

Answer Set Programming in 2010: A Personal Perspective

Enrico Pontelli

Department of Computer Science
New Mexico State University
epontell@cs.nmsu.edu

Abstract. In this talk, I will offer a brief overview of the foundations of Answer Set Programming (ASP). Using some concrete application examples as a reference, I will try to identify issues that current ASP systems are still facing, and define avenues of research that I believe have the potential to greatly enhance future applicability of ASP technology.

1 Answer Set Programming

In recent years, logic programming has been experiencing a new youth, thanks in part to the fast growth of *Answer Set Programming (ASP)*, a logic programming framework capable of default and non-monotonic reasoning.

Answer Set Programming (ASP) arises from the research on using LP as a language for declarative programming of systems exhibiting non-monotonic behavior. ASP programs are expressed as collections of clauses of the form:

$$L_0 \leftarrow L_1 \wedge \cdots \wedge L_m \wedge \text{not } L_{m+1} \wedge \cdots \wedge \text{not } L_n$$

where L_i are atoms. The operator *not* denotes negation as failure and we refer to the literals *not* L_i as *naf-literals*.

The semantics of a program is expressed in terms of *answer sets* of the program, where answer sets are selected minimal supported models of the underlying logic theory. Answer set semantics can be summarized as follows. An *interpretation* is a set of atoms—identifying true atoms. An interpretation M is an answer set of a program Π if M is the least Herbrand model of the program Π^M , obtained by removing from Π all clauses containing a naf-literal *not* L such that $L \in M$, and removing all naf-literals from the remaining clauses. Note that Π^M is a standard logic program without negation, whose semantics $\text{sem}(\Pi^M)$ is characterized by the traditional least Herbrand model.

ASP programs may admit zero, one, or several answer sets. The traditional approach used in modeling a problem P in ASP consists of developing a collection of ASP clauses $\Pi(P)$, such that there exists a one-to-one correspondence between the answer sets of $\Pi(P)$ and the solutions of P . In this sense, each solution to a problem is modeled by a set of atoms (i.e., the elements of an answer set), and the ASP clauses in $\Pi(P)$ can be naturally viewed as *constraints* on the admissible answer sets.

ASP offers a number of advantages over traditional LP: (i) ASP offers a purely declarative language, free of dependencies on operational issues (e.g., order of clauses); (ii) it avoids issues like floundering and non-terminating computations; (iii) ASP is as expressive as many other non-monotonic logics, yet it provides a simpler syntax and well-developed and efficient implementations; (iv) ASP is more expressive than propositional and first-order logic, allowing us to elegantly encode causality and transitive closure; (iv) there is a large body (larger than any other knowledge representation language) of support structure built for ASP, including knowledge building blocks, laying the foundations for systematic development of programs. ASP has been successfully employed in building large reasoning systems, e.g., in the areas of diagnosis, web services, and bioinformatics.

1.1 Some Practical Issues and Interesting Directions

Experiences in the development of practical applications have highlighted several shortfalls of ASP, which are being actively addressed by current research. I will try to summarize next what I feel are important practical issues to be addressed and describe preliminary steps that have been taken to solve them.

1.2 Expressiveness: Constraints

The simplicity of the clausal language has much appeal, but it leads to large and cumbersome encodings in several practical situations—e.g., when dealing with the encoding that require manipulation of numbers. Several implementations of ASP addressed this problem through the introduction of language extensions that offer various forms of *aggregates* (e.g., aggregate atoms in DLV, choice atoms in SMOLETS). Although elegant, these extensions have been fairly unrelated and biased by the choice of incompatible semantics.

An important step is to provide a language and semantical foundation to these extensions, ensuring a broader coverage of the spectrum of extensions and a uniform and well-understood semantics. We will overview the use of *abstract constraints* as a solution to address this problem.

1.3 Raw Performance: Parallelism

Applications in domains like planning and combinatorial optimizations have highlighted that even the most modern ASP systems are not up-to-par in terms of search speed and efficiency. Even though great improvements have been recently achieved (as witnessed by the outstanding results obtained by CLASP in the recent SAT competitions), there is still significant scope for improvement.

The purely declarative nature of ASP place this paradigm well ahead of other declarative languages (e.g., CLP and traditional Prolog) for transparent exploitation of parallelism—since the language is completely free of sequential dependencies. We will overview some recent results in this area, highlighting the lessons learned and the current direction of research.

1.4 Scalability: Non-ground Computations

ASP systems allows the programmer to encode clauses that contain variables—yet, virtually all existing ASP systems are capable of computing answer sets only for ground programs. Grounding is achieved through the use of a grounding preprocessor (e.g., LPARSE or GRINGO). Modern grounders are fast and capable of optimizing the produced ground program—yet, it is not uncommon to encounter situations where the resulting ground program is infeasibly large, to the point of causing the crashing of the grounder or preventing the ASP solver from loading the large ground program.

This has prompted researchers to investigate solutions for non-ground computation of ASP. Research in this field is very preliminary, but it is moving along some promising directions, some of which will be discussed in this presentation.

1.5 Integration: Multi-context Systems

The success of paradigms like CLP showed the importance of integrating different languages and semantics (e.g., Prolog and CSP) within a single framework. ASP has the potential of providing powerful inferential features that could be beneficial within the context of larger applications. Recent work has started exploring the role of ASP in multi-context systems—i.e., systems where different monotonic and non-monotonic logics are combined in the construction of single solutions to a problem.

We will explore some general ideas in this context and illustrate some preliminary results.