specifies a DLP program Π, an input predicate p, and a query predicate q, and the output list specifies $n \geq 0$ terms such that $q(t_1, \dots, t_n)$ is a query atom.

3.2.2 *Satisfaction of Brave and Cautious Atoms*

With a brave or cautious atom that has the input list Π, p, q and the output list $t_1, \dots, t_n$ we want to express a brave or cautious query $q(t_1, \dots, t_n)$ over the specified program Π extended with a variable set of input facts $\Gamma_p(I)$ that depends on the satisfaction of atoms of the form $p(a)$, where p is the input predicate and a is a functional term, by an interpretation I.

[1]The *Nested HEX Plugin* for *dlvhex* by Christoph Redl is available at `http://www.kr.tuwien.ac.at/research/systems/dlvhex/nestedhexplugin.html` (visited on 2022-07-29)

Definition 14 (Input facts)**.** Let p be a predicate name. By Γ_p we denote the function associated with p that maps each interpretation I to a set

$$\Gamma_p(I) := \{a \mid I \models p(a)\} \tag{3.11}$$

of facts a such that each a is encoded as a functional term of an atom $p(a)$ that is satisfied by I. We refer to $\Gamma_p(I)$ as the set of *input facts* due to p under I.

Example 8 (Graph input)**.** Suppose we want a graph as input for a brave or cautious atom ξ in a HEX program Π. To this end, we specify g as the input predicate of ξ. Now, for example, if we have an interpretation

$$I = \{\ldots, \mathrm{g}(\mathrm{node}(\mathrm{v})), \mathrm{g}(\mathrm{node}(\mathrm{w})), \mathrm{g}(\mathrm{edge}(\mathrm{v}, \mathrm{w})), \ldots\},$$

where g encodes two nodes and an edge as functional terms, then the resulting graph input due to g under I is the set $\Gamma_{\mathrm{g}}(I) = \{\mathrm{node}(\mathrm{v}), \mathrm{node}(\mathrm{w}), \mathrm{edge}(\mathrm{v}, \mathrm{w})\}$.

In Section 3.1.2, we have already stated that the satisfaction of an external atom $\&e[\boldsymbol{p}](\boldsymbol{t})$ under an interpretation I is defined via its associated Boolean-valued oracle function $f_{\&e}$, in symbols:

$$I \models \&e[\boldsymbol{p}](\boldsymbol{t}) \quad \text{iff} \quad f_{\&e}(I, \boldsymbol{p}, \boldsymbol{t}) = 1. \tag{3.12}$$

It remains to define the oracle functions associated with brave and cautious atoms.

Definition 15 (Oracle functions of brave and cautious atoms)**.** The oracle functions $f_{\&b}$ and $f_{\&c}$ associated with brave and cautious atoms, respectively, take as arguments an interpretation I, a DLP program Π, an input predicate p, a query predicate q, and a list of ground terms $\boldsymbol{t} = t_1, \ldots, t_n$, $n \geq 0$. They are defined as

$$f_{\&b}(I, \Pi, p, q, \boldsymbol{t}) := \begin{cases} 1 & \text{if } \Pi \cup \Gamma_p(I) \models_{\mathrm{b}} q(\boldsymbol{t}), \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

$$f_{\&c}(I, \Pi, p, q, \boldsymbol{t}) := \begin{cases} 1 & \text{if } \Pi \cup \Gamma_p(I) \models_{\mathrm{c}} q(\boldsymbol{t}), \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

where $\Gamma_p(I)$ is the set of input facts due to p under I.

The following proposition establishes the relationship between modal literals and external atoms.

Proposition 2. *Let Π be a DLP program and let Δ be a set of facts such that $\Pi \cup \Delta$ is consistent. For any interpretation I with $\Gamma_p(I) = \Delta$, the following equivalences hold:*

$$\begin{array}{rr} I \models \&b[\Pi, p, q](\boldsymbol{t}) & \\ \Pi \cup \Gamma_p(I) \models_b q(\boldsymbol{t}) & \text{(i)} \\ \mathrm{AS}(\Pi \cup \Delta) \models M q(\boldsymbol{t}) & \end{array} \qquad \begin{array}{rr} I \models \&c[\Pi, p, q](\boldsymbol{t}) & \\ \Pi \cup \Gamma_p(I) \models_c q(\boldsymbol{t}) & \text{(ii)} \\ \mathrm{AS}(\Pi \cup \Delta) \models K q(\boldsymbol{t}) & \end{array}$$

$$\begin{array}{rr} I \models \mathit{not}\, \&b[\Pi, p, q](\boldsymbol{t}) & \\ \Pi \cup \Gamma_p(I) \not\models_b q(\boldsymbol{t}) & \text{(iii)} \\ \mathrm{AS}(\Pi \cup \Delta) \models K \mathit{not}\, q(\boldsymbol{t}) & \end{array} \qquad \begin{array}{rr} I \models \mathit{not}\, \&c[\Pi, p, q](\boldsymbol{t}) & \\ \Pi \cup \Gamma_p(I) \not\models_c q(\boldsymbol{t}) & \text{(iv)} \\ \mathrm{AS}(\Pi \cup \Delta) \models M \mathit{not}\, q(\boldsymbol{t}) & \end{array}$$

Proposition 2 follows directly from the semantics of external atoms (3.12), Definition 14, Definition 15, and Proposition 1.

Remark 4. A special case of Proposition 2 is when Δ is empty. In this case, for example, for any interpretation I with $\Gamma_p(I) = \Delta$, the following equivalence holds:

$$I \models \&b[\Pi, p, q](\boldsymbol{t}) \quad \text{iff} \quad \Pi \models_b q(\boldsymbol{t}) \quad \text{iff} \quad \mathrm{AS}(\Pi) \models M q(\boldsymbol{t}). \tag{3.15}$$

3.2.3 *Applications*

We can now use brave and cautious atoms to check whether an epistemic guess is consistent. The example below outlines the idea by means of a simple checking program.

Example 9 (Checking consistency)**.** Consider the ELP program Π_3 of Example 4 where $\mathrm{Ep}(\Pi_3) = \{M\,\mathrm{a}, M\,\mathrm{b}\}$ and let $\Phi = \{M\,\mathrm{a}\}$ be a guess of Π_3. Then the epistemic reduct Π_3^Φ of Π_3 with respect to Φ is:

$$\Pi_3^\Phi: \qquad \mathrm{a} \leftarrow \mathit{not}\, \mathrm{b}. \qquad r_1$$

Now, let $\mathcal{A} = \{\{a\}\}$ be the collection of all answer sets of Π_3^Φ. To check the consistency of Φ with $\mathcal{A}$, we need to check whether i) $\mathcal{A} \models M\,\mathrm{a}$ and ii) $\mathcal{A} \not\models M\,\mathrm{b}$ hold.To this end, we construct the HEX program Π_{ok} such that it has the the single answer set $\{\mathrm{ok}\}$ iff i) and ii) hold:

$$\Pi_{\mathrm{ok}}: \qquad \mathrm{ok} \leftarrow \&b[\Pi_3^\Phi, \mathrm{p}, \mathrm{a}], \mathit{not}\, \&b[\Pi_3^\Phi, \mathrm{p}, \mathrm{b}]. \qquad r_1$$

Note that no atom over the input predicate p occurs in Π_{ok} and thus for any interpretation I of Π_{ok}, $\Gamma_\mathrm{p}(I)$ is empty (cf. Remark 4). As a is a brave consequence of Π_3^Φ, the brave atom $\&b[\Pi_3^\Phi, \mathrm{p}, \mathrm{a}]$ is true under all interpretations of Π_{ok}. As b is not a brave consequence of Π_3^Φ, the brave atom $\&b[\Pi_3^\Phi, \mathrm{p}, \mathrm{b}]$ is false under all interpretations of Π_{ok}. Therefore, the body of r_1 in Π_{ok} is satisfied by any interpretation of Π_{ok}. Hence, $\{\mathrm{ok}\}$ is the only FLP answer set of Π_{ok}, which means that Φ is consistent with $\mathcal{A}$.

The program Π_{ok} in itself is not very useful, as it only checks a specific guess – but it does so in a declarative way. In the following section, we develop this idea further using building blocks.

3.3 Building Blocks

In the ongoing subsections we introduce the *building blocks* generated by the EHEX algorithm at run time. Building blocks are HEX programs. The algorithm combines these building blocks into larger HEX programs whose answer set collections represent one or more world views of the input program.

To simplify programs, we use a shorthand notation for auxiliary atoms.

Notation 1 (Auxiliary atoms)**.** An *auxiliary atom* is an ordinary atom associated with an intended meaning. The following expressions, where φ is a modal literal, ρ is a weak modal literal, and Φ is an epistemic guess, are shorthand notations for auxiliary atoms:

$guess(\rho)$	expresses that ρ is guessed to be true.
$true(\varphi)$	expresses that φ is true.
$member(\rho, \Phi)$	states that ρ is a member of Φ.
$input(guess(\rho))$	specifies $guess(\rho)$ as input fact due to *input*.
$gnd(\rho)$	represents a ground modal literal ρ.

For a set A of ground atoms, we denote by $\mathrm{Aux}(A)$ the set of auxiliary atoms in A.

Definition 16 (Aux-equivalence)**.** Let $\mathcal{X}$ be an answer set semantics for logic programs without epistemic specifications. Two programs Π_1, Π_2 are said to be *equivalent with respect to auxiliary atoms* (aux-equivalent for short) under $\mathcal{X}$ if they have the same answer sets minus auxiliary atoms, i.e., if

$$\{A_1 \setminus \mathrm{Aux}(A_1) \mid A_1 \in \mathrm{AS}_{\mathcal{X}}(\Pi_1)\} = \{A_2 \setminus \mathrm{Aux}(A_2) \mid A_2 \in \mathrm{AS}_{\mathcal{X}}(\Pi_2)\}. \qquad (3.16)$$

Example 10. Consider the program $\Pi_7 = \{\mathrm{a} \leftarrow \textit{not not}\,\mathrm{a}\}$ containing the nested expression *not not a* and the program $\Pi'_7 = \{\mathrm{a} \leftarrow \textit{not}\,\mathrm{not\_a}, \mathrm{not\_a} \leftarrow \textit{not}\,\mathrm{a}\}$ containing the auxiliary atom not_a. Under NEX semantics they are aux-equivalent, because here both have the same answer sets $\{\mathrm{a}\}$ and $\varnothing$ minus auxiliary atoms. But under FLP semantics they are not aux-equivalent, because here Π_7 has the single answer set $\varnothing$ and Π'_7 has the answer sets $\{\mathrm{a}\}$ and $\varnothing$ minus auxiliary atoms.

For the auxiliary atoms in Notation 1, we assume that they are fresh with respect to pre-existing atoms. For example, if $guess(M\,b)$ is the auxiliary atom a, where $M\,b$ is a modal literal over the pre-existing atom b, then a and b have different predicate names.

3.3.1 *The Generic Epistemic Reduct*

Recall the epistemic reduct Π^{Φ} of a program Π, which is a transformation of Π relative to an epistemic guess Φ of Π (cf. Definition 8). The basic idea of the *generic* epistemic reduct, denoted $\mathring{\Pi}$, is to decouple this transformation from a specific guess. Instead of removing or replacing modal literals based on their truth value with respect to Φ, in the generic epistemic reduct, modal literals are replaced with ordinary literals over auxiliary atoms. The satisfaction of these replacement literals depends on the satisfaction of additional rules in $\mathring{\Pi}$, which in turn depends on the satisfaction of guessing atoms of another building block.

Definition 17 (Generic epistemic reduct)**.** Let Π be an ELP program. The *generic epistemic reduct* $\mathring{\Pi}$ of Π is obtained from Π by applying the transformation rules of Table 3.1 for each modal literal φ occurring in Π such that

i) each φ is replaced with a *substitution literal* ℓ_φ, and
ii) $\mathring{\Pi}$ contains the *substitution rules* for each ℓ_φ.

Modal literal φ	Substitution literal ℓ_φ	Substitution rules for ℓ_φ
$M\,a$	$\mathit{true}(M\,a)$	$\{\mathit{true}(M\,a) \leftarrow \mathit{guess}(M\,a);$
$K\,\mathit{not}\,a$	$\mathit{not}\,\mathit{true}(M\,a)$	$\mathit{true}(M\,a) \leftarrow a, \neg\mathit{guess}(M\,a)\}$
$K\,a$	$\mathit{true}(K\,a)$	$\{\mathit{true}(K\,a) \leftarrow a, \neg\mathit{guess}(M\,\mathit{not}\,a)\}$
$M\,\mathit{not}\,a$	$\mathit{not}\,\mathit{true}(K\,a)$	

Table 3.1: Transformation rules of the generic epistemic reduct

Remark 5 (On grounding). Note that, in contrast to the epistemic reduct, the grounding of Π is not required to obtain the epistemic reduct. However, if we want to extend $\mathring{\Pi}$ with a guessing program for Π, then we need to compute the complete set $\mathrm{Ep}(\Pi)$ of ground weak modal literals obtained from Π (cf. Definition 6).

Example 11. To illustrate the generic epistemic reduct, we use a simple program:

$$\Pi_8: \qquad \begin{aligned} \mathrm{p} &\leftarrow M\,\mathrm{p}. & r_1\\ \mathrm{r} &\leftarrow M\,\mathit{not}\,\mathrm{p}. & r_2 \end{aligned}$$

After applying the transformation rules for the two modal literals $M\,\mathrm{p}$ and $M\,\mathit{not}\,\mathrm{p}$, the resulting program is the generic epistemic reduct $\mathring{\Pi}_8$ of Π_8:

$$\mathring{\Pi}_8: \qquad \begin{aligned} \mathrm{p} &\leftarrow \mathit{true}(M\,\mathrm{p}). & \mathring{r}_1\\ \mathrm{r} &\leftarrow \mathit{not}\,\mathit{true}(K\,\mathrm{p}). & \mathring{r}_2\\ \mathit{true}(M\,\mathrm{p}) &\leftarrow \mathit{guess}(M\,\mathrm{p}). & r_3\\ \mathit{true}(M\,\mathrm{p}) &\leftarrow \mathrm{p}, \neg\mathit{guess}(M\,\mathrm{p}). & r_4\\ \mathit{true}(K\,\mathrm{p}) &\leftarrow \mathrm{p}, \neg\mathit{guess}(M\,\mathit{not}\,\mathrm{p}). & r_5 \end{aligned}$$

Here, the rules $\mathring{r}_1$ and $\mathring{r}_2$ result from r_1 and r_2, respectively, by replacing the modal literals with auxiliary ordinary literals (substitution literals). The rules r_3, r_4 are the substitution rules for $\mathit{true}(M\,\mathrm{p})$ and r_5 is the substitution rule for $\mathit{not}\,\mathit{true}(K\,\mathrm{p})$.

3.3.2 Guessing Programs

In what follows, we describe guessing programs, which are programs whose answer sets each encode a different epistemic guess.

Definition 18 (Guess encoding)**.** Let Π be an ELP program and let A be a set of ground atoms. Then A *encodes* an epistemic guess Φ of Π if for every $\rho \in \mathrm{Ep}(\Pi)$,

1. $guess(\rho)$ is in A iff ρ is in Φ, and
2. $\neg guess(\rho)$ is in A iff ρ is in $\mathrm{Ep}(\Pi) \setminus \Phi$.

The *guess encoding* G_Φ of Φ is the smallest set of ground atoms that encodes Φ.

For convenience, we identify sets of ground atoms with programs in which those atoms are seen as facts.

Example 12. For a program Π and an epistemic guess Φ of Π, the program $\mathring{\Pi} \cup \mathrm{G}_\Phi$ composed of the generic epistemic reduct $\mathring{\Pi}$ and the guess encoding G_Φ is aux-equivalent to the epistemic reduct Π^Φ.

Definition 19 (Guessing program)**.** Let Π be an ELP program. A *guessing rule* for a (ground) weak modal literal $\rho \in \mathrm{Ep}(\Pi)$ is of the form

$$guess(\rho) \vee \neg guess(\rho). \tag{3.17}$$

The *guessing program* G_Π for Π consists of the guessing rules for all $\rho \in \mathrm{Ep}(\Pi)$.

For referring to epistemic guesses encoded by sets of ground atoms or by the answer sets of a program, we introduce the following notation.

Notation 2. For a set A of ground atoms we denote by

$$\Phi(A) := \{\rho \mid guess(\rho) \in A\} \tag{3.18}$$

the set of weak modal literals encoded by A. For a HEX program Π we denote by

$$\Omega(\Pi) := \{\Phi(A) \mid A \in \mathrm{AS}(\Pi)\} \tag{3.19}$$

the collection of all sets of weak modal literals obtained from the answer sets of Π.

Using this notation, if a set A encodes an epistemic guess Φ, then $\Phi = \Phi(A)$ holds, i.e., we obtain Φ from the auxiliary atoms of the form $guess(\rho)$ in A, where ρ is a weak modal literal. For an ELP program Π, the collection $\Omega(\mathrm{G}_\Pi)$ of sets of modals obtained from the answer sets of the guessing program G_Π for Π represents all possible epistemic guesses of Π, i.e., it coincides with the power set of $\mathrm{Ep}(\Pi)$. Also, a guess Φ of Π is in $\Omega(\mathring{\Pi} \cup \mathrm{G}_\Phi)$ if and only if Π^Φ is consistent.

3.3.3 Consistency Checking Programs

We now describe checking epistemic guesses in a declarative way using external atoms. An epistemic logic program Π is consistent if it has a candidate world view, or equivalently, if there is a guess Φ of Π such that the epistemic reduct Π^{Φ} is consistent and Φ is consistent with the answer sets of Π^{Φ}. The former corresponds to checking the condition i) and the latter to checking the conditions ii) and iii) of candidate world views in Definition 10.

The epistemic reduct Π^{Φ} and the program $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ are aux-equivalent by construction, so we can check the consistency of Φ by reasoning over the answer sets of $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$. The HEX program C_{Π} defined below does just that with the help of brave and cautious atoms.

Definition 20 (Consistency checking program)**.** Let Π be an ELP program. First, for each $\rho \in \mathrm{Ep}(\Pi)$, define the *input declaration* D_{ρ} of ρ as the program

$$\mathrm{D}_{\rho} := \left\{ \begin{array}{c} \mathit{input}(\mathit{guess}(\rho)) \leftarrow \mathit{guess}(\rho). \\ \mathit{input}(\neg \mathit{guess}(\rho)) \leftarrow \neg \mathit{guess}(\rho). \end{array} \right\}, \tag{3.20}$$

such that, for each guess Φ of Π, the equation $\Gamma_{\mathit{input}}(A) = \mathrm{G}_{\Phi}$ holds due to *input* under the unique answer set A of the program $\mathrm{D}_{\rho} \cup \mathrm{G}_{\Phi}$.

Next, for each $\rho \in \mathrm{Ep}(\Pi)$, using brave or cautious atoms depending on ρ, define the *modal consistency constraints* C_{ρ} of ρ. Let $a(\boldsymbol{t})$ be the atom occurring in ρ and let $\boldsymbol{p}$ be the input list $\mathring{\Pi}, \mathit{input}, a$, where $\mathring{\Pi}$ is the generic epistemic reduct of Π. If ρ is of the form $M\, a(\boldsymbol{t})$, define C_{ρ} using brave atoms as the program

$$\mathrm{C}_{\rho} := \left\{ \begin{array}{l} \leftarrow \mathit{guess}(\rho), \mathit{not}\ \&b[\boldsymbol{p}](\boldsymbol{t}). \\ \leftarrow \neg \mathit{guess}(\rho), \&b[\boldsymbol{p}](\boldsymbol{t}). \end{array} \right\}, \tag{3.21a}$$

otherwise, if ρ is of the form $M\, \mathit{not}\ a(\boldsymbol{t})$, define C_{ρ} using cautious atoms as the program

$$\mathrm{C}_{\rho} := \left\{ \begin{array}{l} \leftarrow \mathit{guess}(\rho), \&c[\boldsymbol{p}](\boldsymbol{t}). \\ \leftarrow \neg \mathit{guess}(\rho), \mathit{not}\ \&c[\boldsymbol{p}](\boldsymbol{t}). \end{array} \right\}. \tag{3.21b}$$

Finally, the *consistency checking program* C_{Π} of Π is the program

$$\mathrm{C}_{\Pi} := \bigcup_{\rho \in \mathrm{Ep}(\Pi)} (\mathrm{D}_{\rho} \cup \mathrm{C}_{\rho}), \tag{3.22}$$

such that, for each guess Φ of Π, the program $\mathrm{C}_{\Pi} \cup \mathrm{G}_{\Phi}$ is consistent if and only if for the collection $\mathcal{A}$ of all answer sets of $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$,

every $\rho \in \Phi$ is true in $\mathcal{A}$, and
every $\rho \in \mathrm{Ep}(\Pi) \setminus \Phi$ is false in $\mathcal{A}$.

The following proposition holds by the construction of C_Π.

Proposition 3. *An ELP program Π has a candidate world view w.r.t. a guess Φ of Π if and only if the* HEX *program $C_\Pi \cup G_\Phi$ is consistent.*

3.3.4 *Guessing Constraints*

The number of answer sets of a guessing program G_Π for an ELP program Π depends on the number of weak modal literals in $\mathrm{Ep}(\Pi)$, that is, for n modal literals there are 2^n answer sets each encoding a different guess of Π. We define two checking programs that, when G_Π is extended with either of them, put constraints on the guessing. The first checking program restricts the cardinality of generated guesses depending on the current evaluation level. The second checking program refers to a collection Ω of guesses representing the world views already found. The constraints in this program prevent guesses from being generated that are not maximal relative to the guesses in Ω.

In both checking programs, for simplicity, we express the constraints using *aggregate atoms* of the form $\#\mathrm{count}\ T = k$, where T is a set of terms and k is an integer. Intuitively, such an atom evaluates to true if the number of terms in T equals k.[2]

Definition 21 (Cardinality checking program)**.** Let Π be an ELP program. The *cardinality checking program* L_k of evaluation level $k, 0 \leq k \leq |\mathrm{Ep}(\Pi)|$, is the program

$$L_k := \{\leftarrow \mathit{not}\ \#\mathrm{count}\{X : \mathit{guess}(X)\} = k\}, \tag{3.23}$$

such that, for each guess $\Phi \subseteq \mathrm{Ep}(\Pi)$ and guess encoding G_Φ, the program $L_k \cup G_\Phi$ is consistent exactly when Φ has cardinality k.

Definition 22 (Subset checking program)**.** Let Π be an ELP program and let Ω be a collection of guesses of Π. First, define the set F_Ω of facts encoding all guesses in Ω:

$$F_\Omega := \bigcup_{\Phi \in \Omega} \{\mathit{member}(\rho, \Phi) \mid \rho \in \Phi\} \tag{3.24}$$

Then, assuming this encoding, define the generic subset checking constraint:

$$r_\subset := \ \leftarrow \#\mathrm{count}\{X : \mathit{guess}(X), \mathit{not}\ \mathit{member}(X, Y)\} = 0, \mathit{member}(\_, Y). \tag{3.25}$$

Finally, define the *subset checking program* S_Ω relative to Ω

$$S_\Omega := F_\Omega \cup \{r_\subset\} \tag{3.26}$$

such that for all $\Phi \subseteq \mathrm{Ep}(\Pi)$, the program $S_\Omega \cup G_\Phi$ is consistent exactly when Φ is not a subset of any guess in Ω.

[2] For brevity, our syntax definition in Section 3.1.1 does not include aggregate atoms. They are described in more detail in, e.g., (Faber et al., 2008, 2010; Calimeri et al., 2012).

Algorithm 1: Basic solver for ELP programs

Input: An epistemic logic program Π

Output: A stream of tuples of the form (Φ, A), where Φ is a guess of Π and A is an answer set of the world view $\mathcal{A} = \mathrm{AS}(\Pi^{\Phi})$

```
Basic-Ehex($\Pi$)
 1  generate $\mathring{\Pi}$                          ▷ generic epistemic reduct (Def. 17)
 2  generate $\mathrm{G}_{\Pi}$                        ▷ guessing program (Def. 19)
 3  generate $\mathrm{C}_{\Pi}$                        ▷ consistency checking (Def. 20)
 4  $\Omega \leftarrow \varnothing$
 5  for $k \leftarrow |\mathrm{Ep}(\Pi)|$ downto $0$:
 6      generate $\mathrm{L}_k$                        ▷ cardinality checking (Def. 21)
 7      generate $\mathrm{S}_{\Omega}$                 ▷ subset checking (Def. 22)
 8      $\mathring{\Pi}_k^{\Omega} \leftarrow \mathring{\Pi} \cup \mathrm{G}_{\Pi} \cup \mathrm{C}_{\Pi} \cup \mathrm{L}_k \cup \mathrm{S}_{\Omega}$
 9      foreach $A' \in \mathrm{AS}(\mathring{\Pi}_k^{\Omega})$:
10          $\Phi \leftarrow \Phi(A')$                 ▷ decode $\Phi$ from $A'$
11          $\Omega \leftarrow \Omega \cup \{\Phi\}$   ▷ collect $\Phi$
12          $A \leftarrow A' \setminus \mathrm{Aux}(A')$  ▷ remove auxiliary atoms
13          print $(\Phi, A)$
```

The subset constraint $r_{\subset}$ in S_{Ω} under an interpretation I means: if $|\Phi(I) \setminus \Phi| = 0$ holds for some $\Phi \in \Omega$, then reject I; or equivalently, if $\Phi(I) \subseteq \Phi$ holds for some $\Phi \in \Omega$, then reject I.

With these building blocks at hand, we are now able to construct an algorithm that solves the world view enumeration problem for ELP programs.

3.4 The Ehex Algorithm

The Basic-Ehex procedure shown in Algorithm 1 represents the Ehex algorithm without optimizations.[3] Given an ELP program Π as input, the algorithm iterates over *evaluation levels* in descending order. Evaluation levels correspond to cardinalities of epistemic guesses. At each evaluation level k, $0 \le k \le |\mathrm{Ep}(\Pi)|$, guesses $\Phi \subseteq \mathrm{Ep}(\Pi)$ with cardinality $|\Phi| = k$ are considered. To this end, the algorithm generates *level programs* $\mathring{\Pi}_k^{\Omega}$ of Π relative to k and Ω, where Ω is a collection of guesses of Π representing world views already found at levels greater than k.

A level program $\mathring{\Pi}_k^{\Omega}$ can be broken down into a *shared part* and a *level-specific part*. The shared part consists of: the generic epistemic reduct $\mathring{\Pi}$, the guessing program G_{Π}, and the consistency checking program C_{Π}; the *level-specific part* consists of the cardinality checking program L_k and the subset checking program S_{Ω}.

The programs of the shared part are generated at lines 1–3 and the programs of the level-specific part are generated at lines 6–7. The outer loop starting at line 5

[3] Optimizations of the algorithm are discussed in Section 3.6.

iterates over the evaluation levels k in descending order, and the inner loop starting at line 9 iterates over all answer sets of $\mathring{\Pi}_k^{\Omega}$. To get a sense of these answer sets, we first look at answer sets of simpler programs.

For a guess Φ of Π, the program $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$, where $\mathring{\Pi}$ is the generic epistemic reduct of Π and G_{Φ} the guess encoding of Φ, is aux-equivalent to the epistemic reduct Π^{Φ} of Π; this means that Π^{Φ} has the same answer sets as $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ minus auxiliary atoms. Extending $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ with the checking program C_{Π} ensures the consistency of guesses of Π. Thus, if $\mathcal{A}$ is a candidate world view of Π w.r.t. Φ, then $\mathcal{A}$ is also the collection of all answer sets of $\mathring{\Pi} \cup \mathrm{G}_{\Phi} \cup \mathrm{C}_{\Pi}$ minus auxiliary atoms. Now, substituting the guess encoding G_{Φ} for the guessing program G_{Π}, we obtain the shared part $\mathring{\Pi} \cup \mathrm{G}_{\Pi} \cup \mathrm{C}_{\Pi}$ of the level program. If Π is consistent, then each answer set of the shared part encodes both a guess Φ of Π and an answer set belonging to the candidate world view of Π with respect to Φ.

The level-specific part is needed to ensure the maximality condition of the epistemic semantics; it consists of the cardinality checking program L_k and the subset checking program S_{Ω}. Together with the shared part, we obtain the level program

$$\mathring{\Pi}_k^{\Omega} = \underbrace{\mathring{\Pi} \cup \mathrm{G}_{\Pi} \cup \mathrm{C}_{\Pi}}_{\text{shared part}} \cup \underbrace{\mathrm{L}_k \cup \mathrm{S}_{\Omega}}_{\text{level-specific part}}. \tag{3.27}$$

Remark 6. Loosely following the “Guess&Check” paradigm (Eiter et al., 2000), the level program can also be divided into a *guessing part* and a *checking part*,

$$\mathring{\Pi}_k^{\Omega} = \underbrace{\mathring{\Pi} \cup \mathrm{G}_{\Pi}}_{\text{guessing part}} \cup \underbrace{\mathrm{C}_{\Pi} \cup \mathrm{L}_k \cup \mathrm{S}_{\Omega}}_{\text{checking part}}, \tag{3.28}$$

where the answer sets of the guessing part encode all answer sets of all consistent epistemic reducts of Π, and the checking part tests whether the answer sets represent world views with respect to guesses of cardinality k.

All guesses of Π encoded in the answer sets of $\mathring{\Pi}_k^{\Omega}$ are maximal and have cardinality k only if Ω contains all maximal guesses with cardinality greater than k. In the algorithm, this is ensured by iterating over the values of k in descending order.

After initializing Ω as the empty set at line 4, the algorithm iterates from the largest to the smallest guess cardinality k, i.e., at line 5, it assigns values ranging from $|\mathrm{Ep}(\Pi)|$ to 0 to the variable k in descending order. If Ω is empty, then any guess Φ decoded at line 10 is maximal and is added to Ω at line 11. At subsequent evaluation levels, k is decremented and Ω contains Φ, so guesses Φ' that are a subset of Φ are not produced as L_k and S_{Ω} are regenerated. As a result, at no time are there two guesses Φ_1, Φ_2 in Ω such that $\Phi_1 \subset \Phi_2$ and every Φ in Ω is a maximal guess representing a world view.

If $\mathring{\Pi}_k^{\Omega}$ is inconsistent for all possible values of k and Ω, then no world view exists. But if $\mathring{\Pi}_k^{\Omega}$ is consistent for some k, $0 \leq k \leq \mathrm{Ep}(\Pi)$, and $\Omega = \varnothing$, then a world view exists and the procedure enters the body of the inner loop starting at line 10. Each answer set A' of $\mathring{\Pi}_k^{\Omega}$ encodes both a guess Φ of Π and an answer set A of Π^{Φ}. In the procedure, Φ is obtained from A' at line 10 by extracting all modals ρ encoded with $guess(\rho)$ atoms in A', and A is obtained from A' at line 11 by removing all auxiliary atoms in A'. Finally, the procedure outputs the pair (Φ, A) at line 13.

Example 13 (Applying the algorithm)**.** We now apply the Ehex algorithm to the program Π_3 from Example 4:

$$\begin{array}{lrcll}
\Pi_3: & \mathrm{a} & \leftarrow & K\,not\,\mathrm{b}. & r_1\\
 & \mathrm{b} & \leftarrow & K\,not\,\mathrm{a}. & r_2
\end{array}$$

The Basic-Ehex procedure starts with generating the generic epistemic reduct of Π_3 on the first line:

$$\begin{array}{lrcll}
\mathring{\Pi}_3: & \mathrm{a} & \leftarrow & not\,true(M\,\mathrm{b}). & \mathring{r}_1\\
 & \mathrm{b} & \leftarrow & not\,true(M\,\mathrm{a}). & \mathring{r}_2\\
 & true(M\,\mathrm{a}) & \leftarrow & guess(M\,\mathrm{a}). & r_3\\
 & true(M\,\mathrm{a}) & \leftarrow & \mathrm{a}, \neg guess(M\,\mathrm{a}). & r_4\\
 & true(M\,\mathrm{b}) & \leftarrow & guess(M\,\mathrm{b}). & r_5\\
 & true(M\,\mathrm{b}) & \leftarrow & \mathrm{b}, \neg guess(M\,\mathrm{b}). & r_6
\end{array}$$

On line 2, it generates get the guessing program of Π_3, which consists of guessing rules for the weak modal literals $M\,\mathrm{a}, M\,\mathrm{b} \in \mathrm{Ep}(\Pi_3)$:

$$\begin{array}{lcl}
\mathrm{G}_{\Pi_3}: & guess(M\,\mathrm{a}) \lor \neg guess(M\,\mathrm{a}). & r_1\\
 & guess(M\,\mathrm{b}) \lor \neg guess(M\,\mathrm{b}). & r_2
\end{array}$$

On line 3, it generates the checking program of Π_3, which consists of input declarations and modal consistency constraints:

$$\begin{array}{lrcll}
\mathrm{C}_{\Pi_3}: & input(guess(M\,\mathrm{a})) & \leftarrow & guess(M\,\mathrm{a}). & r_1\\
 & input(\neg guess(M\,\mathrm{a})) & \leftarrow & \neg guess(M\,\mathrm{a}). & r_2\\
 & input(guess(M\,\mathrm{b})) & \leftarrow & guess(M\,\mathrm{b}). & r_3\\
 & input(\neg guess(M\,\mathrm{b})) & \leftarrow & \neg guess(M\,\mathrm{b}). & r_4\\
 & & \leftarrow & guess(M\,\mathrm{a}), not\,\&b[\mathring{\Pi}_3, input, \mathrm{a}]. & r_5\\
 & & \leftarrow & \neg guess(M\,\mathrm{a}), \&b[\mathring{\Pi}_3, input, \mathrm{a}]. & r_6\\
 & & \leftarrow & guess(M\,\mathrm{b}), not\,\&b[\mathring{\Pi}_3, input, \mathrm{b}]. & r_7\\
 & & \leftarrow & \neg guess(M\,\mathrm{b}), \&b[\mathring{\Pi}_3, input, \mathrm{b}]. & r_8
\end{array}$$

On line 4, the procedure initializes Ω as the empty set, and on line 5, it enters the outer loop iterating over the evaluation levels k from $|\mathrm{Ep}(\Pi_3)| = 2$ down to 0.

Evaluation level $k = 2$, $\Omega = \emptyset$. On line 6, the procedure generates the cardinality checking program L_2, which has the effect that only the guess $\Phi_1 = \{M\,\mathrm{a}, M\,\mathrm{b}\}$ of cardinality 2 will be considered. On line 7, the procedure generates the subset checking program S_Ω, which has no effect as Ω is currently empty. On line 8, the generated building blocks are combined into the level program $(\mathring{\Pi}_3)^\Omega_2 = \mathring{\Pi}_3 \cup \mathrm{G}_{\Pi_3} \cup \mathrm{C}_{\Pi_3} \cup \mathrm{L}_2 \cup \mathrm{S}_\Omega$, which has no answer sets because the single answer set of $\mathring{\Pi}_3 \cup \mathrm{G}_{\Pi_3} \cup \mathrm{L}_2$ does not satisfy C_{Π_3},[4] or equivalently, because $M\,\mathrm{a}$ and $M\,\mathrm{b}$ are true in Φ_1 but false under the collection $\{\emptyset\}$ of all answer sets of the epistemic reduct $\Pi_3^{\Phi_1} = \emptyset$. No worldviews were found at level 2.

Evaluation level $k = 1$, $\Omega = \emptyset$. On line 6 on the second iteration, the procedure generates the cardinality checking program L_1, which has the effect that both guesses $\Phi_2 = \{M\,\mathrm{a}\}$ and $\Phi_3 = \{M\,\mathrm{b}\}$ of cardinality 1 will be considered. As on the previous level 2, the subset checking program S_Ω has no effect because Ω is still empty. On line 8, the generated building blocks are combined into the level program $(\mathring{\Pi}_3)^\Omega_1 = \mathring{\Pi}_3 \cup \mathrm{G}_{\Pi_3} \cup \mathrm{C}_{\Pi_3} \cup \mathrm{L}_1 \cup \mathrm{S}_\Omega$. This time $(\mathring{\Pi}_3)^\Omega_1$ is consistent since any answer set of $\mathring{\Pi}_3 \cup \mathrm{G}_{\Pi_3} \cup \mathrm{L}_1$ also satisfies C_{Π_3}. The procedure enters the inner loop iterating over the two answer sets of $(\mathring{\Pi}_3)^\Omega_1$:

$$A'_1 = \left\{\begin{array}{l}\mathrm{a}, \mathit{true}(M\,\mathrm{a}), \mathit{guess}(M\,\mathrm{a}), \neg\mathit{guess}(M\,\mathrm{b}),\\ \mathit{input}(\mathit{guess}(M\,\mathrm{a})), \mathit{input}(\neg\mathit{guess}(M\,\mathrm{b}))\end{array}\right\} \tag{3.29}$$

$$A'_2 = \left\{\begin{array}{l}\mathrm{b}, \mathit{true}(M\,\mathrm{b}), \mathit{guess}(M\,\mathrm{b}), \neg\mathit{guess}(M\,\mathrm{a}),\\ \mathit{input}(\mathit{guess}(M\,\mathrm{b})), \mathit{input}(\neg\mathit{guess}(M\,\mathrm{a}))\end{array}\right\} \tag{3.30}$$

On line 13 under A'_1, the output is (Φ_2, A_1) where $\Phi_2 = \Phi(A'_1)$ and $A_1 = A'_1 \setminus \mathrm{Aux}(A'_1) = \{\mathrm{a}\}$, and analogously under A'_2 the outputs is $(\Phi_3, A_2) = (\Phi(A'_2), \{\mathrm{b}\})$. Indeed the whole output corresponds to world views of Π_3, namely $\mathcal{A}_1 = \{\{\mathrm{a}\}\}$ w.r.t. Φ_2 and $\mathcal{A}_2 = \{\{\mathrm{b}\}\}$ w.r.t. Φ_3. Two world views were found at level 1.

Evaluation level $k = 0$, $\Omega = \{\Phi_2, \Phi_3\}$. On line 6 on the third and last iteration, the procedure generates the cardinality checking program L_0, which has the effect that only the guess $\Phi_4 = \emptyset$ of cardinality 0 will be considered. Now Ω is not empty. With the subset checking program S_Ω generated on line 7, the program $(\mathring{\Pi}_3)^\Omega_0$ does not generate any answer sets on line 9 since Φ_4 is the empty set and thus a subset of every other guess. No world views were found at level 0.

The procedure ends after iterating the last evaluation level. It has found both world views of Π_3 at level 1.

3.5 Correctness of the EHEX Algorithm

In the following sections, we show the correctness of the EHEX algorithm by analyzing the BASIC-EHEX procedure. The procedure is correct if it is both sound and

[4]For more details, please refer to Section 3.1.2 on HEX semantics.

complete. It is sound if, on input Π, every output (Φ, A) is associated with a world view $\mathcal{A}$ of Π w.r.t. Φ and $A \in \mathcal{A}$; and complete if, for any world view $\mathcal{A}$ of Π w.r.t. Φ, the procedure outputs (Φ, A) with input Π for each $A \in \mathcal{A} = \mathrm{AS}(\Pi^{\Phi})$.

The following lemma explicitly states the aux-equivalence between the generic epistemic reduct extended with a guess encoding and the epistemic reduct.

Lemma 1 (Epistemic reduct aux-equivalence)**.** *Let Π be an ELP program and let $\Phi \subseteq \mathrm{Ep}(\Pi)$ be a guess of Π. Then A' is an answer set of $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$, where $\mathring{\Pi}$ is the generic epistemic reduct of Π and G_{Φ} is the guess encoding of Φ, if and only if $A = A' \setminus \mathrm{Aux}(A')$ is an answer set of the epistemic reduct Π^{Φ} of Π w.r.t. Φ.*

Proof. In this proof, we simplify $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ into a ground program $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$ w.r.t. G_{Φ} such that $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$ is free of auxiliary atoms and both have the same answer sets minus auxiliary atoms.

Let Π be an ELP program and let $\Phi \subseteq \mathrm{Ep}(\Pi)$ be a guess of Π. Define $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$ as the program obtained from $\mathrm{ground}(\mathring{\Pi})$ w.r.t. G_{Φ}, where $\mathring{\Pi}$ is the generic epistemic reduct of Π and G_{Φ} is the guess encoding of Φ, by the following transformation.

1. Apply for each pair (r, ℓ), where r is a rule in $\mathrm{ground}(\mathring{\Pi})$ and ℓ is a substitution literal from Table 3.1 that occurs in r, the transformation rules of Table 3.2.
2. Remove all substitution rules from Table 3.1 occurring in $\mathrm{ground}(\mathring{\Pi})$.

$\ell \in \mathrm{Body}(r)$	ρ	If $guess(\rho) \in \mathrm{G}_{\Phi}$	If $\neg guess(\rho) \in \mathrm{G}_{\Phi}$
$true(M\,a)$	$M\,a$	remove ℓ	replace ℓ with a
$not\ true(M\,a)$	$M\,a$	remove r	replace ℓ with $not\ a$
$true(K\,a)$	$M\ not\ a$	remove r	replace ℓ with a
$not\ true(K\,a)$	$M\ not\ a$	remove ℓ	replace ℓ with $not\ a$

Table 3.2: Transformation rules of $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$

Then $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$ contains no auxiliary atoms and has the same answer sets as $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ minus auxiliary atoms. Observe that $\mathring{\Pi}^{\mathrm{G}_{\Phi}}$ is exactly the epistemic reduct Π^{Φ} of Π w.r.t. Φ. We conclude that $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ and Π^{Φ} are aux-equivalent. □

The following proposition intuitively states that a level program $\mathring{\Pi}_k^{\Omega}$ is consistent exactly when there is a guess Φ with $|\Phi| = k$ such that the program $\mathring{\Pi}_{\Phi}^{\Omega}$ resulting from $\mathring{\Pi}_k^{\Omega}$ by substituting G_{Π} for G_{Φ} is consistent. The subsequent proof uses the Splitting Theorem for HEX programs given in (Eiter et al., 2016, Theorem 1).

Proposition 4. *Let $\mathring{\Pi}_k^{\Omega}$ be a level program of a program Π, where Ω is a collection of guesses of Π with cardinality greater than k, $0 \leq k \leq |\mathrm{Ep}(\Pi)|$. Then $\mathring{\Pi}_k^{\Omega}$ is consistent* *if and only if there is a guess Φ with $|\Phi| = k$ such that by substituting G_Π for G_Φ also the program*

$$\mathring{\Pi}^{\Omega}_{\Phi} := \mathring{\Pi}^{\Omega}_{k} \setminus \mathrm{G}_\Pi \cup \mathrm{G}_\Phi = \mathring{\Pi} \cup \mathrm{G}_\Phi \cup \mathrm{C}_\Pi \cup \mathrm{L}_k \cup \mathrm{S}_\Omega \tag{3.31}$$

is consistent.

Proof. Note that no rule in $\mathrm{G}_\Pi \cup \mathrm{L}_k$ depends[5] on rules in $\mathring{\Pi}^{\Omega}_{k} \setminus (\mathrm{G}_\Pi \cup \mathrm{L}_k)$. Therefore, according to the Splitting Theorem with $\mathrm{G}_\Pi \cup \mathrm{L}_k$ as the rule splitting set of $\mathring{\Pi}^{\Omega}_{k}$,

$$A \in \mathrm{AS}(\mathring{\Pi}^{\Omega}_{k}) \quad \text{iff} \quad A \in \mathrm{AS}(\mathring{\Pi}^{\Omega}_{k} \setminus \mathrm{G}_\Pi \cup \mathrm{G}_\Phi) \tag{3.32}$$

with $\mathrm{G}_\Phi \in \mathrm{AS}(\mathrm{G}_\Pi \cup \mathrm{L}_k)$ where $|\Phi| = k$ due to the cardinality checking program L_k. □

3.5.1 *Soundness of the Algorithm*

Assume BASIC-EHEX(Π) outputs (Φ, A) on line 13. Then there is a collection Ω of guesses of Π such that the level program

$$\mathring{\Pi}^{\Omega}_{k} = \mathring{\Pi} \cup \mathrm{G}_\Pi \cup \mathrm{C}_\Pi \cup \mathrm{L}_k \cup \mathrm{S}_\Omega \tag{3.33}$$

is consistent at evaluation level $k = |\Phi|$. Some rules in the building blocks $\mathring{\Pi}$, C_Π, L_k, and S_Ω depend on facts in G_Φ but not vice versa (cf. Section 3.3). Moreover, these building blocks do not dependent on each other, as shown in Figure 3.1. Thus, since $\mathring{\Pi}^{\Omega}_{\Phi}$ is consistent by Proposition 4, the subprograms $\mathring{\Pi} \cup \mathrm{G}_\Phi$, $\mathrm{G}_\Phi \cup \mathrm{C}_\Pi$, and $\mathrm{G}_\Phi \cup \mathrm{S}_\Omega$ are also consistent.

$$\begin{tikzcd}
 & & \mathrm{S}_\Omega \arrow[dl] \\
\mathring{\Pi} \arrow[r] & \mathrm{G}_\Phi & \mathrm{C}_\Pi \arrow[l] \\
 & & \mathrm{L}_k \arrow[ul]
\end{tikzcd}$$

Figure 3.1: Rule dependencies between building blocks of $\mathring{\Pi}^{\Omega}_{\Phi}$: an arrow from one building block B_1 to another B_2 indicates that some rules in B_1 depend on rules in B_2.

Let (Φ, A) be an output of BASIC-EHEX on input Π, where the sets $\Phi = \Phi(A')$ and $A = A' \setminus \mathrm{Aux}(A')$ were computed at evaluation level k from an answer set A' of $\mathring{\Pi}^{\Omega}_{k}$. Then Φ is an epistemic guess of Π by the construction of G_Π. Further let $\mathcal{A}$ be the collection of all answer sets of the epistemic reduct Π^{Φ} of Π w.r.t. Φ

[5]Rules with an empty body such as all rules in G_Π do not depend on other rules; the one rule in L_k depends only on the rules in G_Π. For a detailed description and associated definitions of rule dependencies, please refer to (Eiter et al., 2016).

Claim 1. The epistemic reduct Π^{Φ} of Π w.r.t. Φ is consistent. This means that $\mathcal{A}$ is nonempty.

Proof. As stated above, if the procedure outputs (Φ, A) on input Π, then $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ is consistent. Thus, by Lemma 1, also Π^{Φ} is consistent and $A \in \mathrm{AS}(\Pi^{\Phi})$. Hence, $\mathcal{A}$ is not empty. □

Claim 2. The guess Φ is consistent with $\mathcal{A}$. This means that $\mathcal{A}$ is a candidate world view of Π w.r.t. Φ.

Proof. Since $\mathrm{G}_{\Phi} \cup \mathrm{C}_{\Pi}$ is consistent, the claim follows from Proposition 3. □

Claim 3. The guess Φ is maximal. This means that $\mathcal{A}$ is a world view of Π w.r.t. Φ.

Proof. Assume Φ is not maximal although $\mathrm{G}_{\Phi} \cup \mathrm{S}_{\Omega}$ is consistent for Ω at evaluation level $|\Phi|$. Then there is a world view of Π w.r.t. Φ' such that $\Phi' \supset \Phi$ and Φ' is not in Ω. If Φ' is not in Ω at evaluation level $|\Phi|$ and Φ' is maximal, that means that $\mathring{\Pi} \cup \mathrm{G}_{\Phi'} \cup \mathrm{C}_{\Pi}$ is not consistent at evaluation level $|\Phi'|$; otherwise the algorithm would have added Φ' to Ω as it iterates over the evaluation levels in descending order and $|\Phi'| > |\Phi|$. But then either $\mathring{\Pi} \cup \mathrm{G}_{\Phi'}$ or $\mathrm{G}_{\Phi'} \cup \mathrm{C}_{\Pi}$ is inconsistent. This contradicts that Φ' represents a world view of Π and therefore Φ is a maximal guess. □

The Claims 1 and 2 say that $\mathcal{A}$ is a candidate world view of Π, and Claim 3 says that $\mathcal{A}$ is an actual world view of Π. By proving these claims, we have shown that the output (Φ, A) is associated with a world view $\mathcal{A}$ of Π w.r.t. Φ and $A \in \mathcal{A}$. This means that Basic-Ehex is sound.

3.5.2 *Completeness of the Algorithm*

Assume $\mathcal{A}$ is a world view of Π w.r.t. $\Phi \subseteq \mathrm{Ep}(\Pi)$ and $A \in \mathcal{A}$, then the Basic-Ehex procedure outputs (Φ, A). Then the epistemic reduct Π^{Φ} is consistent and thus, by Lemma 1, also $\mathring{\Pi} \cup \mathrm{G}_{\Phi}$ is consistent. As Φ is consistent with $\mathcal{A}$, also the consistency checking program C_{Π} extended with the guess encoding G_{Φ} of Φ, i.e., $\mathrm{C}_{\Pi} \cup \mathrm{G}_{\Phi}$, is consistent. Since Φ is maximal and Basic-Ehex is sound, each Φ' collected at line 11 of the procedure in Ω is not a superset of Φ. Thus, at evaluation level $|\Phi|$, also the maximality checking program S_{Ω} extended with G_{Φ}, i.e., $\mathrm{S}_{\Omega} \cup \mathrm{G}_{\Phi}$, is consistent. As a result, the program $\mathring{\Pi}^{\Omega}_{\Phi} = \mathring{\Pi} \cup \mathrm{G}_{\Phi} \cup \mathrm{C}_{\Pi} \cup \mathrm{L}_{|\Phi|} \cup \mathrm{S}_{\Omega}$ is consistent and therefore, by Proposition 4, also $\mathring{\Pi}^{\Omega}_{|\Phi|}$ is consistent. The procedure eventually reaches line 13 at evaluation level $|\Phi|$ where it outputs (Φ, A), because every answer set of $\mathring{\Pi}^{\Omega}_{\Phi}$ is also an answer set of of $\mathring{\Pi}^{\Omega}_{|\Phi|}$. This means that Basic-Ehex is complete.

3.6 Optimizations

In general, the search space for world views of an ELP program Π grows exponentially in the number of ground modal literals of Π, that is, the number of possible epistemic guesses is 2^n, where $n = |\mathrm{Ep}(\Pi)|$. By definition, the set $\mathrm{Ep}(\Pi)$ contains the weak forms of all modal literals that occur in the (potentially large) grounding of Π. However, in some cases it is possible to enumerate all world views of a program with a smaller *effective set* $\mathcal{E} \subseteq \mathrm{Ep}(\Pi)$ of weak modal literals.

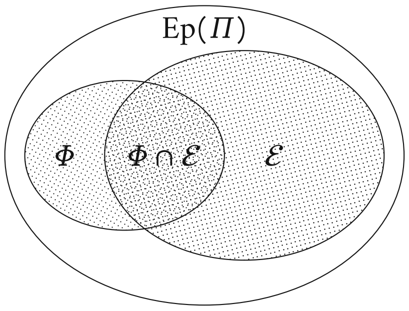

Figure 3.2: Effective sets are a tool to reduce the number of possible truth assignments

In this section, we mainly describe different techniques to compute effective sets. However, we also discuss some other techniques, such as, e.g., exploiting problem-specific properties or skipping evaluation levels due to information from precomputed brave or cautious consequences.

Definition 23 (Effective set of weak modal literals)**.** Let Π be an ELP program. An *effective set* $\mathcal{E}$ (of weak modal literals) of Π is any subset of $\mathrm{Ep}(\Pi)$ such that for all guesses $\Phi^* \subseteq \mathrm{Ep}(\Pi)$ and $\Phi = \Phi^* \cap \mathcal{E}$, the epistemic reducts Π^{Φ^*} and Π^{Φ} both have the same answer sets.

Example 14 (Grounding)**.** The set $\mathrm{Ep}(\Pi_9)$ of the program

$$\Pi_9: \qquad \begin{array}{ll} \mathrm{p}(\mathrm{c}, \text{“s”}, 1). & r_1 \\ \leftarrow K\,\mathrm{q}(X). & r_2 \end{array}$$

is $\{M\,\textit{not}\,\mathrm{q}(\mathrm{c}), M\,\textit{not}\,\mathrm{q}(\text{“s”}), M\,\textit{not}\,\mathrm{q}(1)\}$. Here $|\mathrm{Ep}(\Pi_9)| = 3$, which gives 8 epistemic guesses. This program has one world view $\{\{\mathrm{p}(\mathrm{c}, \text{“s”}, 1)\}\}$ with respect to the guess $\Phi = \mathrm{Ep}(\Pi_9)$. But the smallest effective set of Π_9 is the empty set $\emptyset$, since the epistemic reduct $\Pi_9^{\emptyset}$ of Π_9 with respect to $\emptyset$

$$\Pi_9^{\emptyset}: \qquad \begin{array}{ll} \mathrm{p}(\mathrm{c}, \text{“s”}, 1). & r_1 \\ \leftarrow \mathrm{q}(\mathrm{c}). & r_2 \\ \leftarrow \mathrm{q}(\text{“s”}). & r_3 \\ \leftarrow \mathrm{q}(1). & r_4 \end{array}$$

has the same answer sets as the epistemic reduct $\Pi_9^{\Phi} = \{\mathrm{p(c, ``s", 1)}\}$ of Π_9 with respect to Φ.

The largest effective set of Π is $\mathrm{Ep}(\Pi)$ and the smallest effective set of Π is the intersection of all effective sets of Π. In practice, this means that we can compute all world views of Π using the smallest effective set $\mathcal{E}$ of Π that we can find. To this end we substitute $\mathrm{Ep}(\Pi)$ for $\mathcal{E}$ in the definitions of building blocks and in the algorithm. This works because of the following proposition.

Proposition 5. *Let $\mathcal{E} \subseteq \mathrm{Ep}(\Pi)$ be an effective set of a program Π and a let $\Phi \subseteq \mathcal{E}$ be a guess of Π. Then there is a guess $\Phi^* \subseteq \mathrm{Ep}(\Pi)$ such that $\mathcal{A}_\Phi = \mathrm{AS}(\Pi^{\Phi})$ is a world view of Π w.r.t. Φ^* if the following conditions hold:*

(i) $\mathcal{A}_\Phi$ is nonempty,

(ii) Φ is consistent relative to $\mathcal{E}$, i.e.,
- *every $\rho \in \Phi$ is true in $\mathcal{A}_\Phi$ and*
- *every $\rho \in \mathcal{E} \setminus \Phi$ is false in $\mathcal{A}_\Phi$,*

(iii) Φ is maximal relative to $\mathcal{E}$, i.e., there is no guess $\Phi' \subseteq \mathcal{E}$ with $\Phi' \supset \Phi$, such the conditions (i) and (ii) are satisfied for Φ'.

Proof. Let $\Phi \subseteq \mathcal{E}$ be a guess of Π such that $\mathcal{A}_\Phi$ is nonempty, Φ is consistent relative to $\mathcal{E}$, and Φ is maximal relative to $\mathcal{E}$. Define $\Phi^* \subseteq \mathrm{Ep}(\Pi)$ as the set

$$\Phi^* := \{\rho \in \mathrm{Ep}(\Pi) \mid \mathcal{A}_\Phi \vDash \rho\} \tag{3.34}$$

of all weak modal literals ρ true in $\mathcal{A}_\Phi$. Then $\Phi = \Phi^* \cap \mathcal{E}$ because Φ is consistent and maximal relative to $\mathcal{E}$ and hence, by Definition 23, $\mathcal{A}_\Phi = \mathrm{AS}(\Pi^{\Phi^*})$. So $\mathcal{A}_\Phi$ is a candidate world of Π w.r.t. Φ^* by the construction of Φ^*. Assume Φ^* is not a maximal guess of Π, i.e., there is a larger guess $\Phi' \supset \Phi^*$ such that $\mathrm{AS}(\Pi^{\Phi'})$ is a world view of Π w.r.t. Φ'. Then $\Phi' \cap \mathcal{E} = \Phi$ because Φ is maximal relative to $\mathcal{E}$ and therefore $\mathrm{AS}(\Pi^{\Phi'}) = \mathcal{A}_\Phi$. But then G' is not consistent with $\mathcal{A}_\Phi$ because every ρ true in $\mathcal{A}_\Phi$ is already in Φ^*. It follows that $\mathcal{A}_\Phi$ is a world of Π w.r.t. Φ^*. □

Remark 7. (Caveat) Note that the use of effective sets can eliminate world views that might be considered unintended. For example, consider the following program:

Π_{10}:	$\mathrm{p} \leftarrow M\,\mathrm{q}, \mathit{not}\,\mathrm{q}.$	r_1
	$\mathrm{q} \leftarrow M\,\mathrm{p}, \mathit{not}\,\mathrm{p}.$	r_2
	$\mathrm{z}.$	r_3
	$\mathrm{z} \leftarrow K\,\mathit{not}\,\mathrm{r}.$	r_4
	$\mathrm{r} \vee \neg\mathrm{r} \leftarrow K\,\mathit{not}\,\mathrm{p}.$	r_5

Under EFLP semantics, the program Π_{10} has two world views:

$$\mathcal{A}_1 = \{\{\mathrm{p}, \mathrm{z}\}, \{\mathrm{q}, \mathrm{z}\}\} \text{ w.r.t. } \Phi_1 = \{M\,\mathrm{p}, M\,\mathrm{q}\}$$
$$\mathcal{A}_2 = \{\{\mathrm{r}, \mathrm{z}\}, \{\neg\mathrm{r}, \mathrm{z}\}\} \text{ w.r.t. } \Phi_2 = \{M\,\mathrm{r}\}$$

When the algorithm is run with the effective set $\mathcal{E} = \{M\,\mathrm{p}, M\,\mathrm{q}\}$, only $\mathcal{A}_1$ is found. The reason for this is that $\Phi_2 \cap \mathcal{E}$ is a proper subset of $\Phi_1 \cap \mathcal{E}$ and thus $\Phi_2 \cap \mathcal{E}$ is not maximal relative to $\mathcal{E}$. One could argue that $\mathcal{A}_2$ is an unintended world view because the rule r_4 containing the modal literal $K\,not\,\mathrm{r}$ has no effect on the answer sets of any epistemic reduct and can thus be ignored.

In summary, the truth assignments of weak modal literals that are not in an effective set of Π have no effect effect on the answer sets of epistemic reducts of Π. So we may be able to compute world views of Π much more efficiently by using effective sets.

Below we present two methods for computing effective sets. The first method uses the *positive envelope* of Π, and the second uses an *optimizing ASP grounder*.

As in traditional ASP, ensuring finite grounding of an ELP program Π requires safety restrictions on rules in Π.

Definition 24 (Rule Safety)**.** We say a rule r in an ELP program Π is *safe* if each variable X occurring in r occurs in a positive literal of the form a or $K\,a$, where a is an atom, in the body of r.

Example 15. The rule $\mathrm{p}(X) \leftarrow K\,\mathrm{q}(X)$ is safe while the rule $\mathrm{p}(X) \leftarrow M\,\mathrm{q}(X)$ is not safe. An epistemic reduct of the former rule is safe because either the rule is removed or the modal literal $K\,\mathrm{q}(X)$ is replaced with the positive atom $\mathrm{q}(X)$. An epistemic reducts of the latter rule is not safe because the modal literal $M\,\mathrm{q}(X)$ is either replaced with the positive atom $\mathrm{q}(X)$ or removed from the rule. If it is removed, then X occurs only in the head of the rule in $\mathrm{p}(X)$. Intuitively, if all rules of a program Π are safe, then all rules of epistemic reducts of Π are safe.

Below we assume that all rules of ELP programs are safe.

3.6.1 *Computing a Positive Envelope*

It is possible to compute an effective set of a program Π using a superset of all ground atoms that can be true in the world views of Π. The smaller this superset is, the smaller the effective set can become. The idea of computing a positive envelope is to derive such a superset from a positive version of Π that contains only positive information and is thus monotonic.[6]

In Section 3.1.1 we defined the positive body for rules in HEX syntax. The following definition applies for rules in ELP syntax.

Definition 25 (Positive body of ELP rules)**.** Let r be a rule in ELP syntax of the

[6] If I is a model of a monotonic program $\Pi+$ then any interpretation $I' \supseteq I$ is also a model of Π.

form (2.1). Then the positive body of r, denoted by $\mathrm{Body}^+(r)$, is defined as the set

$$\mathrm{Body}^+(r) \coloneqq \{a \mid a \text{ or } K\,a \text{ is in } \mathrm{Body}(r), \text{ where } a \text{ is an atom}\} \tag{3.35}$$

of atoms a that occur without negation in the body of r or without negation in a strong modal literal in the body of r.

Definition 26 (Positive envelope)**.** Given an ELP program Π, generate a positive program Π^+ from Π so that Π^+ contains no disjunction, no constraints, no modal literals, and no default negated atoms, i.e., define Π^+ as the program consisting of all rules r_a of atoms a occurring in the head of a rule $r \in \Pi$, where

$$r_a \coloneqq a \leftarrow \mathrm{Body}^+(r). \tag{3.36}$$

Then Π^+ is monotonic and the unique answer set A^+_Π of Π^+ is called the *positive envelope* of Π.[7]

Lemma 2. *Let Π be an ELP program and let $\Phi \subseteq \mathrm{Ep}(\Pi)$ be any guess of Π. Then each answer set A of Π^Φ is a subset of the positive envelope A^+_Π of Π.*

Proof. Let A be an answer set of Π^Φ and let $\Pi^+_A \coloneqq \{r_a \in \Pi^+ \mid A \models r_a\}$ be the subset of rules of Π^+ that are satisfied by A. Then A is the unique answer set of Π^+_A because for each atom a in the head of a rule $r \in \Pi^\Phi$ satisfied by A there is a monotonic rule $r_a \in \Pi^+_A$ satisfied by A. Since A^+_Π is the unique answer set of Π^+ and by the monotonicity of both Π^+_A and Π^+ it follows that A is a subset of A^+_Π because Π^+_A is a subset of Π^+. □

To get an effective set of Π, extend the following program M_Π with Π^+ and extract the effective set encoded with *gnd* atoms in the unique answer set of the resulting program.

Proposition 6. *Let Π be an ELP program. Define M_Π as the program consisting of all rules r_φ of modal literals φ occurring in a rule $r \in \Pi$, where*

$$r_\varphi \coloneqq \mathit{gnd}(\mathrm{weak}(\varphi)) \leftarrow \mathrm{Body}^+(r). \tag{3.37}$$

Let A be the unique answer set of $\mathrm{M}_\Pi \cup \Pi^+$. Then $\mathcal{E} = \{\rho \mid \mathit{gnd}(\rho) \in A\}$ is an effective set of Π.

Proof. Let Φ^* be any guess of Π and let $\Phi = \Phi^* \cap \mathcal{E}$. Assume $\mathrm{AS}(\Pi^{\Phi^*}) \neq \mathrm{AS}(\Pi^\Phi)$. Then there is a rule $r \in \Pi$ containing a modal literal φ with $\mathrm{weak}(\varphi) \notin \mathcal{E}$ and

[7] Remember that we view strong negated atoms as atoms with a fresh predicate symbol and an associated constraint. For this reason, an atom a and its syntactical opposite $\neg a$ can both occur in a positive envelope. For example, if $\Pi = \{a \vee \neg a\}$ then $\{a, \neg a\}$ is the unique answer of Π^+, because the implicit constraint $\leftarrow a, \neg a$ is not in Π^+.

an answer set A of either Π^{Φ^*} or Π^{Φ} such that A satisfies all atoms in the positive body of r (otherwise, if no answer set satisfies the positive body of such a rule, then both reducts have the same answer sets). Since A is a subset of the positive envelope A^+_Π by Lemma 2, also A^+_Π satisfies the positive body of r. But then A^+_Π also satisfies $r_\varphi \in M_\Pi$ and thus $\mathrm{weak}(\varphi) \in \mathcal{E}$, which contradicts $\mathrm{weak}(\varphi) \notin \mathcal{E}$. It follows that $\mathrm{AS}(\Pi^{\Phi^*}) = \mathrm{AS}(\Pi^{\Phi})$ for arbitrary Φ^* where $\Phi = \Phi^* \cap \mathcal{E}$ and thus $\mathcal{E}$ is an effective set of Π. □

3.6.2 *Brave and Cautious Consequences*

In Proposition 1 we have made the connection between brave and cautious consequences of a logic program Π under FLP answer set semantics and modal literals over atoms in Π. As before, by $\Pi \models_c a$ we denote that a is a cautious consequence of Π, and by $\Pi \models_b a$ that a is a brave consequence of Π.

Now let Π be an ELP program. Then $\mathring{\Pi} \cup \mathrm{G}_\Pi$ is a logic program under FLP semantics, where $\mathring{\Pi}$ is the generic epistemic reduct of Π and G_Π is the guessing program for Π. Intuitively, the answer sets of $\mathring{\Pi} \cup \mathrm{G}_\Pi$ encode all answer sets of all epistemic reducts of Π.

Proposition 7. *(Properties of consequences of $\mathring{\Pi} \cup \mathrm{G}_\Pi$)*

1. *If a is not a brave consequence of $\mathring{\Pi} \cup \mathrm{G}_\Pi$ then $\mathbf{M}\, a$ is false and $\mathbf{M}\, not\, a$ is true in every world view $\mathcal{A}$ of Π.*
2. *If a is a cautious consequence of $\mathring{\Pi} \cup \mathrm{G}_\Pi$ then $\mathbf{M}\, not\, a$ is false and $\mathbf{M}\, a$ is true in every world view $\mathcal{A}$ of Π.*

With this we aggregate the weak modal literals of Π that are false in any reduct of Π into the set

$$\begin{aligned}\Phi^\perp_\Pi := {} & \{\mathbf{M}\, a \in \mathrm{Ep}(\Pi) \mid \mathring{\Pi} \cup \mathrm{G}_\Pi \not\models_b a\} \\ & \cup \{\mathbf{M}\, not\, a \in \mathrm{Ep}(\Pi) \mid \mathring{\Pi} \cup \mathrm{G}_\Pi \models_c a\}.\end{aligned} \tag{3.38}$$

And analogously, we aggregate the weak modal literals of Π that are known to be true in any reduct of Π into the set

$$\begin{aligned}\Phi^\top_\Pi := {} & \{\mathbf{M}\, not\, a \in \mathrm{Ep}(\Pi) \mid \mathring{\Pi} \cup \mathrm{G}_\Pi \not\models_b a\} \\ & \cup \{\mathbf{M}\, a \in \mathrm{Ep}(\Pi) \mid \mathring{\Pi} \cup \mathrm{G}_\Pi \models_c a\}.\end{aligned} \tag{3.39}$$

The algorithm can now use $\Phi^\perp_\Pi$ and $\Phi^\top_\Pi$ to extend the level programs $\mathring{\Pi}^\Omega_k$, for all possible values of k and Ω, with the set of guessing facts

$$\mathrm{F}_{\mathrm{G}_\Pi} := \{\neg guess(\rho) \mid \rho \in \Phi^\perp_\Pi\} \cup \{guess(\rho) \mid \rho \in \Phi^\top_\Pi\}, \tag{3.40}$$

which reduces guesswork if $\mathrm{F}_{\mathrm{G}_\Pi}$ is not empty. The algorithm can also skip all levels $k < |\Phi^\top_\Pi|$ or $k > |\mathrm{Ep}(\Pi)| - |\Phi^\perp_\Pi|$, since every epistemic guess that represents a world view of Π contains all weak modal literals of $\Phi^\top_\Pi$ and no weak modal literals of $\Phi^\perp_\Pi$.

This technique can be refined with level specific information. We get the sets $\Phi^{\bot}_{\mathit{\Pi},k}$ and $\Phi^{\top}_{\mathit{\Pi},k}$ when we extend $\mathring{\mathit{\Pi}} \cup \mathrm{G}_{\mathit{\Pi}}$ with $\mathrm{L}_k \cup \mathrm{S}_{\Omega}$ in (3.38) and in (3.39), respectively. Then every $\rho \in \Phi^{\bot}_{\mathit{\Pi},k}$ is false and every $\rho \in \Phi^{\top}_{\mathit{\Pi},k}$ is true in any world view $\mathcal{A}$ of $\mathit{\Pi}$ found at evaluation level k. This allows the algorithm to skip the current evaluation level k if $k < |\Phi^{\top}_{\mathit{\Pi},k}|$ or $k > |\mathrm{Ep}(\mathit{\Pi})| - |\Phi^{\bot}_{\mathit{\Pi},k}|$. Otherwise the algorithm can attempt to reduce the search space by extending $\mathring{\mathit{\Pi}}^{\Omega}_{k}$ with the level-specific guessing facts $\mathrm{F}_{\mathrm{G}_{\mathit{\Pi},k}}$, which is defined similarly to $\mathrm{F}_{\mathrm{G}_{\mathit{\Pi}}}$, but using $\Phi^{\top}_{\mathit{\Pi},k}$ and $\Phi^{\bot}_{\mathit{\Pi},k}$.

3.6.3 *Guessing Hints*

We may avoid some epistemic guessing for an ELP program $\mathit{\Pi}$ with the set

$$\mathrm{H}_{\mathit{\Pi}} := \{\leftarrow \neg \mathit{guess}(M\,\ell), \ell \mid M\,\ell \in \mathrm{Ep}(\mathit{\Pi})\}, \tag{3.41}$$

of *guessing hints* for $\mathit{\Pi}$, where ℓ is a standard literal. These constraints are based on the observation that if a literal ℓ is true in some answer set of $\mathring{\mathit{\Pi}} \cup \mathrm{G}_{\Phi}$ with $M\,\ell \notin \Phi$, then $\mathrm{AS}(\mathit{\Pi}^{\Phi})$ is not a world view of $\mathit{\Pi}$ and therefore the answer sets containing ℓ can be killed right away.

In the algorithm, the guessing hints can extend the level programs, which may avoid some computation. Note that guessing hints must not be used with the sub-programs for consistency checking with external atoms.

3.6.4 *Using an ASP Grounder*

Grounding with an optimizing ASP grounder allows us to identify additional weak modal literals in an ELP program $\mathit{\Pi}$ that are not effective.

Example 16. In the following program Π_{11} the third rule is superfluous because its head atom r also occurs in the second rule as fact and therefore any answer set in any world view of Π_{11} must contain r, regardless of whether $K\,\mathrm{p}$ is true or not.

Π_{11}:	$\mathrm{p} \lor \mathrm{q}.$	r_1
	$\mathrm{r}.$	r_2
	$\mathrm{r} \leftarrow K\,\mathrm{p}.$	r_3

But this case is not covered by the other optimizations in this section, i.e., the atom p is in the positive envelope $A^{+}_{\mathit{\Pi}} = \{\mathrm{p}, \mathrm{q}, \mathrm{r}\}$ and it is neither always true nor is it never true in the answer sets of $\mathring{\Pi}_{11} \cup \mathrm{G}_{\mathit{\Pi}}$. Manually, we immediately get the world view $\{\{p, r\}, \{q, r\}\}$ w.r.t. $\{M\,\mathit{not}\,\mathrm{p}\}$ by removing the third rule and observing that $K\,\mathrm{p}$ is false in the resulting answer sets. In the algorithm, the truth value of $M\,\mathit{not}\,\mathrm{p} \subseteq \mathrm{Ep}(\Pi_{11})$ will be guessed and checked.

By an optimizing ASP grounder we mean a grounder (such as *gringo*) that transforms a ground ASP program into a strongly equivalent ground program with a simpler syntactical structure (Gebser et al., 2015; Harrison et al., 2015). For an ASP program Π we denote by $\mathrm{ground_{opt}}(\Pi)$ a transformation of $\mathrm{ground}(\Pi)$ such that $\mathrm{ground_{opt}}(\Pi) \cup \Pi'$ and $\Pi \cup \Pi'$, where Π' is any other ASP program, have the same answer sets.

Given an ELP program Π, its generic epistemic reduct $\mathring{\Pi}$ and its guessing program G_Π are ASP programs. Below we denote $\mathrm{ground_{opt}}(\mathring{\Pi} \cup \mathrm{G}_\Pi)$ by $\mathring{\Pi}_{\mathrm{gnd}}$.

Example 17. An optimizing grounder may transform $\mathring{\Pi}_{11} \cup \mathrm{G}_{\Pi_{11}}$ into the strongly equivalent ground program

$$
\begin{array}{lll}
(\mathring{\Pi}_{11})_{\mathrm{gnd}}: & \mathrm{p} \vee \mathrm{q}. & r_1 \\
 & \mathrm{r}. & r_2 \\
 & \mathit{true}(K\,\mathrm{p}) \leftarrow \mathrm{p}, \neg \mathit{guess}(M\,\mathit{not}\,\mathrm{p}). & r_3 \\
 & \mathit{guess}(M\,\mathit{not}\,\mathrm{p}) \vee \neg \mathit{guess}(M\,\mathit{not}\,\mathrm{p}). & r_4
\end{array}
$$

where the rule $\mathrm{r} \leftarrow \mathit{true}(K\,\mathrm{p})$ has been omitted.

Observe that, by the construction of $\mathring{\Pi}$, if no rule in $\mathring{\Pi}_{\mathrm{gnd}}$ contains the auxiliary atom $\mathit{true}(\varphi)$ in its body, then guessing φ is superfluous. This does not necessarily mean that $\mathrm{weak}(\varphi) \in \mathrm{Ep}(\Pi)$ is not effective; when a grounder determines that $\mathit{true}(\varphi)$ is a fact, it can, e.g., remove $\mathit{true}(\varphi)$ from all rule bodies or remove rules that contain $\mathit{not}\,\mathit{true}(\varphi)$ in their body.[8] But if $\mathit{true}(\varphi)$ occurs neither in the body of a rule nor in a fact, then $\mathrm{weak}(\varphi)$ is not effective. In Example 17 the weak modal literal $M\,\mathit{not}\,\mathrm{p}$ is not effective because $\mathit{true}(K\,\mathrm{p})$ occurs neither in the body of a rule nor in a fact. This is summarized in the following proposition.

Proposition 8. *Let A_{true} be the set of all auxiliary atoms of the form $true(\varphi)$ over modal literals φ occurring in $\mathring{\Pi}_{gnd}$, let B_{true} be the subset of all atoms in A_{true} that occur in the body of a rule $r \in \mathring{\Pi}_{gnd}$, and let F_{true} be the subset of all atoms in A_{true} that occur in facts in $\mathring{\Pi}_{gnd}$. Then $\mathcal{E} := \{\mathrm{weak}(\varphi) \mid true(\varphi) \in B_{true} \cup F_{true}\}$ is an effective set of Π.*

Proof. In this proof, we assume that an optimizing ASP grounder removes an atom a from a rule body only if a occurs in a fact. Then, by the strong equivalence of $\mathring{\Pi} \cup \mathrm{G}_\Pi$ and $\mathring{\Pi}_{\mathrm{gnd}}$ and by the aux-equivalence with the epistemic reduct (Lemma 1), any two epistemic reducts Π^{Φ^*} and Π^{Φ} w.r.t. $\Phi^* \subseteq \mathrm{Ep}(\Pi)$ and $\Phi = \Phi^* \cap \mathcal{E}$, respectively, have the same answer sets. □

3.6.5 *Satisfiability Checking*

The output of an epistemic guess representing a candidate world view is sufficient to testify the satisfiability of an ELP program. The algorithm can be modified to run

[8] Note that facts in $\mathring{\Pi}_{\mathrm{gnd}}$ are also cautious consequences of $\mathring{\Pi} \cup \mathrm{G}_\Pi$. We can take advantage of this as described in Section 3.6.2.

in satisfiability checking mode. In this mode, the algorithm can just output the first guess it finds and exit. It is also possible to remove constraints on the cardinality of guesses, since guesses do not need to be maximal for satisfiability checking; but this may result in higher memory requirements.

Other optimizations outside of the algorithm, such as configuring the underlying HEX solver for satisfiability checking, are also possible.

3.6.6 *Problem-Specific Optimizations*

Sometimes the solutions to a particular problem have properties that we can use to narrow the search space. A well-known example are *conformant planning* problems with incomplete knowledge of the initial states, where each problem instance defines a goal that must be reached by a sequence of actions (i.e., a plan) (Smith and Weld, 1998).

We show optimizations for ELP programs Π that encode planning problem as proposed in (Kahl et al., 2015), where the answer sets in a world view of Π represent plans that reach the goal in exactly n steps, where n is encoded in Π. Such programs encode the condition to reach the goal with the constraints

$$\mathrm{C}_g := \left\{ \begin{array}{l} \leftarrow g, M\, not\, g \\ \leftarrow K\, not\, g \end{array} \right\} \tag{3.42}$$

where g is an atom that represents the goal. These constraints enforce that in every world view of Π both $M\,g$ is true and $M\,not\,g$ is false whenever g is true. The cardinality of guesses of world views of Π is $n+1$ because, by construction, a modal literal of the form $M\,\mathrm{occurs}(a,i)$ must be true at each step $i \in \{0 \ldots n-1\}$ for an action a, and $M\,g$ must be true.

To reduce the guessing in the algorithm, we define the set G_g of guessing facts for g:

$$\mathrm{G}_g := \left\{ \begin{array}{r} \neg guess(M\, not\, g) \leftarrow g \\ guess(M\, g) \leftarrow \end{array} \right\} \tag{3.43}$$

A plan for a problem encoded in Π exists if and only if the program $\mathring{\Pi}_g := \mathring{\Pi}^{\varnothing}_{n+1} \cup \mathrm{G}_g$ is consistent, where $\mathring{\Pi}^{\varnothing}_{n+1}$ is the level program used in the algorithm at level $n+1$. This allows us to change the algorithm so that it switches to *planning mode* for this type of programs, where it evaluates $\mathring{\Pi}_g$ at level $n+1$ and exits.

Implementation

4

In this chapter we present the *ehex* solver, which is an implementation of the EHEX algorithm using the *dlvhex* system.

4.1 The *dlvhex* System

The *dlvhex* system (Eiter et al., 2015) is a reasoner for computing models of HEX programs. It comes with a flexible system architecture allowing authors to extend the reasoner with dedicated external atoms via plugins. The truth of such external atoms is determined by means of an external computation resource.

The following program, which we borrowed from *dlvhex* homepage,[1] illustrates the usage of external atoms:

$$\Pi_{12}: \qquad \mathrm{reached}(X) \leftarrow \&\mathrm{reach}[\mathrm{edge}, \mathrm{a}](X). \qquad r_1$$

The unique answer set of Π_{12} contains ground atoms of the form $\mathrm{reached}(v)$ taking values v from the external predicate &reach. These values are computed via the external atom $\&\mathrm{reach}[\mathrm{edge}, \mathrm{a}]$ and represent all nodes that are reachable from a node labelled “a” in a graph named “edge”. During the evaluation of the program, the task of computing the values is transparently delegated to an external computation source (e.g., a query to a graph database).

Some “real world” examples of plugins that implement external atoms can be found on the *dlvhex* homepage,[1] e.g., the String Plugin, which provides external atoms that allow for common string operations, or the Description Logic Plugin, which provides external atoms that interface with OWL ontologies over HTTP.

Dedicated external atoms can also be used to “call” HEX programs from within the evaluation of another HEX programs and reason about its answer sets (Eiter et al., 2013). In our implementation we use such external atoms provided by the *Nested* HEX *Plugin* for *dlvhex*.

4.1.1 *Programs with Nested Program Calls*

To better understand the external atoms of the *Nested* HEX *Plugin*, we first give a brief overview of the Nested HEX system architecture.

[1] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

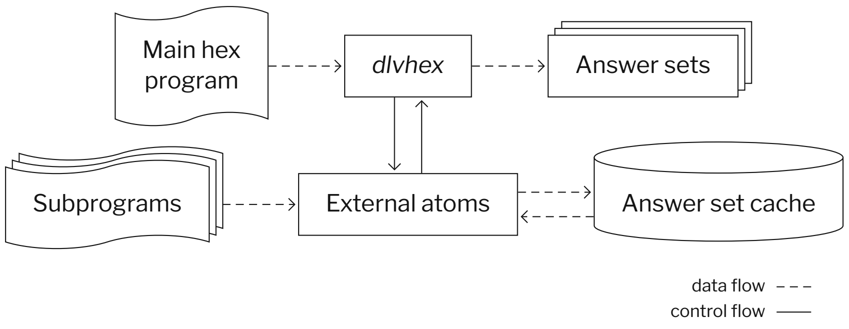

Figure 4.1: Nested HEX system architecture

Nested HEX program calls are realized via plugins for the reasoner *dlvhex*. Such plugins provide a *set of external atoms* and an *answer set cache* for the results.

The general system architecture is pictured in Figure 4.1. A subprogram call corresponds to the evaluation of a special external atom. The program that contains the external atom is called the *calling program* (*host program*) and the subprogram being called via the external atom is called the *callee*. When a subprogram call is encountered in the host program, the plugin creates another instance of the *dlvhex* reasoner to evaluate the subprogram. Its result is then stored in the answer set cache and identified with a unique *handle*, which can later be used to access the cached result. The subprogram is either directly embedded in the host program or it is stored in a separate file on the file system such that it can be included in the host program by filename.

4.1.2 *The Nested HEX Plugin*

The *Nested HEX Plugin*[2] implements nested program calls and provides, among others, the two external atoms &hexBrave and &hexCautious. These external atoms are of particular interest for checking the consistency of epistemic guesses of a program with respect to the answer sets of the corresponding epistemic reducts.

An atom a is a *brave consequence* of a program Π, in symbols $\Pi \models_b a$, if it is in some answer set of Π, or it is a *cautious consequence* of Π, in symbols $\Pi \models_c a$, if it is in all answer sets of Π.

The external atoms allow to query whether an atom a is brave or cautious consequences of an input program Π. A query is specified in terms of a *query predicate* q and a list of output terms $t_1, \ldots, t_n$ such that $a = q(t_1, \ldots, t_n)$ is the atom to check. Additionally a set Δ of facts can be added to Π prior to evaluation via an *input predicate*.

Intuitively, a &hexBrave atom evaluates to true if the specified atom is a brave consequence of $\Pi \cup \Delta$, and likewise, a &hexCautious atom evaluates to true if the

[2]The *Nested HEX Plugin* for *dlvhex* by Christoph Redl is available at `http://www.kr.tuwien.ac.at/research/systems/dlvhex/nestedhexplugin.html` (visited on 2022-07-29)

specified atom is a cautious consequence of $\Pi \cup \Delta$.

Let Π be a HEX program that is to be called from a host program. An external &hexBrave-atom is of the form

$$\&\text{hexBrave}[t, s_\Pi, p, q](c_1, \dots, c_n),$$

with input list t, s_Π, p, q and output list $c_1, \dots, c_n$, $n \geq 0$. The parameters in the input list are:

t	The *type* of the source s_Π where $t \in \{\text{file}, \text{string}\}$.
s_Π	The *source* of the subprogram Π to be evaluated such that s_Π is a string that either contains the source code of Π if $t = \text{string}$, or the path to a file that contains the program if $t = \text{file}$.
p	The *input predicate* that is used to specify *input facts* to be added to the subprogram.
q	A *query predicate* of arity n.

Prior to evaluation, Π is extended with factual knowledge in form of facts that are specified by the input predicate p. The arity of p is the maximum arity m of all facts to be added plus 2. For adding some fact of the form $a(c_1, \dots, c_k)$, the input predicate p is supposed to specify the fact with $p(a, k, c_1, \dots, c_k, \varepsilon, \dots, \varepsilon)$; here ε is a special constant indicating empty terms, and the number of terms of the form ε is $m - k$, i.e., the empty terms fill the remaining term positions in p that are not needed for specifying a certain fact due to its smaller arity.

An external &hexCautious-atom is of the same form, except with a different name.

4.2 The *ehex* Solver

The *ehex* solver is a prototype implementation of the EHEX algorithm extended with the optimizations presented in Section 3.6. The solver takes as input a problem encoded as an ELP program Π and outputs world views of Π that represent solutions to the problem (Figure 4.2). During this process *ehex* generates intermediate ASP or HEX programs, which are evaluated with either *clingo* (Gebser et al., 2014) or *dlvhex* (Eiter et al., 2015). The required *Nested HEX Plugin* for *dlvhex* handles the external atoms that occur in HEX programs (cf. Section 4.1.2).

We implemented the solver as the Python application *ehex*, which is organized into three so-called import packages.[3] The first package *ehex.parser* contains modules for parsing programs and answer sets; the second package *ehex.codegen* contains modules for generating and rendering logic programs; and the third package *ehex.solver* contains the module that implements the algorithm and modules for communicating with *clingo* and *dlvhex* subprocesses. The source code and installing instructions are available on GitHub.[4]

[3] https://packaging.python.org/en/latest/glossary/#term-Import-Package

[4] https://github.com/hexhex/ehex/.

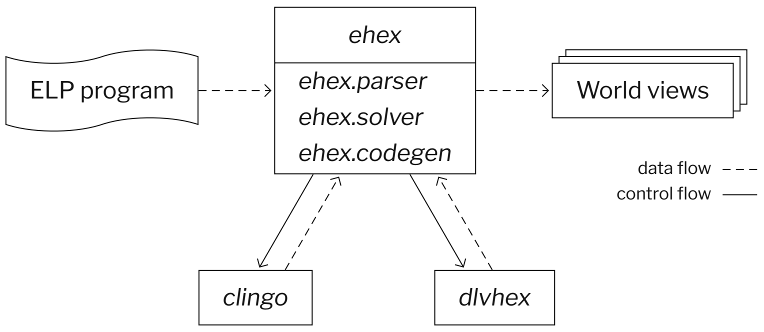

Figure 4.2: Data and control flow of the *ehex* solver

Between reading the input and printing the solutions, the solver performs a series of tasks. First it parses the input program, which results in an abstract syntax tree (AST). Using the AST, it then generates an intermediate logic program that is required for computing the set of all ground modal literals. Tho this end, it opens a *clingo* subprocess, writes the rendered intermediate program to its standard input, and reads the unique answer set from its standard output. The set of all ground modal literals is then obtained from the parsed answer set. In a similar way, the solver generates the level programs required by the algorithm, evaluates them with *dlvhex*, and processes the generated answer sets. This includes user-friendly formatting of the extracted solutions, e.g. grouped by world views, and printing the formatted solutions on the standard output.

4.2.1 *Running the Solver*

The solver can be used as a library or as a command line application. On the command line, *ehex* accepts programs on its standard input:

```
> echo "a :- M a." | ehex
World: 1@1
Modal literals: {M a}
{a}
```

Here we have run the solver with the input program $\Pi = \{a \leftarrow M\, a\}$. The outputs corresponds to the unique world view $\mathcal{A}_{1@1} = \{\{a\}\}$ of Π with respect to the set $\{M\, a\}$ of modal literals that are true in $\mathcal{A}_{1@1}$, found at evaluation level 1.

In the following run we provide the input program `eligible.elp` (Listing 1) as a command line argument:

```
> ehex eligible.elp
World: 1@2
Modals: {M not eligible("Mike"), M not ¬eligible("Mike")}
{fairGPA("Mike"), interview("Mike"), student("Mike")}
{eligible("Mike"), highGPA("Mike"), interview("Mike"), student("Mi
ke")}
```

```
1                     eligible(X) ← highgpa(X), student(X).
2                     eligible(X) ← minority(X), fairGPA(X), student(X).
3                     ¬eligible(X) ← ¬fairGPA(X), ¬highGPA(X), student(X).
4                     interview(X) ← not K eligible(X), not K ¬eligible(X), student(X).
5                     student("Mike").
6                     fairGPA("Mike") | highGPA("Mike").
```

Listing 1: The file `eligible.elp` contains an instance of the *Scholarship Eligibility* problem encoded as ELP program

```
1                     eligible(X) :- highGPA(X), student(X).
2                     eligible(X) :- minority(X), fairGPA(X), student(X).
3                     z_Neg_eligible(X) :- z_Neg_fairGPA(X), z_Neg_highGPA(X), student(X).
4                     interview(X) :- student(X).
5                     student("Mike").
6                     fairGPA("Mike").
7                     highGPA("Mike").
8                     z_Gnd_M_Not_eligible(X) :- student(X).
9                     z_Gnd_M_Not_Neg_eligible(X) :- student(X).
```

Listing 2: The positive envelope program

The second output corresponds to the unique world view $\mathcal{A}_{1@2}$ of `eligible.elp` w.r.t. $\{M\,\mathit{not}\ \text{eligible}(\text{“Mike”}), M\,\mathit{not}\ \neg\text{eligible}(\text{“Mike”})\}$, found at evaluation level 2. During this run, *ehex* parsed the input file and generated intermediate programs, which we discuss now.

In Listing 1, the traditional syntax for modals is used, e.g., $\mathit{not}\,K\,\text{eligible}(X)$ in the body of the rule at line 4. The ELP parser of *ehex* accepts syntax of the form $M\,a$, $K\,a$, $\mathit{not}\,M\,a$, or $\mathit{not}\,K\,a$ for modals over an atom a and equivalent syntax of the form $M\,\ell$ or $K\,\ell$ for modals over a standard literal ℓ.

After parsing the input file, *ehex* computed the set of ground modals of the input program. To this end, the positive envelope program (cf. Section 3.6.1) shown in Listing 2 was generated.

In the generated programs, atoms whose predicate name starts with `z_` represent auxiliary atoms as defined in Notation 1, except atoms whose name start with `z_Neg_`, which represent strong negated atoms. The atom `z_Gnd_M_Not_eligible(X)` at line 9, for example, corresponds to the auxiliary atom $\mathit{gnd}(M\,\mathit{not}\ \text{eligible}(X))$.

After parsing the ground modal literals from the single answer set of the positive envelope program, the shared building blocks (cf. Section 3.3) of the algorithm were generated, namely: the generic epistemic reduct (Listing 3), the guessing program (Listing 4), and the consistency checking program (Listing 5).

The constraints in the lines 11–13 in Listing 3 of the generic epistemic reduct

```
eligible(X) :- highGPA(X), student(X).
eligible(X) :- minority(X), fairGPA(X), student(X).
z_Neg_eligible(X)
:- z_Neg_fairGPA(X), z_Neg_highGPA(X), student(X).
interview(X)
:- not z_True_K_eligible(X), not z_True_K_Neg_eligible(X),
student(X).
student("Mike").
fairGPA("Mike") | highGPA("Mike").

:- eligible(X), z_Neg_eligible(X).
:- fairGPA(X), z_Neg_fairGPA(X).
:- highGPA(X), z_Neg_highGPA(X).

z_True_K_eligible(X)
:- eligible(X), z_Neg_G(z_M_Not_eligible(X)).
z_True_K_Neg_eligible(X)
:- z_Neg_eligible(X), z_Neg_G(z_M_Not_Neg_eligible(X)).
```

Listing 3: The generic epistemic reduct

resulted from rewriting the strong negated atoms that occur in the input program.

The variables in the generated guessing rules at in the lines 1–5 in Listing 4 are bound by grounding atoms. The grounding atoms themselves, which were obtained from the answer set of the positive envelope program, occur in the facts in the lines 7–8. The grounding atoms also occur in the checking program.

The generated checking program in Listing 5 corresponds to Definition 20, but differs from it in that the number of ground rules containing external atoms is reduced by half; we have observed shorter running times with this encoding. In the external atoms, the path `reduct.lp` to the file containing the generic epistemic reduct is specified as a string in the input list, e.g., at line 6.

The level-specific building blocks were generated inside the evaluation loop, i.e., the cardinality checking program and the subset checking program, for each evaluation level . The generated building blocks of evaluation level 1 are shown in Listing 6. The cardinality checking program (cf. Definition 21), which consists of a single constraint, is listed in line 1; the subset checking program (cf. Definition 22) is listed in the lines below, where the epistemic guess of world view $\mathcal{A}_{1@2}$ is encoded in the lines 3–4.

In fact, *ehex* did not enter level 1 and did not generate Listing 6 during this run. After *ehex* found a world view with respect to the largest guess Φ, it stopped searching because all guesses are a subset of Φ and therefore Φ is the only maximal guess.

```
z_G(z_M_Not_Neg_eligible(X)) | z_Neg_G(z_M_Not_Neg_eligible(X))
:- z_Gnd_M_Not_Neg_eligible(X).

z_G(z_M_Not_eligible(X)) | z_Neg_G(z_M_Not_eligible(X))
:- z_Gnd_M_Not_eligible(X).

z_Gnd_M_Not_Neg_eligible("Mike").
z_Gnd_M_Not_eligible("Mike").

:- z_G(X), z_Neg_G(X).
```

Listing 4: The guessing program

```
z_Input(z_G, 1, X) :- z_G(X).
z_Input(z_Neg_G, 1, X) :- z_Neg_G(X).

z_Cautious_Neg_eligible(X)
:- z_Gnd_M_Not_Neg_eligible(X), z_Neg_eligible(X),
&hexCautious[file, "reduct.lp", z_Input, z_Neg_eligible](X).

:- z_G(z_M_Not_Neg_eligible(X)), z_Cautious_Neg_eligible(X).
:- z_Neg_G(z_M_Not_Neg_eligible(X)), not z_Cautious_Neg_eligible(X).

z_Cautious_eligible(X)
:- z_Gnd_M_Not_eligible(X), eligible(X),
&hexCautious[file, "reduct.lp", z_Input, eligible](X).

:- z_G(z_M_Not_eligible(X)), z_Cautious_eligible(X).
:- z_Neg_G(z_M_Not_eligible(X)), not z_Cautious_eligible(X).
```

Listing 5: The checking program

```
:- not #count{M : z_G(M)} = 1.

z_Member(z_M_Not_eligible("Mike"), "world1@2").
z_Member(z_M_Not_Neg_eligible("Mike"), "world1@2").
:- #count{M : z_G(M), not z_Member(M, S)} = 0, z_Member(_, S).
```

Listing 6: Level-specific building blocks of level 1

4.2.2 *Dependencies*

In its current version, *ehex* depends on Python ≥ 3.10[5] and the TatSu grammar compiler ≥ 5.8.[6] It expects recent binaries of *clingo*[7] and *dlvhex*[8] in one of the user's `PATH` directories. Additionally the *Nested* HEX *Plugin*[9] for *dlvhex* must be installed.

[5]`https://www.python.org`
[6]`https://github.com/neogeny/TatSu`
[7]`https://github.com/potassco/clingo`
[8]`https://github.com/hexhex/core`
[9]`https://github.com/hexhex/nestedhexplugin`

Experimental Evaluation

In this chapter, we evaluate our solver *ehex* against the reference solver *ELPsolve* (Kahl et al., 2016) by comparing the running times of the solvers with different problems and different configurations. We use the benchmarks provided by Patrick Kahl together with *ELPsolve*. The benchmarks include instances of the *Scholarship Eligibility* problem (Gelfond, 1991) and instances of the *Yale Shooting* problem (Hanks and McDermott, 1987).

In the next subsection, we briefly describe the solvers and the problem instances. We then present experimental results and conclude with a summary.

5.1 The Solvers

The main difference between *ehex* and *ELPsolve* lies in the approach they take to check epistemic guesses for consistency. Both solvers ensure the maximality condition for epistemic guesses of world views by iterating over *evaluation levels* in descending order; during evaluation, both solvers produce answer sets that encode epistemic guesses. However, while *ELPsolve* produces (a potentially large number of) answer sets of ASP programs that encode epistemic guesses that must be checked for consistency in a post-processing step, *ehex* produces answer sets of HEX programs that encode epistemic guesses that are already consistent.

5.1.1 *ELPsolve*

At a given evaluation level k with input Π, *ELPsolve* generates answer the sets of a special ASP program Π' that encode all answer sets of epistemic reducts of Π and their associated epistemic guesses Φ with cardinality k. In the case of a large number of guesses, the computation of these answer sets is divided such that the resulting groups of answer sets form a partition of all answer sets. This reduces the memory requirements of individual ASP solver calls and enables parallel execution of solving tasks. The answer sets of a group, which now encode a subset of answer sets of epistemic reducts and associated guesses, are then checked for consistency in an subsequent computational step. The number of processors to be used and the size of answer set groups are configurable.

Letter code	Option	Description
c	`--compute-consequences`	Compute brave and cautious consequences
r	`--ground-reduct`	Compute an effective set by grounding the generic epistemic reduct
g	`--guessing-hints`	Add guessing hint rules
p	`--planning-mode`	Enable planning mode

Table 5.1: Letter codes of *ehex* optimization options

5.1.2 *ehex*

At a given evaluation level k with input Π, *ehex* evaluates a special HEX program that contains external atoms to check for consistency of epistemic guesses of Π during a HEX solver call. Also part of this level-specific HEX program are constraints that eliminate all answer sets that do not correspond to world views w.r.t. guesses of of cardinality k (cf. Section 3.4).

The solver implements the optimizations presented in Section 3.6. They are disabled by default and can be enabled with the command line options listed in Table 5.1. In the following we refer to these options by their letter codes; for example, we write *ehex*^{cg} to express that the options `--compute-consequences` and `--guessing-hints` are enabled in *ehex*.

In addition to the optimization options, *ehex* has options to select either FLP or NEX answer set semantics for the epistemic reduct and options to switch into satisfiability checking mode and into planning mode for planning problems. By default, *ehex* assumes the FLP answer set semantics and performs a full search in standard mode for general problems.

5.2 Problem Instances

This section describes the instances of the Scholarship Eligibility problem and the Yale Shooting problem that we used in our experiments. Since *ehex* has its own ELP parser, we adapted the original problem instances to our the syntax.[1] The adapted instances are available on the *ehex* project page.[2]

[1] One goal in designing the parser was to allow the user to enter examples directly from the literature without special directives.

[2] `https://github.com/hexhex/ehex/tree/master/examples`

5.2.1 *The Scholarship Eligibility Problem*

The Scholarship Eligibility problem is a classic problem for epistemic logic programs. It was introduced by Gelfond Lifschitz in the paper *Strong Introspection* (Gelfond, 1991) to describe the problem of "incomplete information in the presence of multiple answer sets". Each instance of the problem is expressed over a number N of students that may be eligible for an interview. We use the labels EN to refer to particular instances of the Scholarship Eligibility problem over N students. These instances share a generic part, but otherwise differ in structure, which means that the difficulty of an instance does not necessarily scale with the number of students.

5.2.2 *The Yale Shooting Problem*

The Yale Shooting problem presented in (Hanks and McDermott, 1987) is a classic planning problem. To solve a Yale Shooting problem, one has to find a plan to reach the goal of shooting a turkey in N steps. In general, more than one plan might be possible. Such a problem can be encoded as an epistemic logic program as described in (Kahl et al., 2015), where the number of steps N is called the *horizon*.

We refer to specific instances of the Yale Shooting problem by the labels YN, where N is the horizon. Instances with different horizons share the same goal and a *conformant planning module*, but otherwise differ in structure, which means that the difficulty of an instance does not necessarily scale with the horizon. In general, instances of planning problem can have multiple initial states; the instances at hand have at most two initial states.

Instances with this encoding have properties that can be exploited for shorter running times. For example, since the solutions to a Yale Shooting problem must reach the goal in a fixed number of steps, there is a fixed correspondence between the horizon of a problem and the cardinality of epistemic guesses of solutions (cf. Section 3.6.6). To take advantage of this and other properties, both solvers can run in planning mode.

5.3 Experimental Results

In this section, we contrast our expectations with experimental results.

5.3.1 *Testing Environment*

The measurements of the running times of both solvers were taken on a ThinkPad X1 series laptop with an Intel Core i7–8550U CPU and 16 GB of RAM. *ELPsolve* was run with the supplied script `elps2`, which configures the solver to use three processors and a group size of 300 for epistemic guesses. Both solvers used *clingo* 5.4, and *ehex* additionally used *dlvhex* 2.5. The measured times include the preprocessing times and the times to output the complete solutions.

5.3.2 *Expected Results*

One advantage of external atoms is that answer sets that do not contribute to world-views need not be held in memory for consistency checking of epistemic guesses. Thus, we expected *ehex* to scale at least as well as *ELPsolve* with the number of epistemic guesses. We also expected that the optimizations implemented in *ehex* would result in shorter running times, since the optimizations aim to reduce the amount of guesswork.

For Yale Shooting problem instances, we expected *ehex* running times in planning mode to be comparable to *ELPsolve* running times in a similar planning mode (Kahl et al., 2016).

We also experimented with the NEX answer set semantics for the epistemic reduct, which is the semantics of programs with nested expression (Lifschitz et al., 1999) and which is the semantics *ELPsolve* assumes. We expected no notable differences between the running times of *ehex* with different semantics.

For the sake of completeness, we also tested *ehex* in satisfiability checking mode, where the solver exits as soon as it has handled the first answer set. We expected *ehex* to always terminate faster in this mode.

5.3.3 *Tables Showing Results*

The results of the evaluation are schown in the Tables 5.2, 5.3, 5.4 and 5.5. Each row of a table contains a problem label P in the first column, which refers to either an instance of the Eligibility problem or the Yale Shooting problem. The second column contains the number 2^n of epistemic guesses of P, where n is the cardinality of the initial effective set computed by *ehex* using the method described in (Section 3.6.1). The other columns contain the wall-clock time of runs with either $ehex^x$, where x is a string of letter codes representing optimizations as in Table 5.1, or with *ELPsolve* under its default configuration.

Values set in bold represent the shortest running time in a row or between two configurations. A dash in a cell indicates that the solver process terminated unexpectedly, e.g., the operating system killed the solver process or the computer crashed.

5.3.4 *Results with the Eligibility Problem*

The results of the Eligibility problem in Table 5.2 confirm our expectation that *ehex* scales well in the number of epistemic guesses. As can be seen in the first column of running times, *ehex* solved the Eligibility problems E1–E16 in about three minutes without any optimizations, whereas *ELPsolve* reached the timeout of 10 minutes with the instances E14 and E16.

The running times of $ehex^{\mathrm{g}}$, $ehex^{\mathrm{r}}$, and $ehex^{\mathrm{gr}}$ correspond to enabling guessing hints, grounding the generic epistemic reduct, and both, respectively. Enabling guessing hints slightly improved the running times with E16. Grounding the generic

Results with Optimizations

P	#	*ehex*	*ehex*^{g}	*ehex*^{r}	*ehex*^{gr}	*ehex*^{c}	*ehex*^{cg}	*ehex*^{cr}	*ehex*^{cgr}	*ELPsolve*
E1	2^{2}	0.19	0.29	0.30	0.31	0.31	0.21	0.21	0.21	**0.10**
E4	2^{8}	0.79	0.87	0.47	0.54	0.24	0.23	0.26	0.26	**0.12**
E8	2^{16}	1.90	2.01	1.10	1.14	**0.25**	**0.25**	0.26	0.26	0.91
E10	2^{20}	4.46	4.62	2.92	3.01	**0.28**	**0.28**	0.29	0.29	6.19
E12	2^{24}	13.82	14.18	11.90	12.16	**0.31**	**0.31**	0.32	0.32	202.45
E14	2^{28}	27.84	27.86	23.91	24.19	**0.32**	**0.32**	0.34	0.33	> 600
E16	2^{32}	136.33	116.34	104.81	104.91	**0.36**	**0.36**	0.37	0.37	> 600
E25	2^{50}	–	–	–	–	**1.32**	1.59	1.33	1.60	–
Y1	2^{4}	0.68	0.50	0.54	0.56	0.32	0.31	0.33	0.32	**0.14**
Y2	2^{6}	0.86	0.70	0.77	0.77	0.51	0.50	0.54	0.51	**0.13**
Y3	2^{8}	1.44	1.11	1.27	1.23	0.89	0.81	0.94	0.81	**0.13**
Y4	2^{10}	2.17	1.49	1.89	1.63	1.06	0.94	1.12	0.96	**0.14**
Y5	2^{17}	16.95	9.71	15.57	10.50	10.37	6.48	10.72	6.63	**0.22**
Y6	2^{20}	80.47	49.35	78.14	52.70	61.26	40.02	62.22	39.67	**0.72**
Y7	2^{23}	412.97	197.61	414.54	201.01	358.04	165.14	356.42	164.98	**3.18**
Y8	2^{34}	> 600	> 600	> 600	> 600	> 600	> 600	> 600	> 600	**151.21**

Table 5.2: Experimental results of comparing *ELPsolve* with *ehex* with different optimizations using instances of the Eligibility problem and the Yale Shooting problem.

epistemic reduct further improved the running times with E12–E16. But the runs with E25 did not complete with either optimization before the operating system terminated the solver process, which is indicated by a dash in the table.

The optimizations enabled by computing brave and cautious consequences (column *ehex*^{c}) resulted in significant shorter running times, as the solver was able to compute the world views of all instances E1–E25 in about three seconds. In combinations with the other optimizations, the running times did not become shorter. Enabling guessing hints actually made the results slightly worse with E25.

5.3.5 *Results with the Yale Shooting Problem*

We measured the running times of *ehex* with the Yale Shooting problem instances in both standard mode for solving general problems and planning mode for solving planning problems. *ELPsolve* seemed to recognize these instances as planning problems, as the solver switched into a corresponding planning mode on these instances by itself. The respective results are shown in Table 5.2 and Table 5.3.

Not surprisingly, the running times in standard mode are consistently longer than they are in planning mode. In standard mode in Table 5.2, *ehex* did not find a solution for Y8 with any combination of optimizations before the timeout of 10 minutes occurred. However, we note that *ehex* solved Y7 in half the time when guessing hints were enabled and that the combination of all optimizations performed best overall.

Results in Planning Mode

P	#	*ehex*$^{\mathrm{p}}$	*ehex*$^{\mathrm{pg}}$	*ehex*$^{\mathrm{pr}}$	*ehex*$^{\mathrm{pgr}}$	*ehex*$^{\mathrm{pc}}$	*ehex*$^{\mathrm{pcg}}$	*ehex*$^{\mathrm{pcr}}$	*ehex*$^{\mathrm{pcgr}}$	*ELPsolve*
Y1	2^4	0.29	0.28	0.29	0.30	0.31	0.31	0.32	0.32	**0.14**
Y2	2^6	0.28	0.29	0.29	0.31	0.31	0.32	0.32	0.32	**0.13**
Y3	2^8	0.29	0.31	0.31	0.32	0.33	0.33	0.34	0.34	**0.13**
Y4	2^{10}	0.31	0.32	0.33	0.33	0.35	0.35	0.36	0.36	**0.14**
Y5	2^{17}	0.38	0.40	0.40	0.41	0.42	0.43	0.44	0.44	**0.22**
Y6	2^{20}	**0.51**	0.53	0.53	0.55	0.56	0.57	0.58	0.58	0.72
Y7	2^{23}	**1.06**	1.09	1.09	1.11	1.14	1.14	1.15	1.15	3.18
Y8	2^{34}	**0.46**	0.48	0.49	0.49	0.52	0.52	0.53	0.54	151.21

Table 5.3: Experimental results of comparing *ELPsolve* with *ehex* in planning mode using instances of the Yale Shooting problem.

In planning mode in Table 5.3, much shorter running times occur in the results; in particular, *ehex* performed better than *ELPsolve* with Y6–Y8. Adding other optimizations had no particular effect on the running times.

5.3.6 *Results with Different Semantics*

In this experiment, *ehex* was configured with different answer set semantics for the epistemic reduct, namely FLP and NEX semantics. This configurations corresponds to the evaluation of ELP programs under EFLP ($\mathrm{SE16}_{\mathrm{FLP}}$) and K16 ($\mathrm{SE16}_{\mathrm{NEX}}$) epistemic semantics, respectively. We discuss epistemic semantics in more detail in Section 6.2.

Table 5.4 shows the results for *ehex* with FLP and NEX semantics under four different configurations.

For Eligibility instances, the results show that running times were not consistently shorter with either semantics. This is in line with our expectation that the differences in running times between the two semantics are negligible.

More surprisingly, on the larger Yale Shooting instances Y5–Y7 in standard mode for general problems but with NEX semantics, running times were either consistently longer or the solver crashed.

5.3.7 *Results with Satisfiability Checking*

In satisfiability checking mode, *ehex* turns off cardinality checking of epistemic guesses, thereby removing the maximality condition of world views, and exits as soon as it finds a consistent guess indicating satisfiability.

Table 5.5 shows the results for *ehex* in full searching (FS) and in satisfiability checking (SC) mode under four different configurations. Note that all optimization strings contain the letter code 'c' because we decided to let *ehex* compute brave and

Semantics Results

		ehex		*ehex*^{g}		*ehex*^{r}		*ehex*^{c}	
P	#	FLP	NEX	FLP	NEX	FLP	NEX	FLP	NEX
E1	2^{2}	0.19	0.19	0.29	**0.19**	0.30	**0.19**	0.31	**0.20**
E4	2^{8}	**0.79**	0.85	0.87	**0.81**	0.47	**0.46**	0.24	**0.23**
E8	2^{16}	**1.90**	1.95	2.01	**1.93**	1.10	**1.07**	0.25	0.25
E10	2^{20}	**4.46**	4.53	4.62	**4.02**	**2.92**	3.25	0.28	**0.27**
E12	2^{24}	**13.82**	14.00	**14.18**	16.40	11.90	**11.63**	0.31	**0.30**
E14	2^{28}	**27.84**	27.85	27.86	**27.62**	23.91	**23.51**	0.32	**0.31**
E16	2^{32}	136.33	**135.75**	116.34	**115.85**	104.81	**104.48**	0.36	0.36
Y1	2^{4}	0.68	**0.56**	**0.50**	0.69	**0.54**	0.62	**0.32**	0.48
Y2	2^{6}	0.86	**0.79**	**0.70**	0.84	**0.77**	0.79	**0.51**	0.61
Y3	2^{8}	1.44	**1.28**	**1.11**	1.30	**1.27**	1.34	**0.89**	1.14
Y4	2^{10}	2.17	**1.94**	**1.49**	1.79	**1.89**	1.93	**1.06**	1.17
Y5	2^{17}	**16.95**	18.47	**9.71**	12.88	**15.57**	19.02	**10.37**	16.17
Y6	2^{20}	**80.47**	125.45	**49.35**	62.77	**78.14**	127.91	**61.26**	118.41
Y7	2^{23}	**412.97**	–	**197.61**	366.04	**414.54**	–	**358.04**	–

Table 5.4: Experimental results of comparing the default FLP semantics with NEX semantics for the epistemic reduct.

Results in Satisfiability Checking mode

		ehex^c		*ehex*^{cg}		*ehex*^{cr}		*ehex*^{cgr}	
P	#	FS	SC	FS	SC	FS	SC	FS	SC
E1	2^{2}	0.31	0.31	0.21	0.21	**0.21**	0.23	**0.21**	0.23
E4	2^{8}	**0.24**	0.25	**0.23**	0.24	**0.26**	0.27	**0.26**	0.27
E8	2^{16}	**0.25**	0.27	**0.25**	0.27	**0.26**	0.27	**0.26**	0.27
E10	2^{20}	**0.28**	0.29	**0.28**	0.29	**0.29**	0.30	**0.29**	0.30
E12	2^{24}	0.31	0.31	0.31	0.31	0.32	0.32	0.32	0.32
E14	2^{28}	0.32	0.32	**0.32**	0.33	0.34	0.34	0.33	0.33
E16	2^{32}	0.36	**0.34**	0.36	**0.35**	0.37	**0.36**	0.37	**0.35**
E25	2^{50}	1.32	**0.55**	1.59	**0.58**	1.33	**0.56**	1.60	**0.56**

Table 5.5: Experimental results of comparing the default full searching mode (FS) with the satisfiability checking mode (SC).

cautious consequences to prune the search space. Without this optimization, the solver on our system would run out of memory or be terminated by the operating system, since the number of epistemic guesses per HEX solver call is not constrained in this mode.

The data show that satisfiability checking mode only made a difference with the largest instance E25, since the running times with the other instances were already short. Different configurations did not seem to make a difference either.

5.4 Summary of the Results

In our results, we found that *ehex* performed better than *ELPsolve* on the largest instances of the Eligibility problem without optimizations. We attribute this to the fact that *ehex* cancels invalid guesses at evaluation time, thereby pruning the search space. The data suggest that the effectiveness of particular optimizations depends on the problem instance. Instances of the Eligibility problem were solvable in shorter time when we let *ehex* ground the epistemic reduct or compute brave and cautious consequences. Instances of the Yale Shooting problem were solvable in less time when we enabled guessing hints and in much less time when we ran the solver in planning mode. Running *ehex* with NEX answer set semantics for the epistemic reduct as opposed to FLP semantics did not affect the running time in a significant way with instances of the Eligibility problem. But with instances of the Yale Shooting problem the running times were longer when using NEX semantics. Finally, in satisfiability checking mode, we saw shorter running times only with the largest problem instance, as running times with other instances were already short.

Conclusion

Below we conclude this thesis with a summary, a discussion on program transformations and semantics, and outlook and issues.

6.1 Summary

Our motivation for this work was to provide an efficient algorithm for the world view enumeration problem of epistemic logic programs (ELP), which extend disjunctive logic programs with the modal operators M and K (Gelfond, 1991). Satisfaction of modal literals occurring in a program, i.e., standard literals qualified with modal operators, is defined relative to a collection of interpretations. The semantics of ELP programs is defined over epistemic reducts (Shen and Eiter, 2016). The epistemic reduct of a given program is obtained by applying transformation rules with respect to an epistemic guess on truth of the modal literals occurring in the program.

In Algorithm 1 we present a novel procedure (Basic-Ehex) to solve the world view enumeration problem of ELP programs based on special logic programs containing *external atoms* (hex programs). The algorithm evaluates an ordered sequence of dynamically generated hex programs whose iterative construction depends on previously obtained solutions. These hex programs consist of the *generic epistemic reduct* (Section 3.3.1) extended with rules that are intended to either guess solutions, restrict the search space, or check solutions. The rules for checking solutions are expressed declaratively by means of external atoms that allow brave and cautious reasoning over answer sets of subprograms. The algorithm is sound an complete and it is possible to optimize it with different pruning techniques.

A prototype implementation of the algorithm with optional optimizations is provided in form of the *ehex* solver, which is a configurable Python application that depends on *dlvhex* and *clingo* for reasoning tasks. The solver accepts an ELP program as input, for which it can output all solutions, provided there are sufficient resources. We describe *ehex* by running it and inspecting the generated programs.

In our experiments, *ehex* performed quite well in terms of running times with large instances of the Eligibility problem and the Yale Shooting problem, i.e., problems with up to 2^{50} possible epistemic guesses were solved with optimizations in less than one second on a laptop.

6.2 On Program Transformations and Epistemic Semantics

Generally speaking, an epistemic logic program Π can be evaluated by making an assumption of Π on the truth values of the modal literals occurring in Π. For this purpose the program is transformed into another program such that the resulting program is free of modal literals. This program is then evaluated under answer set semantics and its solutions are checked for consistency with the initial assumption.

There are several ways of expressing assumptions, and given an assumption, there are also several ways of transforming the program so that it does not contain modal literals. Consider, for example, the modal reduct (Kahl, 2014) for the K14 semantics on the one hand and the epistemic reduct (Shen and Eiter, 2016) for the SE16$_{\mathcal{X}}$ semantics under $\mathcal{X}$ answer set semantics on the other. The modal reduct is based on a nonempty collection $\mathcal{A}$ of consistent sets of ground literals, whereas the epistemic reduct is based on a set Φ of weak modal literals of Π (Definition 7).

For the K14 semantics, the modal reduct $\Pi^{\mathcal{A}}$ of an epistemic logic program Π relative to $\mathcal{A}$ is defined by transformation rules as in the following table, where a is an atom:

Modal literal φ	If $\mathcal{A}$ satisfies φ	If $\mathcal{A}$ does not satisfy φ
$K\,a$	replace φ with a	delete rule containing φ
not $K\,a$	remove φ	replace φ with *not* a
$M\,a$	remove φ	replace φ with *not not* a
not $M\,a$	replace φ with *not* a	delete rule containing φ

A program Π is transformed into the modal reduct by applying the corresponding rule to each occurrence of a modal literal in Π. As stated in (Kahl, 2014, p. 16), if the syntax allows standard literals ℓ next to the K operator, then $M\,a$ and *not* $M\,a$ can be seen as a shorthand for *not* K *not* a and K *not* a, respectively. Now the transformation rules can be expressed in terms of the K operator:

Modal literal φ	If $\mathcal{A}$ satisfies φ	If $\mathcal{A}$ does not satisfy φ
$K\,\ell$	replace φ with ℓ	delete rule containing φ
not $K\,\ell$	remove φ	replace φ with *not* ℓ

Note the rule that applies when $\mathcal{A}$ does not satisfy $M\,a$ (which expands to *not* K *not* a), i.e., no interpretation $I \in \mathcal{A}$ satisfies a. The application of this rule replaces $M\,a$ with the doubly negated expression *not not* a. The modal reduct is evaluated under the semantics of logic programs with nested expressions (Lifschitz et al., 1999). Under this answer set semantics (NEX semantics), the program $\{\mathrm{p} \leftarrow \textit{not not}\ \mathrm{p}\}$, for example, has the two answer sets $\{\mathrm{p}\}$ and $\emptyset$.[1]

For the SE16$_{\mathcal{X}}$ semantics, the epistemic reduct Π^{Φ} of an epistemic logic program Π is defined with respect to an epistemic guess Φ of Π, where Φ represents a complete truth assignment for modal literals occurring in Π. As before, we denote

[1]This is also how *clingo* evaluates double negation.

the epistemic negation operator by N to distinguish it from our default negation operator *not*.[2] Here $N\,\ell$ over a standard literal ℓ expresses that ℓ cannot be proved to be true, i.e., some answer set of Π^{Φ} does not satisfy ℓ. Remember that $K\,\ell$ and $M\,\ell$ are viewed as shorthand for $not\,N\,\ell$ and $N\,not\,\ell$, respectively. The following table lists modal literals in ELP syntax over an atom a and their corresponding epistemic negations.

Modal literal	Epistemic formula	Intuition
$M\,not\,a$	$N\,not\,not\,a$	"a is sometimes false"
$K\,a$	$not\,N\,a$	"a is always true"
$M\,a$	$N\,not\,a$	"a is sometimes true"
$K\,not\,a$	$not\,N\,not\,a$	"a is always false"

The epistemic reduct Π^{Φ} of Π w.r.t. Φ is obtained by first replacing in Π the epistemic negations that are assumed to be true with $\top$, leading to the program $\Pi^{\top}$. Then the remaining epistemic negations $N\,\psi$ occurring in $\Pi^{\top}$, where ψ is a formula without epistemic negation, are replaced with $not\,\psi$, resulting in the epistemic reduct Π^{Φ}. The following tables show the substitutions of both steps for modal literals in ELP syntax:

Step 1: Replace epistemic negations $N\,\psi \in \Phi$ with $\top$:

Modal literal φ	Epistemic formula	Replace φ with
$M\,not\,a$	$N\,not\,not\,a$	$\top$
$K\,a$	$not\,N\,a$	$not\,\top$
$M\,a$	$N\,not\,a$	$\top$
$K\,not$ a	$not\,N\,not\,a$	$not\,\top$

Step 2: Replace epistemic negations $N\,\psi \in \mathrm{Ep}(\Pi) \setminus \Phi$ with $not\,\psi$:

Modal literal φ	Epistemic formula	Replace φ with
$M\,not\,a$	$N\,not\,not\,a$	$not\,not\,not\,a$
$K\,a$	$not\,N\,a$	$not\,not\,a$
$M\,a$	$N\,not\,a$	$not\,not\,a$
$K\,not$ a	$not\,N\,not\,a$	$not\,not\,not\,a$

If Φ is a correct guess, then for the epistemic negations $N\,\psi$ remaining in $\Pi^{\top}$ after step 1, the formula ψ is supposed to be satisfied by each answer set of $\Pi^{\top}$. Consequently, since Π^{Φ} is obtained from $\Pi^{\top}$ by replacing $N\,\psi$ with $not\,\psi$ in step 2, both $\Pi^{\top}$ and Π^{Φ} are expected to have the same answer sets.

Note that the resulting epistemic reduct may contain rules with multiply negated expressions in their bodies, which some ASP solvers do not accept. Thus, to compute the answer sets of the epistemic reduct with an ASP solver that does not accept such expressions, one has to

[2] In (Shen and Eiter, 2016) the symbols **not** and $\neg$ are used for epistemic negation and default negation, respectively.

1. chose a suitable answer set semantics for programs containing such expressions, e.g., FLP or NEX semantics, and
2. redefine the replacements rules of the epistemic reduct such that the ASP solver accepts the syntax and the newly defined epistemic reduct has the same answer sets as the original epistemic reduct under the chosen answer set semantics.

Example 18. The program $\Pi_{13} = \{\mathrm{p} \leftarrow M\,\mathrm{p}; \leftarrow K\,\mathrm{p}\}$ has the modal reducts

$$\begin{aligned}
\Pi_{13}^{\mathcal{A}_1} &= \{\mathrm{p}; \leftarrow \mathrm{p}\} && \text{w.r.t. } \mathcal{A}_1 = \{\{\mathrm{p}\}\},\\
\Pi_{13}^{\mathcal{A}_2} &= \{\mathrm{p}; \} && \text{w.r.t. } \mathcal{A}_2 = \{\{\mathrm{p}\}, \varnothing\},\\
\Pi_{13}^{\mathcal{A}_3} &= \{\mathrm{p} \leftarrow \mathit{not\,not}\,\mathrm{p}; \} && \text{w.r.t. } \mathcal{A}_3 = \{\varnothing\},
\end{aligned}$$

which under EFLP semantics correspond to the epistemic reducts

$$\begin{aligned}
\Pi_{13}^{\Phi_1} &= \{\mathrm{p}; \leftarrow \mathrm{p}\} && \text{w.r.t. } \Phi_1 = \{M\,\mathrm{p}\},\\
\Pi_{13}^{\Phi_2} &= \{\mathrm{p}; \} && \text{w.r.t. } \Phi_2 = \{M\,\mathrm{p}, M\,\mathit{not}\,\mathrm{p}\},\\
\Pi_{13}^{\Phi_3} &= \{\mathrm{p} \leftarrow \mathrm{p}; \} && \text{w.r.t. } \Phi_3 = \{M\,\mathit{not}\,\mathrm{p}\},
\end{aligned}$$

respectively, where double negation is canceled. Note that Π_{13} has no world view under K14 semantics, because $\mathrm{AS}(\Pi_{13}^{\mathcal{A}_i}) \neq \mathcal{A}_i$ for $i = 1, 2, 3$. But under EFLP ($\mathrm{SE16}_{\mathrm{FLP}}$) semantics it has the unique world view $\{\varnothing\} = \mathrm{AS}(\Pi^{\Phi_3})$.

The EFLP ($\mathrm{SE16}_{\mathrm{FLP}}$) semantics is an instance of the general epistemic semantics, where the epistemic reduct is evaluated under FLP semantics. However, evaluating the epistemic reduct under the semantics for logic programs with nested expressions (NEX semantics) results in the K16 ($\mathrm{SE16}_{\mathrm{NEX}}$) semantics.

6.3 Outlook and Issues

The semantics of programs with epistemic specifications does not seem to be settled yet. Above we discuss that K16 semantics is slightly different from $\mathrm{SE16}_{\mathrm{FLP}}$ semantics. As noted in Cabalar et al. (2019b), both fail to satisfy the *epistemic splitting property*, which holds for, e.g., their newly proposed *Founded Autoepistemic Equilibrium Logic* semantics (Cabalar et al., 2019a) or the semantics of Gelfond's initial approach (G91 semantics). Common to most semantics, however, is the general goal of avoiding self-supporting world views; they differ mainly in the definition of "self-supportedness".

Another issue is the choice of the answer set semantics $\mathcal{X}$ for logic programs without epistemic negation as in Definition 8 of the general epistemic semantics of (Shen and Eiter, 2016). The choice depends on being able to delete double negation (as in FLP semantics) or having the semantics of double negation syntactically for evaluation. For example, if double negations are deleted, then K16 and $\mathrm{SE16}_{\mathrm{NEX}}$ semantics do not coincide because the epistemic reduct contains no nested expressions.

On the implementation side, one could think of more efficient external atoms for consistency checking of epistemic guesses instead of the external atoms of the *Nested* HEX *Plugin*. For example, a dedicated HEX plugin for epistemic logic programs could compute sets of brave and cautious consequences of a subprogram only once per epistemic guess. This might be more (memory) efficient in case that a subprogram (which in this context represents some epistemic reduct of a program) has a large number of answer sets. One can also easily imagine a parallel version of the EHEX algorithm that works similar to the parallel version of the algorithm implemented by *ELPsolve* (Kahl et al., 2016).

In recent years, new approaches to ELP solvers have emerged. For instance, the solver *selp* (Bichler et al., 2018) encodes an input program using large rules and checks for its satisfiability using a single solver call. Other solvers such as *EP-ASP* (Son et al., 2017) or *eclingo* (Cabalar et al., 2020) make use of advanced *clingo* features.

On a concluding note, many problems can be reduced in polynomial time to ELP as computing world views is Σ_4^p-hard, since deciding the existence of a world view is Σ_4^p-complete. Having a larger number of standard problems in addition to the set of problems that are currently used in the literature (e.g., Scholarship Eligibility, Yale Shooting) could inspire more research and better solvers.

Bibliography

Balai, E. and P. Kahl (2014). Epistemic logic programs with sorts. In *Proceedings of ASPOCP 2014*. Source code available at `https://github.com/iensen/elps`.

Balduccini, M., Y. Lierler, and S. Woltran (Eds.) (2019). *Logic Programming and Non-monotonic Reasoning*. Springer International Publishing.

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Bichler, M., M. Morak, and S. Woltran (2018). Single-shot epistemic logic program solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden.*, pp. 1714–1720. Source code available at `https://dbai.tuwien.ac.at/proj/selp/`.

Cabalar, P., J. Fandinno, and L. Fariñas del Cerro (2019a). Founded world views with autoepistemic equilibrium logic. See Balduccini et al. (2019), pp. 134–147.

Cabalar, P., J. Fandinno, and L. Fariñas del Cerro (2019b). Splitting epistemic logic programs. See Balduccini et al. (2019), pp. 120–133.

Cabalar, P., J. Fandinno, J. Garea, J. Romero, and T. Schaub (2020). eclingo: A solver for epistemic logic programs. *Theory and Practice of Logic Programming 20*(6), 834–847.

Calimeri, F., W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub (2012, 13 December). ASP-Core-2: Input language format. Technical report, ASP Standardization Working Group.

Citrigno, S., T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello (2004). The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming*.

Eiter, T., W. Faber, N. Leone, and G. Pfeifer (2000). *Declarative Problem-Solving Using the DLV System*, pp. 79–103. Boston, MA: Springer US.

Eiter, T., M. Fink, G. Ianni, T. Krennwallner, C. Redl, and P. Schüller (2016). A model building framework for answer set programming with external computations. *Theory and Practice of Logic Programming 16*(4), 418–464.

Eiter, T., G. Ianni, and T. Krennwallner (2009). *Answer Set Programming: A Primer*, Volume 5689, pp. 40–110. Berlin, Heidelberg: Springer Berlin Heidelberg.

Eiter, T., G. Ianni, R. Schindlauer, and H. Tompits (2005). A uniform integration of higher-order reasoning and external evaluations in answer set programming. In L. P. Kaelbling and A. Saffiotti (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 90–96. Professional Book Center.

Eiter, T., T. Krennwallner, and C. Redl (2013, October). HEX-programs with nested program calls. In H. Tompits (Ed.), *Proceedings of the Nineteenth International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011)*, Volume 7773 of *LNAI*, pp. 1–10. Springer.

Eiter, T., M. Mehuljic, C. Redl, and P. Schüller (2015). User guide: dlvhex 2.X. Technical Report INFSYS RR-1843-15-05, Vienna University of Technology, Institute for Information Systems.

Faber, W., G. Pfeifer, and N. Leone (2010). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175*(1), 278–298.

Faber, W., G. Pfeifer, N. Leone, T. Dell'Armi, and G. Ielpa (2008). Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming 8*(5–6), 545–580.

Gebser, M., A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub (2015). Abstract Gringo. *Theory and Practice of Logic Programming 15*(4–5), 449–463.

Gebser, M., R. Kaminski, B. Kaufmann, and T. Schaub (2014). Clingo = ASP + control: Preliminary report. *CoRR abs/1405.3694*.

Gelfond, M. (1991). Strong introspection. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pp. 386–391. G91 semantics.

Gelfond, M. (1994). Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence 12*(1–2), 89–116. G94 semantics.

Gelfond, M. (2011). New semantics for epistemic specifications. In *Logic Programming and Nonmonotonic Reasoning – 11th International Conference LPNMR*, pp. 260–265. G11 semantics.

Gelfond, M. and N. Leone (2002). Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence 138*(1–2), 3–38.

Gelfond, M. and V. Lifschitz (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Hanks, S. and D. McDermott (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence 33*(3), 379–412.

Harrison, A., V. Lifschitz, D. Pearce, and A. Valverde (2015). Infinitary equilibrium logic and strong equivalence. In F. Calimeri, G. Ianni, and M. Truszczynski (Eds.), *Logic Programming and Nonmonotonic Reasoning*, Cham, pp. 398–410. Springer International Publishing.

Hintikka, J. (1962). *Knowledge and Belief – an Introduction to the Logic of the Two Notions*. Contemporary philosophy. Cornell University Press.

Kahl, P., R. Watson, E. Balai, M. Gelfond, and Y. Zhang (2015). The language of epistemic specifications (refined) including a prototype solver. *Journal of Logic and Computation*.

Kahl, P. T. (2014). *Refining the Semantics for Epistemic Logic Programs*. Ph. D. thesis, Texas Tech University, Lubbock, USA. K14 semantics.

Kahl, P. T., A. P. Leclerc, and T. C. Son (2016). A parallel memory-efficient epistemic logic program solver: Harder, better, faster. In *Proceedings of ASPOCP 2016*. Solver software available on request: patrick.kahl@navy.mil.

Kelly, M. (2007). Wviews: A worldview solver for epistemic logic programs. Honour's thesis, University of Western Sydney. Source code available at `https://github.com/galactose/wviews`.

Leclerc, A. P. and P. T. Kahl (2018, September). A survey of advances in epistemic logic program solvers.

Lifschitz, V., L. R. Tang, and H. Turner (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*(3/4), 369–389.

Reiter, R. (1978). On closed world data bases. In H. Gallaire and J. Minker (Eds.), *Logic and Data Bases*, pp. 119–140. New York: Plennum Press.

Shen, Y.-D. and T. Eiter (2016). Evaluating epistemic negation in answer set programming. *Artificial Intelligence 237*, 115–135. SE16 semantics.

Shen, Y. D., K. Wang, T. Eiter, M. Fink, C. Redl, T. Krennwallner, and J. Deng (2014). FLP answer set semantics without circular justifications for general logic programs. *Artificial Intelligence*.

Smith, D. E. and D. S. Weld (1998). Conformant graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pp. 889–896.

Son, T. C., T. Le, P. Kahl, and A. Leclerc (2017). On computing world views of epistemic logic programs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 1269–1275. Source code available at `https://github.com/tiep/EP-ASP`.

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*, Volume 6565 of *Lecture Notes in Computer Science*, pp. 315–333. Springer.

Zhang, Z., K. Zhao, and R. Cui (2013, November). ESmodels: An inference engine of epistemic specifications. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 769–774.