Artificial Intelligence 233 (2016) 84–121

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Domain expansion for ASP-programs with external sources ☆

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl *

Institut für Informationssysteme, Technische Universität Wien, Favoritenstrasse 9-11, A-1040 Vienna, Austria

ARTICLE INFO

Article history:
Received 2 September 2014
Received in revised form 21 December 2015
Accepted 2 January 2016
Available online 7 January 2016

Keywords:
Answer set programming
Knowledge representation formalisms
Nonmonotonic reasoning
External source access
Grounding
Computational logic

ABSTRACT

Answer set programming (ASP) is a popular approach to declarative problem solving which for broader usability has been equipped with external source access. The latter may introduce new constants to the program (known as *value invention*), which can lead to infinite answer sets and non-termination; to prevent this, syntactic safety conditions on programs are common which considerably limit expressiveness (in particular, recursion). We present *liberal domain-expansion (lde) safe programs*, a novel generic class of ASP programs with external source access and value invention that enjoy finite restrictability, i.e., equivalence to a finite ground version. They use *term bounding functions* as a parametric notion of safety, which can be instantiated with syntactic, semantic or combined safety criteria; this empowers us to generalize and integrate many other notions of safety from the literature, and modular composition of criteria makes future extensions easy. Furthermore, we devise a grounding algorithm for lde-safe programs which in contrast to traditional algorithms can ground any such program directly without the need for program decomposition. While we present our approach on top of a proposed formalism in order to make the formalization precise, the general concepts carry over to related formalisms and important special cases as well. An experimental evaluation of lde-safety on various applications confirms the practicability of our approach.

1. Introduction

Answer Set Programming (ASP) is a declarative programming approach which due to expressive and efficient systems like CLASP [29],[1] DLV [40],[2] and SMODELS [48][3] has been gaining popularity in several application areas, and in particular in artificial intelligence [8]. A problem at hand is represented by a set of rules (an ASP program) such that its models, called *answer sets*, encode the solutions to the problem. Compared to the similar approach of SAT solving, the rules might contain variables as a shortcut for all ground instances, transitive closure can be readily expressed as well as negation as failure; further extensions including optimization constructs, aggregates, preferences and many other features have turned ASP into an expressive and powerful problem solving tool.

Recent developments in computing require access from ASP programs to external sources, as information is increasingly stored in different sources and formats, or because complex, specific tasks cannot be expressed directly or efficiently in

☆ Preliminary results of this work have been presented at AAAI 2013 [19], the 2nd GTTV workshop [18], INAP 2013 [22], and in [20].

* Corresponding author.
E-mail addresses: eiter@kr.tuwien.ac.at (T. Eiter), fink@kr.tuwien.ac.at (M. Fink), tkren@kr.tuwien.ac.at (T. Krennwallner), redl@kr.tuwien.ac.at (C. Redl).

[1] http://potassco.sourceforge.net.
[2] http://www.dlvsystem.com.
[3] http://www.tcs.hut.fi/Software/smodels.

http://dx.doi.org/10.1016/j.artint.2016.01.003
0004-3702/

the program itself. A prominent example are *DL-programs* [23], which integrate rules with description logic ontologies in such a way that queries to an ontology can be made in the rules; the formalism supports reasoning tasks which cannot be realized in ontologies alone, e.g., default classification. Another application with need for external access is planning in agent systems, which might require to import information from sensors and send commands back to agents, e.g. robots [47]; action or plan feasibility under physical or geometric constraints might be tested using special external libraries, etc. In other scenarios, the actions might be simple, but the planning domain is implicit in an external data structure; for example, in advanced route planning tasks for smart city applications [26] where Open Street Map data or some connection database may be used. Abstracting and accessing such data through an external interface is natural, as the data might not be fully accessible or too big to be simply added to the ASP program. Related to this is light-weight data access on the Web (e.g., XML, RDF [38] or other data repositories), which is getting more frequent and desired in complex applications, for instance in information integration; but like for a street map, a complete a priori data import is usually infeasible (in particular, in case of recursive data access). A concrete application scenario is, for instance, from the biomedical domain [27] where different online knowledge resources about genes, drugs and diseases are assessed in order to answer complex queries regarding their mutual relationships, e.g., for drugs that treat a certain disease while not targeting a particular gene.

Finally, ASP is a popular host for experimental implementations of logic-based AI formalisms; however, the expressive capability of ordinary ASP may not be sufficient to cater a particular formalism, or a direct encoding in ASP may be cumbersome; in this case, it is convenient if some condition checks can be outsourced to external computation. For example, implementations of Dung-style semantics for abstract argumentation [14] (this will be more discussed below) or of multi-context systems [7], fall in this class.

To cater for the need of external source access, HEX-*programs* [24] extend ASP with so-called *external atoms*, through which the user can couple any external data source with a logic program. Roughly, an external atom can be seen as an API that passes information from the program, given by predicate extensions, to an external source and receives output values of an (abstract) function which that source computes, implemented in whatever language. This powerful abstraction is, for specific data sources that amount to logical theories, related in spirit to *SAT modulo theories* [4] but more versatile due to the possibility to use variables. It also generalizes a number of ASP extensions, among them *VI-programs* [11], ASP with monotone constraint atoms [42], constraint ASP [30], programs with aggregates [28], programs with function symbols [49], as well as other formalisms which aim at the integration of external sources (e.g., DL-programs). The abstract nature of HEX-*programs* makes them a representative of a class of formalisms that is useful to discuss the new techniques on a concrete yet general level; the results carry over to the less general formalisms (which we shall exemplify) and are thus relevant beyond HEX-programs alone.

HEX-programs and instantiations such as DL-programs have already been considered for a range of applications; besides those mentioned above they include e.g. complaint management in e-government [53], material culture analysis [43], user interface adaptation [50], or ontology integration in the biomedical domain [36] (see also [45]). However, a wider takeup in practice requires efficient evaluation methods. Due to the abstract nature of HEX-programs, providing such methods is non-trivial; scalable algorithms have been introduced only recently (cf. [17,21]). In a sense, the situation is akin to ordinary ASP: although the formal semantics was around since the early 1990's, only with the advent of efficient solvers (such as SMODELS [48] and DLV [40]) ASP could be widely established; broader deployment to real-world applications still took further time.[4]

The predominant evaluation approach of current ASP solvers is grounding & search, which roughly speaking means that a ground (variable-free) version of the program is generated by substituting constants for variables, and thereafter an answer set of the resulting ground (propositional) program is searched; both steps use quite sophisticated algorithms. In this paper, we focus on the grounding step. This step is non-trivial and significantly more involved than for ordinary ASP because external atoms may introduce new constants that are not present in the program; this is commonly referred to as *value invention*.

In particular, a naive support of value invention may easily lead to infinite program groundings and answer sets, as the following example demonstrates. Consider the program

$$\Pi = \left\{ \begin{array}{ll} r_1 : start(a). & r_3 : s(Y) \leftarrow r(X), \&appendA[X](Y). \\ r_2 : r(X) \leftarrow start(X) & r_4 : r(X) \leftarrow s(X). \end{array} \right\}$$

where the external atom $\&appendA[X](Y)$ returns in Y the string in X with character 'A' appended. Due to the cycle over r_3 and r_4, the start string a will be expanded infinitely many times, i.e., the program has an infinite grounding and answer set (assuming that the external source processes all finite strings over an alphabet).

Moreover, even in cases where a finite subset of the grounding suffices to compute the answer sets of the original program, the set of relevant constants is often not known a priori. For instance, let Π' be program Π where rule r_3 is replaced by $r_3' : s(Y) \leftarrow r(X), \&reach[X](Y)$, where the external atom $\&reach[X](Y)$ computes for a (fixed and finite) graph that is stored externally the set of all nodes Y which are directly reachable from the node X. The overall program then

[4] To further support this process, current ongoing work also includes means on the software engineering side for making a prototype system easily accessible, e.g. by providing pre-compiled and ready-to-use binary packages, an online demo, and tutorials. Moreover, additional language features make our implementation compliant with the ASP-Core-2 standard [13].