

Epistemic Logic Programs with Sorts

Evgenii Balai and Patrick Kahl

Department of Computer Science, Texas Tech University
{evgenii.balai,patrick.kahl}@ttu.edu

Abstract. We introduce the notion of an *epistemic logic program with sorts*—a program written in the language of Epistemic Specifications (an extension of ASP) with the addition of a sorted signature. This extends recent efforts to define a logic programming language that includes the means for explicitly specifying the domains of predicate parameters. An algorithm and its implementation as a solver for epistemic logic programs with sorts is also discussed.

Keywords: Epistemic Specifications, epistemic logic programs, sorted signature, rule safety, ASP, logic programming, non-monotonic reasoning, knowledge representation

1 Introduction

An *epistemic logic program* is a program written in the language of *Epistemic Specifications*, an extension of the language of answer set programming (ASP) that allows for introspective reasoning through the use of modal operators K and M. The language of Epistemic Specifications was introduced by Gelfond in the early 1990s[14, 15] after observing the need for a more powerful form of introspective reasoning than that offered by ASP alone. The semantics of an epistemic logic program was given in terms of its *world view*—a collection of sets of literals (*belief sets*), analogous to the answer sets of an ASP program. In 2011[16], Gelfond proposed a change to the semantics in a preliminary attempt to avoid certain unintended world views. Kahl, Watson, Gelfond, and Zhang[20] continued this work in the hopes of finding a semantics that was satisfactory with respect to intuition and modeling of problems for which the added power of Epistemic Specifications showed promise.

Concurrent with this latter work was an effort by Balai, Gelfond, and Zhang[4, 5] to enhance ASP by providing a framework for specifying sorts for the parameters of predicates in a language they called *SPARC*. This promoted better program design, helped to eliminate certain common errors (e.g., misspellings), and encouraged development of rules specific to a particular domain without the concern for rule safety that has been associated with variable terms by popular ASP solvers (e.g., `clingo`[23], `DLV`[11]). With this in mind, consider the following epistemic logic program using the syntax and semantics of [20]:

$$\begin{aligned} p(X) &\leftarrow \text{not K } q(X). \\ r(a). \end{aligned}$$

with world view $\{\{r(a), p(a)\}\}$. The first rule of this program is considered unsafe. Adding the seemingly unrelated fact $r(b)$ introduces object constant b to the program

signature, thereby yielding a new ground instance of the first rule: $p(b) \leftarrow \text{not } K \ q(b)$. This results in adding $p(b)$ to the belief set. We consider this new result to be unexpected and easily missed by a programmer. Similar undesirable artifacts can be observed from programs that have subjective literals formed by M . Giving explicit domains for the parameters of predicates resolves these concerns.

In this paper, we combine these recent efforts by providing within the language of Epistemic Specifications a framework for specifying predicate parameter domains. We call a program written in this new language an *epistemic logic program with sorts (ELPS)*. In addition to the benefits a sorted signature provides to the programmer, it also allows for a straightforward algorithm for computing world views. We describe such an algorithm—one that can take advantage of existing state-of-the-art ASP solvers—and give a synopsis of test results from our implementation. We conclude with a discussion of related work and a brief summary.

2 Syntax and Semantics

In general, the syntax and semantics of the language of Epistemic Specifications follow that of ASP with the notable addition of modal operators K and M , plus the new notions of *belief sets* (analogous to answer sets) and a *world view* (a collection of belief sets, analogous to the set of all answer sets of a program). We assume familiarity with logic programming—ASP in particular. For a good introduction, see any of [7, 18, 13, 17, 1].

2.1 A New Meaning for Rules with Variables

Under the original semantics of both ASP and Epistemic Specifications, the signature of a program is typically implied by the symbols that occur, with universal domain for all predicate parameters. Additionally, a rule containing a variable is considered a shorthand for all its ground instantiations. Thus, for the following program

$$\begin{aligned} p(a) \circ \text{r } p(b). \\ q(c). \\ r(X) \leftarrow M \ p(X), \text{ not } q(X). \end{aligned}$$

the last rule is simply a shorthand for the rules

$$\begin{aligned} r(a) \leftarrow M \ p(a), \text{ not } q(a). \\ r(b) \leftarrow M \ p(b), \text{ not } q(b). \\ r(c) \leftarrow M \ p(c), \text{ not } q(c). \end{aligned}$$

Now consider the same initial program, but with a sorted signature explicitly given that specifies the domain for each predicate parameter per the following table:

PREDICATE SYMBOL	PARAMETER DOMAIN
p	$\{a, b\}$
q	$\{a, b, c\}$
r	$\{a, c\}$

Note that the parameter domains for p , q , and r differ. One might ask whether the last rule of the program is still valid, and if so, what its ground instantiations should be. We use the same convention as in [5] where the ground instantiations of a rule with variables

are only those where each variable instantiation maintains predicate parameter values within their declared domains. Thus, the last rule is now a shorthand for only one rule

$$r(a) \leftarrow \mathbf{M} \, p(a), \text{ not } q(a).$$

since any other instantiation for the variable X would not result in values for the parameters of p , q , and r that would be in all their respective domains.

2.2 Syntax of an ELPS

A program written in the language of Epistemic Specifications is called an *epistemic logic program*. Syntactically, the difference between an epistemic logic program and an *epistemic logic program with sorts* is that the latter includes explicit sort definitions and predicate declarations in addition to the program rules. A *sort* is a (non-empty) set of ground terms intended for use in defining the domain of predicate parameters.

Various means for defining sorts have been proposed. In this paper, we adopt the same framework as that of SPARC[5] that requires a preface before the program rules to define the sorts and provide predicate declarations.

The first section of the framework begins with the keyword `sorts` followed by sort definitions of the form

$$\text{sort_name} = \text{sort_expression}.$$

where *sort_name* begins with the symbol `#` followed by a unique identifier starting with a lower-case letter, and *sort_expression* is one of the following forms

- $\{t_1, \dots, t_n\}$
- $f(\text{sort_name}_1, \dots, \text{sort_name}_n)$
- $\text{sort_name}_1 \otimes \dots \otimes \text{sort_name}_n$

where $n > 0$, each t is a ground term, f is a function symbol, and $\otimes$ denotes a set operator (union, intersection, or difference) where the set operations result in a non-empty set.

The next section begins with the keyword `predicates` followed by predicate declarations of the form

$$\text{predicate_name}(\text{sort_name}_1, \dots, \text{sort_name}_n).$$

where $n \geq 0$. Every predicate in the program must be declared uniquely in this section before being used in a program rule. In addition to explicitly providing the domain for each parameter of a predicate, this allows for static type-checking of rules to discover misspellings and arity mismatches.

The final section begins with the keyword `rules` followed by the rules of the program. Each rule is of the form

$$\ell_1 \text{ or } \dots \text{ or } \ell_k \leftarrow g_1, \dots, g_m, \text{ not } g_{m+1}, \dots, \text{ not } g_n.$$

where $k \geq 0$, $m \geq 0$, $n \geq m$, each ℓ is a literal, and each g is either a literal or a *subjective literal* (i.e., a literal immediately preceded by $\mathbf{K}$ or $\mathbf{M}$).

2.3 Semantics

For a program to be considered valid, every occurrence of a ground atom in a program rule must match its predicate declaration with respect to predicate name, arity, and parameter values within the sort domain. Rules with variables are instantiated as previously

described. In short, for a valid ELPS, the net effect of the sorted signature is to restrict the grounding of its rules. The grounding of an ELPS results in a set of ground rules; hence, in the following definitions, a *ground ELPS* can be viewed simply as a ground epistemic logic program.

Definition 1. [*When a Subjective Literal Is Satisfied*]

Let W be a non-empty collection of consistent sets of ground literals.

- $K\ell$ is satisfied by W iff $\forall A \in W : \ell \in A$.
- $\text{not } K\ell$ is satisfied by W iff $\exists A \in W : \ell \notin A$.
- $M\ell$ is satisfied by W iff $\exists A \in W : \ell \in A$.
- $\text{not } M\ell$ is satisfied by W iff $\forall A \in W : \ell \notin A$.

Definition 2. [*Modal Reduct/World View/Inconsistent Program*]

Let Π be a ground ELPS and W be a non-empty collection of consistent sets of ground literals in the language of Π . We denote by Π^W —the *modal reduct* of Π with respect to W —the ASP program (with nested expressions¹) obtained from Π by replacing/removing subjective literals and deleting rules according to the following table.

SUBJECTIVE LITERAL	IF SATISFIED BY W	IF NOT SATISFIED BY W
$K\ell$	replace $K\ell$ with ℓ	delete the rule
$\text{not } K\ell$	remove $\text{not } K\ell$	replace $\text{not } K\ell$ with $\text{not } \ell$
$M\ell$	remove $M\ell$	replace $M\ell$ with $\text{not not } \ell$
$\text{not } M\ell$	replace $\text{not } M\ell$ with $\text{not } \ell$	delete the rule

W is a *world view* of Π iff $W = \text{AS}(\Pi^W)$ where $\text{AS}(\Pi^W)$ is the set of all answer sets of Π^W . If no such W exists (i.e., Π has no world view) then Π is said to be *inconsistent*.

2.4 Example Programs

To illustrate the syntax and semantics of an ELPS, we provide several example programs with their ground versions, candidate world views, modal reducts, answer sets of modal reducts, and world views. We will use the syntax of the ELPS solver[2] which is similar to the syntax used by ASP solver `clingo`[23] but with the addition of the sorted signature preface, plus $K\$$ and $M\$$ used for modal operators K and M , respectively.

ELPS Π	Ground Program Π_{grd}	$W, \Pi_{\text{grd}}^W, \text{AS}(\Pi_{\text{grd}}^W), \text{World view?}$
<i>Example 1.</i> sorts % no sorts predicates p(). rules p:-K\$p.	p:-K\$p.	$W_1 = \{\{\}\}$ $\Pi_{\text{grd}}^{W_1} : (\text{empty program})$ $\text{AS}(\Pi_{\text{grd}}^{W_1}) = \{\{\}\}$ Is W_1 a world view? YES
		$W_2 = \{\{p\}\}$ $\Pi_{\text{grd}}^{W_2} : p:-p.$ $\text{AS}(\Pi_{\text{grd}}^{W_2}) = \{\{\}\}$ Is W_2 a world view? NO

¹Nested expressions of the form $\text{not not } \ell$ are defined in [22].

Example 2.

ELPS Π	Ground Program Π_{grd}	$W, \Pi_{\text{grd}}^W, \text{AS}(\Pi_{\text{grd}}^W), \text{World view?}$
<pre> sorts % no sorts predicates p(). rules p:-M\$p.</pre>	$p:-M\$p.$	$W_1 = \{\{\}\}$ $\Pi_{\text{grd}}^{W_1} : p:-\text{not not } p.$ $\text{AS}(\Pi_{\text{grd}}^{W_1}) = \{\{\}, \{p\}\}$ Is W_1 a world view? NO
		$W_2 = \{\{p\}\}$ $\Pi_{\text{grd}}^{W_2} : p.$ $\text{AS}(\Pi_{\text{grd}}^{W_2}) = \{\{p\}\}$ Is W_2 a world view? YES

Example 3.

ELPS Π	Ground Program Π_{grd}	$W, \Pi_{\text{grd}}^W, \text{AS}(\Pi_{\text{grd}}^W), \text{World view?}$
<pre> sorts #s1={a,b,c}. #s2={a,c}. predicates p(#s1). q(#s2). r(#s2). rules p(a) p(b). q(X):-not M\$p(X). r(X):-M\$p(X).</pre>	<pre> p(a) p(b). q(a):-not M\$p(a). q(c):-not M\$p(c). r(a):-M\$p(a). r(c):-M\$p(c).</pre>	$W = \{\{p(a), q(c), r(a)\}, \{p(b), q(c), r(a)\}\}$ $\Pi_{\text{grd}}^W :$ $p(a) p(b).$ $q(c):-\text{not } p(c).$ $r(a).$ $r(c):-\text{not not } p(c).$ $\text{AS}(\Pi_{\text{grd}}^W) = \{\{p(a), q(c), r(a)\}, \{p(b), q(c), r(a)\}\}$ Is W a world view? YES

Example 4.

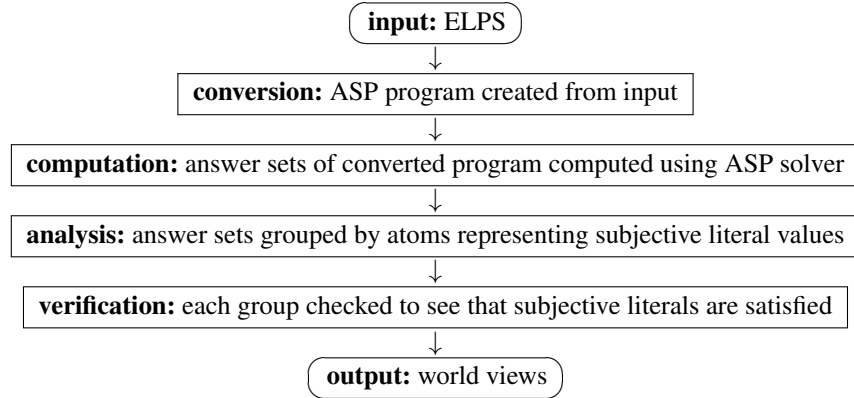
ELPS Π	Ground Program Π_{grd}	$W, \Pi_{\text{grd}}^W, \text{AS}(\Pi_{\text{grd}}^W), \text{World view?}$
<pre> sorts #s1={a,b}. #s2=h(#s1). predicates p(#s1). q(#s2). r(#s1). rules p(a) p(b). q(h(X)):-M\$p(X). r(X):-K\$q(h(X)).</pre>	<pre> p(a) p(b). q(h(a)):-M\$p(a). q(h(b)):-M\$p(b). r(a):-K\$q(h(a)). r(b):-K\$q(h(b)).</pre>	$W = \{\{q(h(a)), q(h(b)), p(a), r(a), r(b)\}, \{q(h(a)), q(h(b)), p(b), r(a), r(b)\}\}$ $\Pi_{\text{grd}}^W :$ $p(a) p(b).$ $q(h(a)).$ $q(h(b)).$ $r(a):-q(h(a)).$ $r(b):-q(h(b)).$ $\text{AS}(\Pi_{\text{grd}}^W) = \{\{q(h(a)), q(h(b)), p(a), r(a), r(b)\}, \{q(h(a)), q(h(b)), p(b), r(a), r(b)\}\}$ Is W a world view? YES

3 Algorithm

Just as solving an ASP program has been equated with finding its answer sets, solving an ELPS is equated with finding its world views. We therefore propose an algorithm that takes an ELPS as its input and outputs the world views (if any) of the program.

The algorithm follows a generate and test approach. First, we generate candidate solutions. This is done by creating an ASP program from the input ELPS that emulates the possible modal reducts with respect to every combination of truth assignments to the distinct ground subjective literals of the ELPS. The answer sets of the ASP program are then computed and grouped into candidate world views based on associated subjective literal truth values. Finally, the candidates are checked by verifying that the associated subjective literals are all satisfied. Validated candidates are world views of the ELPS.

Outline of Our Algorithm for Finding the World Views of an ELPS



The algorithm is designed to take advantage of an existing state-of-the-art ASP solver. Details concerning the algorithm from which this is based are given in [20, 19]. The first (conversion) step is a combination of that given in [20] and that used by SPARC[5, 3]. When compared to the conversion described in [20], the one given herein is guaranteed to have only safe rules² and it can therefore be run directly with an existing ASP solver (e.g., Clingo[23]).

In the following algorithm, $k1_l$ is to be intuitively understood as “ $K\ l$ is true” and $k0_l$ as “ $K\ l$ is false.” Respectively, $m1_l$ corresponds to “ $M\ l$ is true” and $m0_l$ corresponds to “ $M\ l$ is false.”

²Note that our definitions do not include built-in predicates such as $<$, $=$, etc. However, if built-ins are allowed, unsafe rules are possible; e.g. the rule $p \leftarrow X < 2$ will not be made safe in our conversion because the built-in predicate $<$ would not have its parameters limited by sorts in the program preface.

Algorithm 1: Step 1 [Conversion from ELPS to ASP Program]

Input: ELPS program Π
Output: ASP program Π_{ASP}

```

1  $\Pi_{\text{ASP}} := \emptyset$ .
2 for each distinct sort name  $s$  in the sort definitions of  $\Pi$  do
3    $\Pi_{\text{ASP}} := \Pi_{\text{ASP}} \cup \{s(t) \mid t \text{ is an element of the sort } s \text{ as defined in } \Pi\}$ .
4 for each rule  $R$  in  $\Pi$  do
5    $S_R := \{s_i(t_i) \mid 1 \leq i \leq n, s_i \text{ is a sort name occurring in a predicate}$ 
      $\text{declaration } p(s_1, \dots, s_n) \text{ of } \Pi, \text{ and } t_i \text{ is a variable term (or term containing a}$ 
      $\text{variable) appearing as a predicate parameter of a [possibly negated] atom}$ 
      $p(t_1, \dots, t_n) \text{ appearing in } R\}$ .
6   Let  $\text{sort\_atoms}(R)$  denote the comma-delimited list of elements in  $S_R$ .
7   Let  $R'$  be the rule resulting from adding  $\text{sort\_atoms}(R)$  to the body of  $R$ .
8   Let  $\ell'$  denote the string obtained from literal  $\ell$  by replacing  $\neg$  with 0. (In the
     case where  $\neg$  is not present, strings  $\ell$  and  $\ell'$  are identical.)
9   for each subjective literal of the form  $K \ell$  or  $\text{not } K \ell$  in  $R'$  do
10    Replace  $\text{not } K \ell$  with  $\neg k\_ \ell'$  in  $R'$ .
11    Replace  $K \ell$  with  $\text{not } \neg k\_ \ell', \ell$  in  $R'$ .
12    Add the following rules to  $\Pi_{\text{ASP}}$ :
13       $k1\_ \ell' \leftarrow \text{not } k0\_ \ell', \text{sort\_atoms}(R)$ .
14       $k0\_ \ell' \leftarrow \text{not } k1\_ \ell', \text{sort\_atoms}(R)$ .
15       $\neg k\_ \ell' \leftarrow k0\_ \ell'$ .
16       $\neg k\_ \ell' \leftarrow k1\_ \ell', \text{not } \ell$ .
17   for each subjective literal of the form  $M \ell$  or  $\text{not } M \ell$  in  $R'$  do
18    Replace  $\text{not } M \ell$  with  $\text{not } m\_ \ell'$  in  $R'$ .
19    Replace  $M \ell$  with  $m\_ \ell'$  in  $R'$ .
20    Add the following rules to  $\Pi_{\text{ASP}}$ :
21       $m1\_ \ell' \leftarrow \text{not } m0\_ \ell', \text{sort\_atoms}(R)$ .
22       $m0\_ \ell' \leftarrow \text{not } m1\_ \ell', \text{sort\_atoms}(R)$ .
23       $m\_ \ell' \leftarrow m1\_ \ell'$ .
24       $m\_ \ell' \leftarrow m0\_ \ell', \text{not not } \ell$ .
25   Add  $R'$  to  $\Pi_{\text{ASP}}$ .
26 return  $\Pi_{\text{ASP}}$ 

```

Pseudocode for the first step of the algorithm—conversion from an ELPS to an ASP program—is given above. (Any type-checking should be done before or concurrently.) The basic idea is to add the appropriate sort atoms to the bodies of rules containing variables to ensure safety and proper grounding, and to replace subjective literals with fresh ASP literals while adding rules that define the fresh literals. For simplicity, we assume that no predicate in the input ELPS starts with prefix $k_$, $k0_$, $k1_$, $m_$, $m0_$, or $m1_$. We refer to literals formed using these prefixes as *k-/m-literals*, respectively. It is also assumed that the ASP solver used in the computation step accepts rules with $\text{not not } \ell$ in the body as defined in [22]. An alternative to using $\text{not not } \ell$ is to instead use $\text{not } n_ \ell'$ where $n_ \ell'$ is another fresh literal, and also add the rule $n_ \ell' \leftarrow \text{not } \ell$. Fresh literals are understood to be removed from the algorithm's output.

The remaining steps of the algorithm follow. These steps are essentially the same as those given in [20] except that step 2 is simplified due to the use of a sorted signature.

-
2. Compute the answer sets of Π_{ASP} with an appropriate ASP solver.
 3. Group the answer sets of Π_{ASP} by common k-/m-atoms of the form $k0_l'$, $k1_l'$, $m0_l'$, and $m1_l'$. Each group W of answer sets is said to *represent* (i.e., if its k-/m-literals and sort atoms are removed) a *candidate world view* of Π .
 4. For each W representing a candidate world view of Π , check that the following conditions are met for all its k-/m-atoms:
 - (a) if $k1_l'$ is in the sets of W , then l is in every set of W ;
 - (b) if $k0_l'$ is in the sets of W , then l is missing from at least one set of W ;
 - (c) if $m1_l'$ is in the sets of W , then l is in at least one set of W ; and
 - (d) if $m0_l'$ is in the sets of W , then l is missing from every set of W .

$W_{\setminus \text{KM} \setminus \text{sorts}}$ (W with k-/m-literals and sort atoms removed) is a *world view* of Π if conditions (a)–(d) are met.

The correctness of the algorithm follows from Theorem 1 of [5] and Theorems 1 and 2 of [19].

4 Implementation and Test Results

The algorithm was implemented (see [2]) using the Java programming language, working off a previous implementation of SPARC. This made coding fairly easy since the same framework for a sorted signature is shared by both ELPS and SPARC. The solver is a loosely coupled system that uses Potassco’s `clingo` for finding the answer sets of the ASP program created in step 1 of the algorithm. The choice of `clingo` was appropriate due to its excellent performance and direct support for doubly default-negated literals (i.e., nested expressions of the form `not not  $\ell$`). Other solvers could be used with (possibly) minor modifications made for differences in syntax/supported features.

Test results showed that, provided the number of ground subjective literals is small, performance is reasonably good—even for this loosely coupled proof-of-concept solver. It is noted, however, that the number of candidate world views generated and checked is exponential in the number of distinct ground subjective literals using the algorithm presented. For many of the test programs we ran, the independence of various groups of subjective literals was clear, hinting at a possible approach for improving the algorithm for certain classes of programs.

5 Related Work

Yan Zhang[27] investigated the computational properties of epistemic logic programs which led him to the development of an algorithm for computing world views. This effort culminated in the development of a solver known as `Wviews`[26] (developed by Michael Kelly). `Wviews` relies on the ASP solver `DLV`[11] and is based on the 1994 version of Epistemic Specifications[15, 8].

Truszczyński[25] offered alternative semantics to Gelfond’s original version of Epistemic Specifications[14] by using Kleene’s three-valued logic[21]. He also introduced a semantic framework using two-value logic to define other versions of Epistemic Specifications. He went on to derive complexity results that suggest the language is strictly more expressive than ASP. Finally, he elaborated on the use of the language for modeling problems that require meta-reasoning.

Faber and Woltran[12] presented a framework for combining ASP programs in a manner that allows consequences of programs to be used as input to other programs. They presented a technique for compiling such a framework into a single ASP program using the concept of manifold programs. A *manifold program* is an expanded encoding of an ASP program that allows for reasoning about the consequences of its answer sets from within the encoding, thus providing a limited capacity for introspective reasoning similar to that provided by modal operators in the language of Epistemic Specifications. Their approach relies on the ASP extension known as *weak constraints*[9]. For certain epistemic logic programs, the methods described by Faber and Woltran may provide an efficient method for finding world views.

In 2011, Gelfond[16] proposed an update to the syntax and semantics of Epistemic Specifications. His new reduct definition solved the problem of unintended world views associated with recursion through modal operator K ; however, unintended world views associated with recursion through M remained.

Cui, Z. Zhang, and Zhao[10] investigated the problem of grounding an epistemic specification. That work culminated in the development of a grounder known as `ESParser`. Their more recent efforts[28] include the development of an associated solver called `ESmodels`[24] that relies on ASP solver `claspD`[23]. The approach used in the optimization they refer to as “EFFICIENT ESMODELS” may be applicable to ELPS as well. `ESmodels` is based on Gelfond’s 2011 version of Epistemic Specifications[16].

Balai, Gelfond, and Yuanlin Zhang[4, 5] extended ASP by adding a sorted signature, expanding the earlier work of Balduccini[6]. Included was an algorithm for applying the sorted signature to rules. This work culminated in the development of a solver/inference engine called `SPARC`[3]. Our paper extends their work to the language of Epistemic Specifications.

Gelfond, Kahl, Watson, and Yuanlin Zhang[20] continued Gelfond’s 2011 work[16] by proposing yet another update to the syntax and semantics of Epistemic Specifications. That work includes the modal reduct used in this paper, as well as an algorithm for finding world views that is the primary basis for the one described herein. The new semantics offers a solution to the problem of unintended world views that had been associated with recursion through modal operators. The paper also includes a number of examples, as well as a framework for a conformant planner using Epistemic Specifications.

6 Summary

We believe adding a sorted signature to epistemic logic programs is an important enhancement to the language of Epistemic Specifications. Benefits include the fact that a sorted signature can:

- eliminate rule safety concerns;

- help avoid common errors by allowing static type-checking of predicates in rules;
- help avoid belief sets with unintended/nonsensical term relations;
- promote good programming design practices; and
- allow for a straightforward algorithm for computing the world views of a program.

The proposed syntax and semantics are relatively easy to understand for someone already familiar with answer set programming. We emphasize the fact that the semantic effect of using a sorted signature with a valid program is limited to its grounding. For those familiar with Epistemic Specifications, the modal reduct shown here is different from the one given by Gelfond in 2011[16] and previously[15, 8]. For more information on the new proposed semantics for epistemic logic programs, see [20, 19].

The algorithm described herein shows that solving an ELPS can be achieved by a method that includes the use of an existing state-of-the-art ASP solver, thereby avoiding “reinventing the wheel” with respect to some of the more difficult tasks. Our implementation and testing of that algorithm demonstrates that using ELPS to model certain problems can be a practical method of problem solving.

Acknowledgments

We would like to thank the members of the Texas Tech University Knowledge Representation Lab who provided incite, support, and technical/editorial comments, especially Michael Gelfond, Yulia Kahl, Richard Watson, Edward Wertz, and Yuanlin Zhang.

References

1. ASP Standardization Working Group: ASP-Core-2 input language format (2013), available at <https://www.mat.unical.it/aspcomp2013/ASPStandardization>
2. Balai, E.: ELPS, Texas Tech University, software & documentation available for download at <https://github.com/iensen/elps/wiki/>
3. Balai, E.: SPARC, Texas Tech University, software & documentation available for download at <https://github.com/iensen/sparc/wiki/>
4. Balai, E., Gelfond, M., Zhang, Y.: *SPARC* – sorted ASP with consistency restoring rules. CoRR abs/1301.1386 (2013)
5. Balai, E., Gelfond, M., Zhang, Y.: Towards answer set programming with sorts. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. Lecture Notes in Artificial Intelligence, vol. 8148. Springer-Verlag (2013)
6. Balduccini, M.: Modules and signature declarations for A-Prolog: Progress report. In: Software Engineering for Answer Set Programming Workshop (SEA07) (2007)
7. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2003)
8. Baral, C., Gelfond, M.: Logic programming and knowledge representation. Journal of Logic Programming Supplement 1—Special Edition: Ten Years of Logic Programming, 19/20, 73–148 (1994)
9. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Transactions on Knowledge and Data Engineering 12(5), 845–860 (2000)
10. Cui, R., Zhang, Z., Zhao, K.: ESParser: An epistemic specification grounder. In: Delgrande, J.P., Faber, W. (eds.) CESS-12. pp. 1823–1827. IEEE Computer Society CPS (2012)

11. DLVSYSTEM s.r.l.: DLV (2012), software & documentation available for download at <http://www.dlvsystem.com/>
12. Faber, W., Woltran, S.: A framework for programming with module consequences. In: Vos, M.D., Schaub, T. (eds.) SEA-09. pp. 34–48 (2009)
13. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
14. Gelfond, M.: Strong introspection. In: Dean, T.L., McKeown, K. (eds.) AAAI-91. vol. 1, pp. 386–391. AAAI Press / The MIT Press (1991)
15. Gelfond, M.: Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence* 12(1–2), 89–116 (1994)
16. Gelfond, M.: New definition of epistemic specifications. In: Delgrande, J.P., Faber, W. (eds.) LPNMR-11. *Lecture Notes in Computer Science*, vol. 6645, pp. 260–265. Springer (2011)
17. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press (2014)
18. van Harmelen, F., Lifschitz, V., Porter, B. (eds.): Handbook of Knowledge Representation. Foundations of Artificial Intelligence, Elsevier (2008)
19. Kahl, P.: Refining the Semantics for Epistemic Logic Programs. PhD dissertation, Texas Tech University, Lubbock, TX, USA (May 2014)
20. Kahl, P., Watson, R., Gelfond, M., Zhang, Y.: A refinement of the language of epistemic specifications. In: Inclezan, D., Maratea, M. (eds.) ASPOCP 2014. CoRR (2014)
21. Kleene, S.C.: Introduction to Metamathematics. University Series in Higher Mathematics, D. Van Nostrand, Princeton, NJ (1952)
22. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25, 369–389 (1999)
23. Potassco: `clasp`, `claspD`, `clingo`, `gringo` (2014), University of Potsdam, software & documentation available for download at <http://potassco.sourceforge.net/>
24. Southeast University: `ESparser`, `ESmodels` (2013), software & documentation available for download at http://cse.seu.edu.cn/people/seu/_zzz/index.htm
25. Truszczyński, M.: Revisiting epistemic specifications. In: Balduccini, M., Son, T.C. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. *Lecture Notes in Computer Science*, vol. 6565, pp. 315–333. Springer (2011)
26. University of Western Sydney AI Research Group: `Wviews` (2007), UWS AI Research Group (Intelligent Systems Laboratory), software & documentation available for download at <http://staff.scm.uws.edu.au/%7eyan/Wviews.html>
27. Zhang, Y.: Computational properties of epistemic logic programs. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) KR-06. pp. 308–317. AAAI Press (2006)
28. Zhang, Z., Zhao, K., Cui, R.: ESmodels: An inference engine of epistemic specifications. In: Luo, J. (ed.) ICTAI 2013. IEEE (2013)