A Survey of Advances in Epistemic Logic Program Solvers

Anthony Leclerc and Patrick Kahl

June 1, 2018

A survey of advances in epistemic logic program solvers

ANTHONY P. LECLERC
Space and Naval Warfare Systems Center Atlantic, North Charleston, SC, USA
College of Charleston, Charleston, SC, USA
(e-mail: anthony.leclerc@navy.mil; leclerca@cofc.edu)

PATRICK THOR KAHL
Space and Naval Warfare Systems Center Atlantic, North Charleston, SC, USA
(e-mail: patrick.kahl@navy.mil)

submitted TBD; revised TBD; accepted TBD

Abstract

Recent research in extensions of Answer Set Programming has included a renewed interest in the language of Epistemic Specifications, which adds modal operators K ("known") and M ("may be true") to provide for more powerful introspective reasoning and enhanced capability, particularly when reasoning with incomplete information. An *epistemic logic program* is a set of rules in this language. Infused with the research has been the desire for an efficient solver to enable the practical use of such programs for problem solving. In this paper, we report on the current state of development of epistemic logic program solvers.

KEYWORDS: Epistemic Logic Program Solvers, Epistemic Specifications, Epistemic Logic Programs, World Views, Solvers, Epistemic Negations, Answer Set Programming Extensions, Logic Programming

1 Introduction

In the study of knowledge representation and reasoning as related to logic programming, the need for sufficient expressive power in order to correctly represent incomplete information and perform introspective reasoning when modeling an agent's knowledge of the world has been slowly realized. As such, Michael Gelfond's language of *Epistemic Specifications* (Gelfond 1991; Gelfond 1994) has seen renewed interest (Faber and Woltran 2011; Truszczyński 2011; Gelfond 2011; Kahl 2014; Kahl et al. 2015; Su 2015; Fariñas del Cerro et al. 2015; Shen and Eiter 2016; Zhang and Zhang 2017a). Much of the focus of late has been on semantic subtleties, particularly for rules involving recursion through modal operators. However, concomitant interest in the development of solvers for finding the *world views* (collections of *belief sets* analogous to the answer sets of an ASP program) of an epistemic logic program has progressed to the point that a number of choices are now available: *ESmodels* (Zhang et al. 2013b), *ELPS* (Balai 2014), *ELPsolve* (Leclerc and Kahl 2016), *EP-ASP* (Le and Son 2017), *Wviews* (Kelly 2018), *EHEX* (Strasser 2018), and *selp* (Bichler et al. 2018). Additionally, *GISolver* (Zhang et al. 2015a) and *PelpSolver* (Zhang and Zhang 2017b) are tools for finding the world views of extensions of Epistemic Specifications that can also be used for epistemic logic programs with minor syntactic translation. For awareness and to promote continued research, development, and use of Epistemic Specifications and its variants, we present a survey of epistemic logic program solvers.[1]

---

[1] DISCLAIMER: The views and opinions expressed may not reflect those of the US Government.

The paper is organized as follows. In section 2 we provide a brief overview of the language of Epistemic Specifications including a synopsis of the syntax and semantics of the different versions supported by the solvers included in this survey. In section 3 we discuss the solvers themselves and consider history, influences, implementation, and key features. In section 4 we include performance data on extant solvers compiled from experiments on select epistemic logic programs. We close with a summary and statements about the future of ELP solvers.

2 Epistemic Specifications

Gelfond presented the following example in (Gelfond 1991) to demonstrate the need for extending the language of what we now call answer set programming (ASP) in order to "allow for the correct representation of incomplete information in the presence of multiple answer sets."

% rules for scholarship eligibility at a certain college
$eligible(S) \leftarrow highGPA(S).$
$eligible(S) \leftarrow fairGPA(S),\ minority(S).$
$\neg eligible(S) \leftarrow \neg highGPA(S),\ \neg fairGPA(S).$
% ASP attempt to express an interview requirement when eligibility cannot be determined
$interview(S) \leftarrow \texttt{not}\ eligible(S),\ \texttt{not}\ \neg eligible(S).$
% applicant data
$fairGPA(mike)\ \texttt{or}\ highGPA(mike).$

This program correctly computes that the eligibility of Mike is indeterminate, but its answer sets, $\{fairGPA(mike), interview(mike)\}$ and $\{highGPA(mike), eligible(mike)\}$, do not conclude that an interview is required since only one contains $interview(mike)$.

Gelfond's solution was to extend ASP by adding modal operator K ("known") and changing the fourth rule above to:

% updated rule to express interview requirement using modal operator K
$interview(S) \leftarrow \texttt{not}\ \mathrm{K}\, eligible(S),\ \texttt{not}\ \mathrm{K}\, \neg eligible(S).$

The updated rule means that $interview(S)$ is true if both $eligible(S)$ and $\neg eligible(S)$ are each *not known* (i.e., not in all belief sets of the world view).

The new program has a world view with two belief sets: $\{fairGPA(mike), interview(mike)\}$ and $\{highGPA(mike), eligible(mike), interview(mike)\}$, both containing $interview(mike)$. It therefore correctly entails that Mike is to be interviewed.

Since its 1991 introduction, four revisions of the language of Epistemic Specifications have been implemented in solvers. Other revisions of Epistemic Specifications have been proposed (Fariñas del Cerro et al. 2015; Zhang and Zhang 2017a), but to the best of our knowledge, no solvers for those versions were implemented. The revision we call ***ES1994*** is described in (Gelfond 1994; Baral and Gelfond 1994). With a renewed interest in Epistemic Specifications nearly two decades later, Gelfond proposed an update (Gelfond 2011) to the language in an attempt to avoid unintended world views due to recursion through modal operator K. We refer to this version as ***ES2011***. Continuing with Gelfond's efforts to avoid unintended world views due to recursion, but through modal operator M, Kahl proposed a further update (Kahl 2014). We refer to this version as ***ES2014***. Most recently, Shen and Eiter proposed yet another update (Shen and Eiter 2016) to address perceived issues with unintended world views remaining in the language. We call this version ***ES2016***.

A synopsis of the syntax and semantics of the different versions of Epistemic Specifications covered by the surveyed solvers is given below. We encourage the reader to see the papers previously referenced for more detailed discussions of individual language versions.

In general, the syntax and semantics of Epistemic Specifications follow those of ASP with the notable addition of modal operators K and M and the new notion of a *world view*. A world view of an ELP is a collection of *belief sets* (analogous to the answer sets of an ASP program) that satisfies the the rules of the ELP and meets certain other requirements as given in the table below.

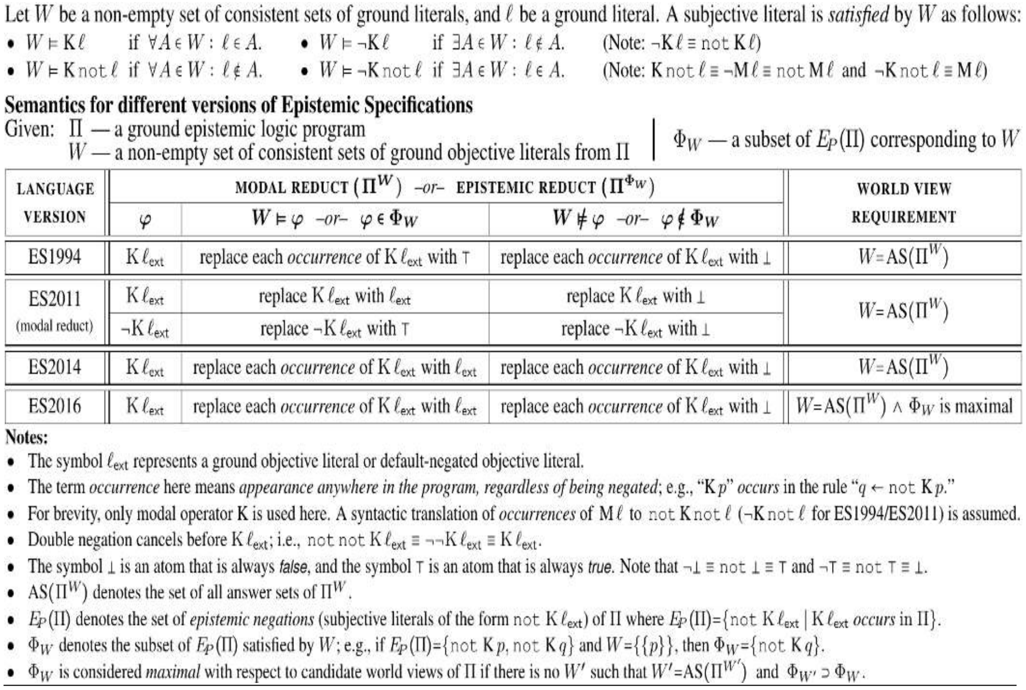
Syntax

An epistemic logic program (ELP) is a set of rules in the language of Epistemic Specifications, a rule having the form

$$\ell_1 \texttt{ or } \ldots \texttt{ or } \ell_k \leftarrow e_1,\ \ldots,\ e_n.$$

where $k \geq 0$, $n \geq 0$, each ℓ_i is a *literal* (an atom or a classically-negated atom; called an *objective literal* when needed to avoid ambiguity), and each e_i is a literal or a *subjective literal* (a literal immediately preceded by K or M) possibly preceded by `not` (default negation).[2] As in ASP, a rule having an objective/subjective literal with a variable term is a shorthand for all ground instantiations of the rule. The $\leftarrow$ symbol is optional if the body of the rule is empty (i.e., n=0).

When a Subjective Literal Is Satisfied



Let W be a non-empty set of consistent sets of ground literals, and ℓ be a ground literal. A subjective literal is *satisfied* by W as follows:

- $W \models \mathrm{K}\ell$ if $\forall A \in W : \ell \in A$.
- $W \models \neg\mathrm{K}\ell$ if $\exists A \in W : \ell \notin A$. (Note: $\neg\mathrm{K}\ell \equiv \texttt{not}\,\mathrm{K}\ell$)
- $W \models \mathrm{K}\,\texttt{not}\,\ell$ if $\forall A \in W : \ell \notin A$.
- $W \models \neg\mathrm{K}\,\texttt{not}\,\ell$ if $\exists A \in W : \ell \in A$. (Note: $\mathrm{K}\,\texttt{not}\,\ell \equiv \neg\mathrm{M}\ell \equiv \texttt{not}\,\mathrm{M}\ell$ and $\neg\mathrm{K}\,\texttt{not}\,\ell \equiv \mathrm{M}\ell$)

Semantics for different versions of Epistemic Specifications

Given: Π — a ground epistemic logic program
W — a non-empty set of consistent sets of ground objective literals from Π
Φ_W — a subset of $E_P(\Pi)$ corresponding to W

LANGUAGE VERSION	MODAL REDUCT (Π^W) *–or–* EPISTEMIC REDUCT (Π^{Φ_W})			WORLD VIEW REQUIREMENT
	φ	$W \models \varphi$ *–or–* $\varphi \in \Phi_W$	$W \not\models \varphi$ *–or–* $\varphi \notin \Phi_W$	
ES1994	$\mathrm{K}\,\ell_{\text{ext}}$	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with $\top$	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with $\bot$	W=AS(Π^W)
ES2011 (modal reduct)	$\mathrm{K}\,\ell_{\text{ext}}$	replace $\mathrm{K}\,\ell_{\text{ext}}$ with ℓ_{ext}	replace $\mathrm{K}\,\ell_{\text{ext}}$ with $\bot$	W=AS(Π^W)
	$\neg\mathrm{K}\,\ell_{\text{ext}}$	replace $\neg\mathrm{K}\,\ell_{\text{ext}}$ with $\top$	replace $\neg\mathrm{K}\,\ell_{\text{ext}}$ with $\bot$	
ES2014	$\mathrm{K}\,\ell_{\text{ext}}$	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with ℓ_{ext}	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with $\bot$	W=AS(Π^W)
ES2016	$\mathrm{K}\,\ell_{\text{ext}}$	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with ℓ_{ext}	replace each *occurrence* of $\mathrm{K}\,\ell_{\text{ext}}$ with $\bot$	W=AS$(\Pi^W) \wedge \Phi_W$ is maximal

Notes:

- The symbol ℓ_{ext} represents a ground objective literal or default-negated objective literal.
- The term *occurrence* here means *appearance anywhere in the program, regardless of being negated*; e.g., "K p" *occurs* in the rule "$q \leftarrow \texttt{not}\ \mathrm{K}\,p$."
- For brevity, only modal operator K is used here. A syntactic translation of *occurrences* of $\mathrm{M}\,\ell$ to $\texttt{not}\,\mathrm{K}\,\texttt{not}\,\ell$ ($\neg\mathrm{K}\,\texttt{not}\,\ell$ for ES1994/ES2011) is assumed.
- Double negation cancels before $\mathrm{K}\,\ell_{\text{ext}}$; i.e., $\texttt{not}\,\texttt{not}\,\mathrm{K}\,\ell_{\text{ext}} \equiv \neg\neg\mathrm{K}\,\ell_{\text{ext}} \equiv \mathrm{K}\,\ell_{\text{ext}}$.
- The symbol $\bot$ is an atom that is always *false*, and the symbol $\top$ is an atom that is always *true*. Note that $\neg\bot \equiv \texttt{not}\,\bot \equiv \top$ and $\neg\top \equiv \texttt{not}\,\top \equiv \bot$.
- AS(Π^W) denotes the set of all answer sets of Π^W.
- $E_P(\Pi)$ denotes the set of *epistemic negations* (subjective literals of the form $\texttt{not}\ \mathrm{K}\,\ell_{\text{ext}}$) of Π where $E_P(\Pi)$=$\{\texttt{not}\ \mathrm{K}\,\ell_{\text{ext}} \mid \mathrm{K}\,\ell_{\text{ext}}$ *occurs* in $\Pi\}$.
- Φ_W denotes the subset of $E_P(\Pi)$ satisfied by W; e.g., if $E_P(\Pi)$=$\{\texttt{not}\ \mathrm{K}\,p, \texttt{not}\ \mathrm{K}\,q\}$ and W=$\{\{p\}\}$, then Φ_W=$\{\texttt{not}\ \mathrm{K}\,q\}$.
- Φ_W is considered *maximal* with respect to candidate world views of Π if there is no W' such that W'=AS$(\Pi^{W'})$ and $\Phi_{W'} \supset \Phi_W$.

3 Solvers

In the subsections below we discuss the ELP solver development efforts spanning, in chronological order, the years from 1994 to 2018. Included in the group are two solvers, *GISolver* and *PelpSolver*, which were designed for different extensions of ASP, but nevertheless are able to compute the world views of ELPs given simple translations of the input language encoding.

We note that all of the extant solvers discussed operate from the command line, which is to say that no Integrated Development Environment (IDE) or Graphical User Interface (GUI) currently

---

[2] In ES1994 and ES2011, negated subjective literals have their modal operators prefaced with $\neg$ rather than `not`. In the semantics given above, we extend the syntax by allowing default-negated literals to follow modal operator K and consider $\mathrm{M}\,\ell$ to be simply a shorthand for $\texttt{not}\ \mathrm{K}\,\texttt{not}\ \ell$ (or $\neg\mathrm{K}\,\texttt{not}\ \ell$ in ES1994/ES2011 syntax).

exists for solving ELPs. We also note that all extant ELP solvers generate what can be called an *epistemic reduct framework* for the ELP. This is a core ASP program that when instantiated with a “guess” (truth value assignments for the subjective literals represented by a subset of the epistemic negations that are considered *true*) will correspond to the *epistemic reduct* for that guess. An underlying (or background) ASP solver such as *DLV* (DLVSYSTEM S.r.l. 2012), *DLVHEX2* (Redl et al. 2017), *claspD*, or *clingo* (Kaminski and Kaufmann 2018)) is then used to compute the answer sets of the epistemic reduct.

The terms “loosely coupled” and “tightly coupled” are used in our discussions of the implementations of the solvers. By loosely coupled we mean that the underlying ASP solver is invoked as a separate process rather than through a library with a specific Application Programming Interface (API). A loosely coupled implementation has the advantage that it can be easily modified to utilize a different underlying ASP solver, assuming the capabilities and input language syntax of the ASP solvers are similar. A tightly coupled implementation is not as flexible but generally more efficient, as it avoids the overhead of creating and communicating with a separate process.

The input language of a given solver is typically a subset of the ASP Core 2 standard (Calimeri et al. 2013) with the addition of modal operators K and M. For example, the “$\leftarrow$” symbol is typically represented by the 2-character string “`:-`” though some solvers may accept other representations. *ELPsolve* and *EP-ASP* rely on *ELPS* for preprocessing the input program, which requires additional statements in the program to explicitly define the domain for predicate terms as a *sorted signature*. The input language of *ELPS* also uses “`K$`” and “`M$`” to represent modal operator symbols “K” and “M” (respectively). The *selp* system accepts the same input language as *ELPS*, but does not depend on *ELPS* for processing. It can alternatively accept “`$not$`” as the *epistemic negation operator*, which is equivalent to “`not` K” in our notation. We refer the reader to documentation and example programs available with the solver distributions for specifics on the individual input languages. We will continue to use the notation described in Section 2 with the understanding that it differs from the actual input languages of the various solvers.

Near the end of the paper are a number of summary tables. These include a historical synopsis of solver development (Table 1), a brief summary of solver features (Table 2), and a listing of solver contacts & download information (Table 4).

3.1 ELMO

The earliest work on the development of an ELP solver was that of Richard Watson in 1994 while a graduate student of Michael Gelfond when he was at the University of Texas at El Paso. Though not a solver per se, Watson’s *ELectronic MOnk* (*ELMO*) was a Prolog implementation of an inference engine for a limited class of ELPs. *ELMO* also required the *SLG* system developed at Southern Methodist University and State University of New York (SUNY) at Stony Brook (Chen and Warren 1993). There is no extant electronic binary or source; however, the printed source code is listed as an appendix of Watson’s master’s thesis.

In his thesis, Watson demonstrates the efficacy of *ELMO* by reporting the answers to queries using ELMO for various examples, including the scholarship eligibility problem of Section 2.

3.2 sismodels

In 2001, Marcello Balduccini, working as a graduate student with Michael Gelfond at Texas Tech University, began work on a solver that extended *Smodels* (Simons 2000; Syrjänen and Simons 2010) with strong introspection. He called his solver *sismodels*. The work, however, never progressed beyond proof-of-concept. As with *ELMO*, there is no extant electronic binary or source for *sismodels*. It is included here as it is the first known attempt to implement an ELP solver in the sense that its output was the world views of the input ELP.

3.3 Wviews

Working with Yan Zhang as his advisor for his honours thesis (Kelly 2007) at the University of Western Sydney, Michael Kelly implemented an ES1994 solver *Wviews* based on the algorithm suggested in (Zhang 2006). Kelly's implementation features a grounder and a solver in a single executable that is loosely coupled with *DLV* as the background ASP solver. This was the first general epistemic logic program solver, and it is still available as a Microsoft Windows executable. Although the original C++ source code for this version of the solver was lost, Kelly has recently posted a Python version of *Wviews* (Kelly 2018) that we will refer to as *Wviews2*. This new version contains "major modifications" according to its author.

User Experience: Wviews2 is the one to use for ES1994 semantics. We note that *Wviews2* tries one guess at a time, which can result in calling the underlying ASP solver 2^k times, where k is the number of epistemic negations, limiting its practical use to relatively small (w.r.t. the number of epistemic negations) ELPs. Overcoming this limitation is a challenge for all solver developers. *Wviews2* exhausts the search space iteratively to ensure all world views are computed.

3.4 ESmodels

After spending the summer of 2011 at Texas Tech University, Zhizheng Zhang returned to Southeast University with the idea of implementing a solver for Gelfond's new version of Epistemic Specifications, ES2011. He started with a grounder, and by 2012 had implemented (with the help of graduate students Rongcun Cui and Kaikai Zhao) *ESParser* (Cui et al. 2012). This was followed by *ESsolve* in 2013, resulting in a grounder-solver system they called *ESmodels* (Zhang et al. 2013a). *ESsolve* is loosely coupled with ASP solver *claspD*.

Although work on *ESmodels* continued for a short time (Zhang and Zhao 2014), the system is available today only as a Microsoft Windows executable from Zhang's homepage at Southeast University. It is the only ES2011 solver known.

User Experience: ESmodels appears to work reasonably well with programs that are relatively small w.r.t. the number of epistemic negations. With larger programs, we sometimes observed a runtime error or the unexpected result of no world views for programs known to be consistent.

We note that the M modal operator is not directly supported; however, equivalent[3] constructs can be created by replacing each *occurrence* of $\mathrm{M}\,\ell$ as follows:

1. Replace $\mathrm{M}\,\ell$ with $\neg\mathrm{K}\,\ell'$ where ℓ' is a fresh atom. (Remove any double negation before K.)
2. Add the following new rule: $\ell' \leftarrow \texttt{not}\,\ell$.

Classical/strong negation is also not directly supported other than to denote a negated subjective literal, but, as before, a workaround exists by replacing each occurrence of $\neg\ell$ as follows:

1. Replace $\neg\ell$ with ℓ' where ℓ' is a fresh atom.
2. Add the following constraint: $\leftarrow \ell,\ \ell'$.

---

[3] Equivalence here is with respect to the world views of respective programs, modulo any fresh atoms introduced.

3.5 ELPS

As graduate students at Texas Tech University, Evgenii Balai and Patrick Kahl worked together on a version of Epistemic Specifications that uses a sorted signature. A program written in this version is called an *epistemic logic program with sorts* (Balai and Kahl 2014). This effort was strongly influenced by Balai's work on *SPARC* (Balai et al. 2013), a version of the language of ASP using a sorted signature. Balai implemented the ES2014 (with sorted signature) solver *ELPS* using an algorithm formed by combining Kahl's ES2014 algorithm with Balai's SPARC algorithm. Much of the Java code from an old version of SPARC was able to be reused, allowing Balai to create a working solver in about three days worth of work—an impressive feat. *ELPS* is loosely coupled with the ASP solver *clingo*.

Although *ELPS* is a stable, reliable ES2014 solver for small (in number of epistemic negations) programs that makes only one call to the underlying ASP solver, its memory requirements can grow exponentially with the number of epistemic negations (Kahl et al. 2016). It does, however, provide a nice front end for other solvers, such as *ELPsolve* and *EP-ASP*, to be able to translate an ELP with sorts into an ASP epistemic reduct framework. Java source code and a pre-built *.jar* file are available.

User Experience: ELPS works very well for programs that are relatively small with respect to the number of epistemic negations, but due to exponentially-growing memory needs as the number of epistemic negations grow, it has limited application as a solver. Nonetheless, it is one of the only solvers with a detailed user manual. We note that it outputs all world views of its input program with no option for changing this. It does have the option "-o" for outputting a file representing the epistemic reduct framework of the input program, along with rules for generating all combinations of subjective literal truth values. This gives *ELPS* potential value as a front end for other solvers.

3.6 GISolver

Zhizheng Zhang and graduate students Bin Wang and Shutao Zhang embarked on developing the solver *GISolver* for an extension of ASP called *GI-log* (Zhang et al. 2015b). *GISolver* can be used to find world views of ES2014 programs after minor syntactic translations. It is loosely coupled with *clingo* as the underlying ASP solver. Like *ESmodels*, this solver is currently available only as a Microsoft Windows executable from Zhang's homepage at Southeast University. It appears to have been a stepping stone in the development of *PelpSolver* discussed later.

User Experience: GISolver works well for relatively small (w.r.t. the number epistemic negations) ELPs provided they are appropriately translated to GI-log syntax by converting subjective literals as shown below:

ES2014 syntax		**GI-log syntax**
`K p`	⇒	`K[1,1] p`
`not K p`	⇒	`K[0,1) p`
`M p`	⇒	`K(0,1] p`
`not M p`	⇒	`K[0,0] p`

3.7 ELPsolve

ELPsolve was developed in 2016 by the authors. Two primary efficiency goals were pursued: (1) develop an ELP solver that avoids the large memory requirements of *ELPS*; and (2) parallelize the solver to take advantage of multi-core processors. Other goals included support for the updated semantics of Shen & Eiter (ES2016) and optimization for conformant planning. To solve the memory issue, *ELPsolve* partitions guesses into fixed-sized groups, rather than computing all guesses with one ASP solver call. These groups are systematically generated in an order that guarantees the maximality requirement of ES2016 and permits pruning of the search space when multiple world views are desired. Groups of guesses are mutually exclusive so that parallelization can occur with minimal synchronization. *ELPsolve* supports both ES2014 and ES2016 semantics. Binary executables for Windows, Mac, and Linux are available upon request.

User Experience: ELPsolve has several options, including the ability to specify the (maximum) number of world views to output, the number of processors to be used, conformant planning mode (with planning horizon), and a configuration file. The configuration file is used to specify less volatile configuration options such as group size, language semantics to use (ES2014 or ES2016), and ASP solver path. *ELPsolve* itself is invoked from a script which first seamlessly calls ELPS for translating the ELP (with sorts) input program into an epistemic reduct framework, then invokes ASP grounder *gringo* to ground the program, and finally calls *ELPsolve* for further processing. *ELPsolve* is loosely-coupled with *clingo* for backend ASP program solving.

3.8 EP-ASP

Tran Cao Son worked as an Office of Naval Research faculty researcher at Space and Naval Warfare Systems Center Atlantic in the summer of 2016. His work with the authors on the development of *ELPsolve* stimulated his interest and led to his own approach, resulting in a new solver: *EP-ASP*. The core idea of this solver is to take the epistemic reduct framework (as in *ELPS* and *ELPsolve*), but instead of solving for all possible guesses at once (like *ELPS*) or systematically in groups of guesses (like *ELPsolve*), it uses the underlying ASP solver to compute a single answer set. Due to the way the epistemic reduct framework is constructed, this answer set represents a *consistent* guess (i.e., one that results in a consistent epistemic reduct). The framework is instantiated for that guess, all answer sets are computed, and the answer sets are checked to see if they represent a world view. A constraint is then added to eliminate this guess from further consideration, and the process is repeated until all world views of the program are discovered.

For input to *EP-ASP*, an epistemic reduct framework representation of the ELP is created first using *ELPS*. *EP-ASP* works completely within the *clingo* runtime environment, using embedded Python to control iteration in a *multi-shot ASP solving* approach (Gebser et al. 2017). After creating a proof-of-concept version for ES2014, Son enlisted the aid of his New Mexico State University graduate student Tiep Le to implement support for ES2016, the use of brave and cautious reasoning for pruning the search space, and optimizations for conformant planning.

The solver supports both ES2014 and ES2016 semantics and is among the fastest solvers for the sample programs used in our tests.

User Experience: EP-ASP has several options, including the ability to specify the use of brave and cautious consequences as a preliminary step to prune the search space, language semantics to use (ES2014 or ES2016), and conformant planning mode.

3.9 PelpSolver

Continuing with the success of *GISolver*, Zhizheng Zhang and Shutao Zhang developed a solver for probabilistic-epistemic logic programs (Zhang and Zhang 2017a) called *PelpSolver*. With appropriate syntactic translation, *PelpSolver* can be used to solve ES2016 programs. It is implemented in Java and is loosely coupled with *clingo* as the underlying ASP solver.

The development of the language of probabilistic-epistemic logic programs was a culmination of language extensions that were positively influenced by ELP solver development. During development of *ESmodels*, implementation of the world view verification step involved counting the number of occurrences, $count(\ell)$, of the objective literal part, ℓ, of each subjective literal in the computed belief sets. For example, if checking subjective literals against a set of, say, 5 belief sets, to verify $\mathrm{K}\, p$, $count(p) = 5$ is required, to verify $\mathrm{M}\, q$, $count(q) \geq 1$ is required, and so forth. They observed that other numbers/number ranges could easily be checked, leading to the realization that the ability to specify the *fraction* of belief sets required to contain a particular literal might be useful for modeling certain problems. This led to the new language extensions.

User Experience: PelpSolver comes with a pre-built *.jar* file, but can also be built using a Maven *pom.xml* file. One command-line option exists for optimization. The conversion from an ELP program to a probabilistic-epistemic logic program is the same as that given for *GISolver*.

3.10 ELPsolve2

ELPsolve2 was developed in 2017 by the authors. Unlike *ELPsolve*, this version of the software has not been officially released to the public, nor have there been any technical papers written about it. For this reason we describe *ELPsolve2* in a little more detail for this survey.

Two primary design goals guided the development of *ELPsolve2*: efficiency and support for additional features. Specifically, *ELPsolve2* improves on *ELPsolve* in five ways:

- replaces “loosely coupled” ASP solver interaction with “tightly coupled” interaction
- implements an “invalid guess” filter
- uses brave and cautious reasoning to reduce the number of epistemic negations
- improves the optimization used for conformant planning problems
- implements *World View Constraints* (WVCs)

Both *ELPsolve* and *ELPsolve2* utilize the *clingo* ASP solver for solving the epistemic reduct framework. With *ELPsolve*, calls to *clingo* are performed as external processes that require time to instantiate. Furthermore, these processes communicate results less efficiently through the operating system. Instead, *ELPsolve2* utilizes *clingo's C* programming language interface. Time to invoke a *clingo* call and store the results is therefore reduced.

We call a guess that contains epistemic negations that cannot co-exist an “invalid guess.” *ELPsolve2* filters such guesses, thus avoiding unnecessary computation. The following pairs of epistemic negations cannot co-exist:

- $\mathrm{K}\, \ell$ and $\texttt{not}\, \mathrm{M}\, \ell$
- $\mathrm{K}\, \ell$ and $\mathrm{M}\, \overline{\ell}$
- $\mathrm{K}\, \ell$ and $\mathrm{K}\, \overline{\ell}$

where $\overline{\ell}$ denotes the logical complement of ℓ. For example, if $\ell = \neg p$ then $\overline{\ell} = p$.

Brave and cautious reasoning was first successfully used in *EP-ASP* to reduce the number of epistemic negations under consideration, pruning the search space for certain ELPs. *ELPSolve2* incorporates this optimization. We note that for some problems, brave and cautious reasoning yields no reduction (e.g., conformant planning problem); however, for others a considerable reduction is achieved (e.g., scholarship eligibility problem).

ELPsolve2 improves the optimization for conformant planning problems over *ELPsolve* by further reducing the search space based on the assumption that only one action is performed at each step. Although this assumption may seem too constraining, optimizations related to conformant planning are highly specialized and can result in dramatic improvements in performance when applied as intended.

Finally, *ELPsolve2* allows for the extension known as *world view constraints* proposed by the authors in (Kahl and Leclerc 2018). This has the potential for reduction of the search space over encodings that do not use world view constraints. Thus, from a solver perspective, this can be viewed as a general approach with the potential for performance improvement rather than an optimization applicable only to very specific applications such as conformant planning.

User Experience: ELPsolve2 comes with all the options from *ELPsolve*, and adds options for brave and cautious reasoning as well as different output formats.

3.11 EHEX

At the time of this writing, Anton “Tonico” Strasser is a graduate student at TU Wien working under the advisement of Thomas Eiter and Christoph Redl. His ES2016 solver *EHEX* adds epistemic negations to HEX programs, which allows integration of external computation sources. *EHEX* works with *DLVHEX2* as the underlying ASP solver, but uses *clingo* as well to perform optional brave and cautious reasoning. *EHEX* is written in Python and is loosely coupled with the ASP solver.

User Experience: EHEX has a number of options and many example programs are available on the developer’s GitHub page. Given an already existing installation of *DLVHEX2* with the *NestedHexPlugin*, *EHEX* builds and installs easily. However, we found it challenging to build *DLVHEX2* with the *NestedHexPlugin* from source. Even though it is a work-in-progress as of this writing, *EHEX* performed quite well. We look forward to further developments.

3.12 selp

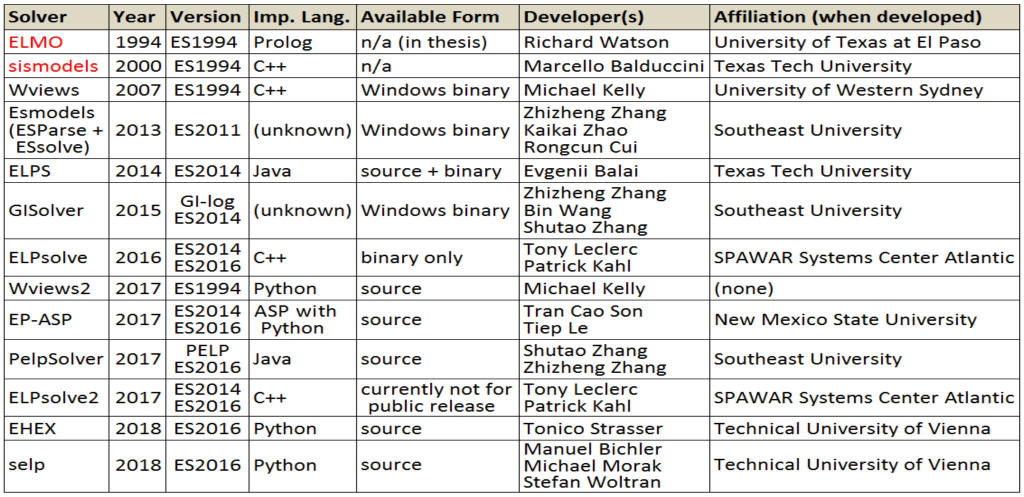
Another graduate student at TU Wien, Manuel Bichler, working under the advisement of Stefan Woltran and Michael Morak, applied ASP rule decomposition (Bichler et al. 2016) to ELP solving to develop a single-shot (w.r.t. ASP solver calls) epistemic logic program solver called *selp*. The *selp* system is loosely coupled with *clingo*, and uses the *lpopt* tool (Bichler 2015) to efficiently decompose “large” logic programming rules into smaller rules with the expectation that such rules are more manageable/easier for *clingo* to handle.

User Experience: The *selp* system includes a number of Python scripts, including its own tool for processing an input epistemic logic programs with sorts. It generates rules containing a relatively large number of body literals. The intent is to optimize the rules for decomposition using the *lpopt* tool. This approach appears to work quite well for certain programs (e.g., the scholarship eligibility problem described in Section 2) based on our experiments. It also appears to benefit from the use of multiple threads with the backend ASP solver *clingo*.

3.13 Solver Summary

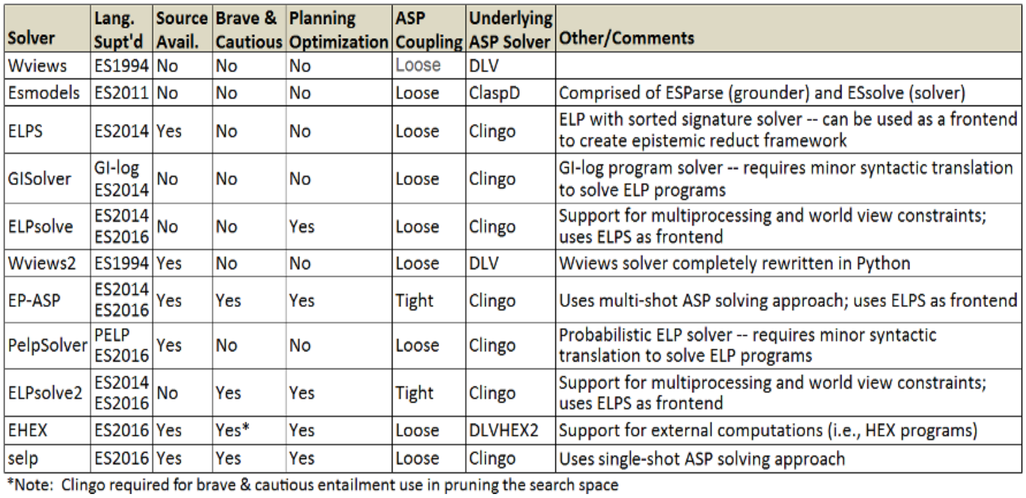
Table 1 provides a general summary of all known ELP solvers. *ELMO* and *sismodels* are highlighted in red to indicate they no longer exist. Table 2 shows some of the key features of the ELP solvers included in the performance experiments discussed in the next section.

Table 1. Epistemic Logic Program Solvers



Solver	Year	Version	Imp. Lang.	Available Form	Developer(s)	Affiliation (when developed)
ELMO	1994	ES1994	Prolog	n/a (in thesis)	Richard Watson	University of Texas at El Paso
sismodels	2000	ES1994	C++	n/a	Marcello Balduccini	Texas Tech University
Wviews	2007	ES1994	C++	Windows binary	Michael Kelly	University of Western Sydney
Esmodels (ESParse + ESsolve)	2013	ES2011	(unknown)	Windows binary	Zhizheng Zhang Kaikai Zhao Rongcun Cui	Southeast University
ELPS	2014	ES2014	Java	source + binary	Evgenii Balai	Texas Tech University
GISolver	2015	GI-log ES2014	(unknown)	Windows binary	Zhizheng Zhang Bin Wang Shutao Zhang	Southeast University
ELPsolve	2016	ES2014 ES2016	C++	binary only	Tony Leclerc Patrick Kahl	SPAWAR Systems Center Atlantic
Wviews2	2017	ES1994	Python	source	Michael Kelly	(none)
EP-ASP	2017	ES2014 ES2016	ASP with Python	source	Tran Cao Son Tiep Le	New Mexico State University
PelpSolver	2017	PELP ES2016	Java	source	Shutao Zhang Zhizheng Zhang	Southeast University
ELPsolve2	2017	ES2014 ES2016	C++	currently not for public release	Tony Leclerc Patrick Kahl	SPAWAR Systems Center Atlantic
EHEX	2018	ES2016	Python	source	Tonico Strasser	Technical University of Vienna
selp	2018	ES2016	Python	source	Manuel Bichler Michael Morak Stefan Woltran	Technical University of Vienna

Table 2. ELP Solver Features



Solver	Lang. Supt'd	Source Avail.	Brave & Cautious	Planning Optimization	ASP Coupling	Underlying ASP Solver	Other/Comments
Wviews	ES1994	No	No	No	Loose	DLV	
Esmodels	ES2011	No	No	No	Loose	ClaspD	Comprised of ESParse (grounder) and ESsolve (solver)
ELPS	ES2014	Yes	No	No	Loose	Clingo	ELP with sorted signature solver -- can be used as a frontend to create epistemic reduct framework
GISolver	GI-log ES2014	No	No	No	Loose	Clingo	GI-log program solver -- requires minor syntactic translation to solve ELP programs
ELPsolve	ES2014 ES2016	No	No	Yes	Loose	Clingo	Support for multiprocessing and world view constraints; uses ELPS as frontend
Wviews2	ES1994	Yes	No	No	Loose	DLV	Wviews solver completely rewritten in Python
EP-ASP	ES2014 ES2016	Yes	Yes	Yes	Tight	Clingo	Uses multi-shot ASP solving approach; uses ELPS as frontend
PelpSolver	PELP ES2016	Yes	No	No	Loose	Clingo	Probabilistic ELP solver -- requires minor syntactic translation to solve ELP programs
ELPsolve2	ES2014 ES2016	No	Yes	Yes	Tight	Clingo	Support for multiprocessing and world view constraints; uses ELPS as frontend
EHEX	ES2016	Yes	Yes*	Yes	Loose	DLVHEX2	Support for external computations (i.e., HEX programs)
selp	ES2016	Yes	Yes	Yes	Loose	Clingo	Uses single-shot ASP solving approach

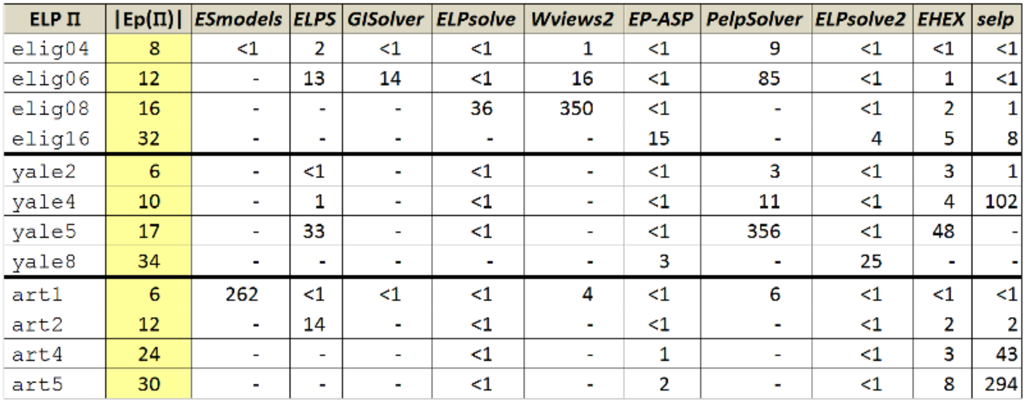
*Note: Clingo required for brave & cautious entailment use in pruning the search space

4 Experiments

Three epistemic logic programs of various sizes (w.r.t. the number of epistemic negations) were used to test the capabilities and performance of different solvers. The *elig*`NN` programs are instances of the scholarship eligibility example described in Section 2, where `NN` indicates the number of applicants. The *yale*`N` programs are instances of a variation of the Yale shooting problem (Hanks and McDermott 1987) encoded as describe in (Kahl et al. 2015), where `N` indicates the plan horizon. The *art*`N` programs are instances of a scalable artificial problem we constructed involving combinations of both K and M modal operators, where `N` is the scaling factor. Program listings are not included due to space constraints but are available upon request.

The test machine has an Intel i7 820QM @ 1.73 GHz processor with 8 GB RAM. *ESmodels* and *GISolver* were run using a 64-bit Windows 10 operating system. All other solvers were run using a 64-bit Ubuntu 16.04 (Linux) operating system. *ELPsolve* and *EP-ASP* use *ELPS* to create an epistemic reduct framework file from the input ELP (with sorts) file. Table 3 shows the runtime results (in seconds) for our tests. Times reported are for the entire solving experience, including (as appropriate) time for creating the epistemic reduct framework file, time for grounding, and time for displaying the results to the screen. Shell scripts were used as warranted to minimize delay between processing steps. A dash ('-') indicates that the solver was unable to solve the ELP on our system within 10 minutes (600 seconds).

Table 3. Experimental Results (total elapsed time in seconds for best run)



ELP Π	**|Ep(Π)|**	***ESmodels***	***ELPS***	***GISolver***	***ELPsolve***	***Wviews2***	***EP-ASP***	***PelpSolver***	***ELPsolve2***	***EHEX***	***selp***
`elig04`	8	<1	2	<1	<1	1	<1	9	<1	<1	<1
`elig06`	12	-	13	14	<1	16	<1	85	<1	1	<1
`elig08`	16	-	-	-	36	350	<1	-	<1	2	1
`elig16`	32	-	-	-	-	-	15	-	4	5	8
`yale2`	6	-	<1	-	<1	-	<1	3	<1	3	1
`yale4`	10	-	1	-	<1	-	<1	11	<1	4	102
`yale5`	17	-	33	-	<1	-	<1	356	<1	48	-
`yale8`	34	-	-	-	-	-	3	-	25	-	-
`art1`	6	262	<1	<1	<1	4	<1	6	<1	<1	<1
`art2`	12	-	14	-	<1	-	<1	-	<1	2	2
`art4`	24	-	-	-	<1	-	1	-	<1	3	43
`art5`	30	-	-	-	<1	-	2	-	<1	8	294

The results indicate that the use of brave and cautious entailment by *ELPsolve2*, *EP-ASP*, and *EHEX* have the potential to improve performance dramatically for input similar to the *elig*`NN` programs. The approach used by *selp* also appears quite effective for programs of this type. For the *yale*`N` programs, results are skewed in favor of solvers with special optimizations for conformant planning problem encodings. It is also apparent that solvers supporting ES2016 have an advantage for the *art*`N` programs as solutions are found early, i.e., when all or most of the epistemic negations are true. Although we included *GISolver* and *PelpSolver* in our tests, we note that these solvers were designed for languages where Epistemic Specifications is but a subset.

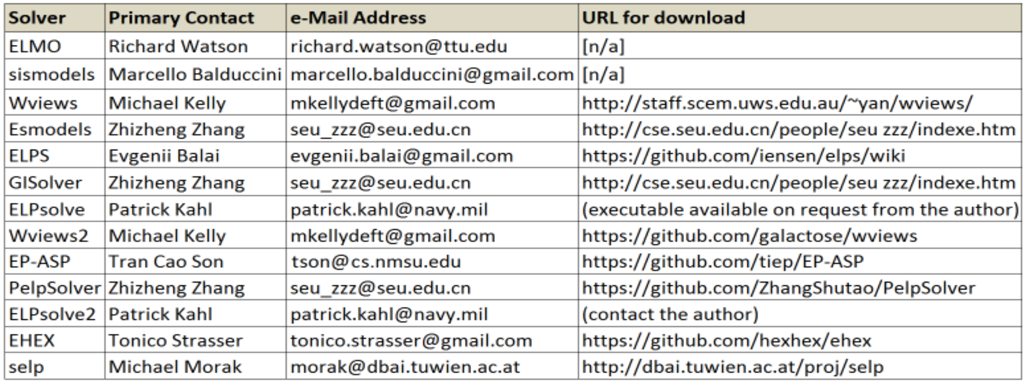
5 Conclusions

Work on epistemic logic program solvers is clearly active. We have reviewed a number of solvers, most of which were developed within the last five years. Significant improvements in both performance and the ability to solve harder (w.r.t. the number of epistemic negations) programs are evident. The development of efficient and easier to use solvers have allowed experimentation with different problems, syntax, and semantics, and have in fact been useful to reveal and assess different consequences of language variants.

Other ideas for improving performance include the use of world view constraints, which have the potential to reduce the number of epistemic negations (Kahl and Leclerc 2018). For many solvers the search space of epistemic negations can be partitioned into mutually exclusive (independent) subsets providing an opportunity for parallelization.

The "invalid guess" filter mentioned in the discussion of *ELPsolve2* applies to any ELP solver (and may already be implemented in other solvers). Yet another idea is to construct a "hybrid" solver which runs multiple different solvers in parallel (e.g., EP-ASP and EHEX), terminating further computation once any solver completes with the required solution.

Table 4. ELP Solver Contact and Download Information



Solver	Primary Contact	e-Mail Address	URL for download
ELMO	Richard Watson	richard.watson@ttu.edu	[n/a]
sismodels	Marcello Balduccini	marcello.balduccini@gmail.com	[n/a]
Wviews	Michael Kelly	mkellydeft@gmail.com	http://staff.scem.uws.edu.au/~yan/wviews/
Esmodels	Zhizheng Zhang	seu_zzz@seu.edu.cn	http://cse.seu.edu.cn/people/seu zzz/indexe.htm
ELPS	Evgenii Balai	evgenii.balai@gmail.com	https://github.com/iensen/elps/wiki
GISolver	Zhizheng Zhang	seu_zzz@seu.edu.cn	http://cse.seu.edu.cn/people/seu zzz/indexe.htm
ELPsolve	Patrick Kahl	patrick.kahl@navy.mil	(executable available on request from the author)
Wviews2	Michael Kelly	mkellydeft@gmail.com	https://github.com/galactose/wviews
EP-ASP	Tran Cao Son	tson@cs.nmsu.edu	https://github.com/tiep/EP-ASP
PelpSolver	Zhizheng Zhang	seu_zzz@seu.edu.cn	https://github.com/ZhangShutao/PelpSolver
ELPsolve2	Patrick Kahl	patrick.kahl@navy.mil	(contact the author)
EHEX	Tonico Strasser	tonico.strasser@gmail.com	https://github.com/hexhex/ehex
selp	Michael Morak	morak@dbai.tuwien.ac.at	http://dbai.tuwien.ac.at/proj/selp

References

Balai, E. 2014. ELPS. Texas Tech University. URL: `https://github.com/iensen/elps`.

Balai, E., Gelfond, M., and Zhang, Y. 2013. SPARC–Sorted ASP with consistency restoring rules. *CoRR abs/1301.1386*.

Balai, E. and Kahl, P. 2014. Epistemic logic programs with sorts. In *Proceedings of ASPOCP 2014*, D. Inclezan and M. Maratea, Eds.

Baral, C. and Gelfond, M. 1994. Logic programming and knowledge representation. *Journal of Logic Programming, Supplement 1—Special Edition: Ten Years of Logic Programming, 19/20*, 73–148.

Bichler, M. 2015. *Optimizing Non-Ground Answer Set Programs via Rule Decomposition.* BSc thesis, TU Wien.

Bichler, M., Morak, M., and Woltran, S. 2016. The power of non-ground rules in answer set programming. *Theory and Practice of Logic Programming 16,* 5-6, 552–569.

Bichler, M., Morak, M., and Woltran, S. 2018. selp. TU Wien. URL: `http://dbai.tuwien.ac.at/proj/selp/`.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., and Schaub, T. 2013. ASP-Core-2 input language format, version 2.01c. URL: `https://www.mat.unical.it/aspcomp2013/ASPStandardization`.

Chen, W. and Warren, D. S. 1993. A goal-oriented approach to computing the well-founded semantics. *J. Log. Program. 17,* 2/3&4, 279–300.

Cui, R., Zhang, Z., and Zhao, K. 2012. ESParser: An epistemic specification grounder. In *Proceedings of CSSS 2012*, J. P. Delgrande and W. Faber, Eds. IEEE Computer Society CPS, 1823–1827.

DLVSYSTEM S.r.l. 2012. DLV. URL: `http://www.dlvsystem.com/dlv`.

Faber, W. and Woltran, S. 2011. Manifold answer-set programs and their applications. In *LPNMR—Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. LNCS, vol. 6565. Springer, 44–63.

Fariñas del Cerro, L., Herzig, A., and Su, E. I. 2015. Epistemic equilibrium logic. In *Proceedings of IJCAI 2015*, Q. Yang and M. Wooldridge, Eds. AAAI Press / IJCAI.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. 2017. Multi-shot ASP solving with clingo. *CoRR abs/1705.09811*.

Gelfond, M. 1991. Strong introspection. In *Proceedings of AAAI 1991*, T. L. Dean and K. McKeown, Eds. Vol. 1. AAAI Press / MIT Press, 386–391.

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

Gelfond, M. 2011. New semantics for epistemic specifications. In *Proceedings of LPNMR 2011*, J. P. Delgrande and W. Faber, Eds. LNCS, vol. 6645. Springer, 260–265.

HANKS, S. AND MCDERMOTT, D. 1987. Nonmonotonic logic and temporal projection. *Artificial Intelligence 33,* 3 (November), 379–412.

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

KAHL, P. T. 2014. *Refining the Semantics for Epistemic Logic Programs.* Ph.D. thesis, Texas Tech.

KAHL, P. T. AND LECLERC, A. P. 2018. Epistemic logic programs with world view constraints. In *Technical Communications of ICLP 2018* [to appear], A. D. Palù, P. Tarau, N. Saeedloei, and P. Fodor, Eds. OASIcs. Schloss Dagstuhl.

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

KAMINSKI, R. AND KAUFMANN, B. 2018. `clingo, gringo, claspd`. University of Potsdam. URL: `https://potassco.org`.

KELLY, M. 2007. *Wviews: A Worldview Solver for Epistemic Logic Programs.* Honour's thesis, University of Western Sydney.

KELLY, M. 2007, 2018. `Wviews`. 2007 version: University of Western Sydney AI Research Group. URL: `http://staff.scem.uws.edu.au/~yan/Wviews.html`. 2018 version: Michael Kelly. URL: `https://github.com/galactose/wviews`.

LE, T. AND SON, T. C. 2017. `EP-ASP`. NMSU. URL: `https://github.com/tiep/EP-ASP`.

LECLERC, A. P. AND KAHL, P. T. 2016. `ELPSolve` (version 1.0). SPAWAR Systems Center Atlantic. Available on request; send e-mail to `patrick.kahl@navy.mil`.

REDL, C., SHÜLLER, P., ET AL. 2017. `DLVHEX2`. TU Wien. URL: `https://github.com/hexhex/core`.

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

SIMONS, P. 2000. Extending and Implementing the Stable Model Semantics. Ph.D. thesis, Helsinki University of Technology.

STRASSER, A. 2018. `EHEX`. TU Wien. URL: `https://github.com/hexhex/ehex`.

SU, E. I. 2015. *Extensions of Equilibrium Logic by Modal Concepts.* Ph.D. thesis, University of Toulouse.

SYRJÄNEN, T. AND SIMONS, P. 2010. `Lparse,Smodels`. Aalto University (previously Helsinki University of Technology). URL: `http://www.tcs.hut.fi/Software/smodels`.

TRUSZCZYŃSKI, M. 2011. Revisiting epistemic specifications. In *LPNMR—Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. LNCS, vol. 6565. Springer, 315–333.

ZHANG, Y. 2006. Computational properties of epistemic logic programs. In *Proceedings of KR 2006*, P. Doherty, J. Mylopoulos, and C. A. Welty, Eds. AAAI Press, 308–317.

ZHANG, Y. AND ZHANG, Y. 2017a. Epistemic specifications and conformant planning. In *Proceedings of KnowProS 2017*, R. Barták, T. L. McCluskey, and E. Pontelli, Eds.

ZHANG, Z., WANG, B., AND ZHANG, S. 2015a. `GISolver`. Southeast University. URL: `http://cse.seu.edu.cn/people/seu_zzz/indexe.htm`.

ZHANG, Z., WANG, B., AND ZHANG, S. 2015b. Logic programming with graded introspection. In *Proceedings of ASPOCP 2015*, D. Inclezan and M. Maratea, Eds.

ZHANG, Z. AND ZHANG, S. 2017b. `PelpSolver`. Southeast University. URL: `https://github.com/ZhangShutao/PelpSolver`.

ZHANG, Z. AND ZHAO, K. 2014. ESmodels: An epistemic specification solver. *CoRR abs/1405.3486*.

ZHANG, Z., ZHAO, K., AND CUI, R. 2013a. ESmodels: An inference engine of epistemic specifications. In *Proceedings of ICTAI 2013*, J. Luo, Ed. IEEE, 769–774.

ZHANG, Z., ZHAO, K., AND CUI, R. 2013b. `ESParser` and `ESmodels`. Southeast University. URL: `http://cse.seu.edu.cn/people/seu_zzz/indexe.htm`.