Extensions of Answer Set Programming

Alex Brik[1]

1 Department of Mathematics, UC San Diego, U.S.A

Abstract

This paper describes a doctoral research in three areas: Hybrid ASP – an extension of Answer Set Programming for reasoning about dynamical systems, an extension of Set Constraint atoms for reasoning about preferences, computing stable models of logic programs using Metropolis type algorithms. The paper discusses a possible application of all three areas to the problem of maximizing total expected reward.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving

Keywords and phrases answer set programming, hybrid systems, modeling and simulation, preferences

Digital Object Identifier 10.4230/LIPIcs.ICLP.2011.261

1 Introduction and Problem Description

The research investigates three areas related to ASP.

Area 1: H-ASP. The main motivation for this area is the question of how to reason about dynamical systems that exhibit both discrete and continuous behavior. The unique feature of Hybrid ASP (H-ASP) is that H-ASP rules can be thought of as general input-output devices. In particular, H-ASP programs allow the user to include ASP type rules that act as controls for when to apply a given algorithm to advance the system to the next position. This feature allows H-ASP to be used with Partial Differential Equation (PDE) solvers and Ordinary Differential Equation (ODE) solvers.

Area 2: preferences as SC atoms. The notion of a set constraint atom (SC atom) was introduced by Marek and Remmel [18]. This notions is extended to the notion of an extended set constraint atom (ESC atom) to model preferences.

Area 3: computing stable models of logic programs using Metropolis type algorithms. The Metropolis algorithm introduced by Metropolis et al. [16] in 1953, is a widely applicable procedure for drawing samples from a specified distribution on a large finite set. It was later generalized to the Metropolis-Hastings algorithm [8]. Since its introduction the Metropolis algorithm has found many applications in statistical physics, biology, statistics, and other areas of science [5]. The subject of the research is to produce algorithms that use Metropolis type algorithms for the following two tasks:

1. Given a finite propositional logic program P which has a stable model, find a stable model M of P.

2. Given a finite propositional logic program P which has no stable model, find a maximal program $P' \subseteq P$ which has a stable model and find a stable model M' of P'.

Finding maximal subprograms that have stable models is important for certain extensions of ASP where arbitrary set constraints are used to model both hard and soft preferences. In such situations, one may not be able to satisfy all soft preferences so that stable models may not exist that satisfy all preferences. However, if certain soft preferences are dropped, then the subprograms that do have stable models may be found.

Technical Communications of the 27th International Conference on Logic Programming (ICLP'11).
Editors: John P. Gallagher, Michael Gelfond; pp. 261–267

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The three areas can be combined when one considers certain problems. For instance: what is a next action that an agent acting in a dynamical system has to perform in order to maximize the total expected reward. The idea is to describe a dynamical system in H-ASP, define optimal strategy as a set of preferred stable models and then perform computations using Metropolis type algorithm.

2 Background and Overview of Existing Literature

The following is a review of the definitions of normal propositional logic programs and stable model semantics. A *normal propositional logic program* P consists of clauses of the form $C = a \leftarrow a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n$ where $a, a_1, \ldots, a_m, b_1, \ldots, b_n$ are atoms. Here $a_1, \ldots, a_n$ are called the *premises* of clause C, $b_1, \ldots, b_m$ are called the *constraints* of clause C, and a is called the *conclusion* of clause C. For any clause C as above, let $prem(C) = \{a_1, \ldots, a_n\}$, $cons(C) = \{b_1, \ldots, b_m\}$, and $c(C) = a$. Either $prem(C)$, $cons(C)$, or both may be empty. C is said to be a Horn clause if $cons(C)$ is empty. Let $mon(P)$ denote the set of all Horn clauses of P and $nmon(P) = P \setminus mon(P)$. The elements of $nmon(P)$ will be called *nonmonotonic* clauses. Let $H(P)$ denote the Herbrand base of P. A subset $M \subseteq H(P)$ is called a **model** of a clause C if whenever $prem(C) \subseteq M$ and $cons(C) \cap M = \emptyset$, then $c(C) \in M$. M is a model of a program P if it is a model of every clause $C \in P$. The Gelfond-Lifschitz reduct of P with respect to M denoted P^M is obtained by removing every clause C such that $cons\,(C) \cap M \neq \emptyset$ and then removing the constraints from all the remaining clauses. M is called a **stable model** of P if M is the least model of P^M.

H-ASP

Modern computational models and simulations such as the model of dog's heart described in [11], or the model of internal tides within Monterey Bay and the surrounding area described in [10] rely on PDE solvers and ODE solvers to determine the values of parameters. Such simulations proceed by invoking appropriate algorithms to advance a system to the next state, which is often distanced by a short time interval into the future from the current state. In this way, a simulation of continuously changing parameters is achieved. The parameter passing mechanisms and the logic for making decisions regarding what algorithms to invoke and when are part of the ad-hoc control algorithm. Thus the laws of a system are implicit in the ad-hoc control software.

Action languages [7] which are also used to model dynamical systems allow the users to describe the laws of a system explicitly. Initially action languages did not allow simulation of the continuously changing parameters. Recently, Chintabathina introduced an action language H [4] where he proposed an elegant approach to modeling continuously changing parameters. However, the implementation of H discussed in [4] cannot use PDE solvers nor ODE solvers. This means that parameters governed by physical processes such as the distribution of heat or air flow, that cannot be described explicitly as functions of time and realistic simulations of which require sophisticated numerical methods, cannot be modeled using the current implementations of H.

Hybrid ASP [3] is an extension of ASP that allows users to combine the strength of the ad-hoc approaches, i.e. the use of numerical methods to faithfully simulate physical processes, and the expressive power of ASP, which provides the ability to elegantly model laws of a system. Hybrid ASP provides mechanisms to express the laws of the modeled system via hybrid ASP rules which can control execution of algorithms relevant for simulation.

Let S be a parameter space and let At be a set of atoms. The universe is $At \times S$. Given $M \subseteq At \times S$ and $B_i = a_1^{(i)}, \ldots, a_{n_i}^{(i)}, \neg b_1^{(i)}, ..., \neg b_{m_i}^{(i)}$, where $a_1^{(i)}$,....$a_{n_i}^{(i)}$, $b_1^{(i)}$,...,$b_{m_i}^{(i)} \in At$

and $\mathbf{p} \in S$, M satisfies B_i at $\mathbf{p}$, if $(a_j^{(i)}, \mathbf{p}) \in M$ for $j = 1, \ldots, n_i$, and $(b_j^{(i)}, \mathbf{p}) \notin M$ for $j = 1, \ldots, m_i$. **Advancing Rules** are of the form

$$\frac{B_1; B_2; \ldots; B_r : A, O}{a}$$

where A is an algorithm, each B_i is as above, $a \in At$, and $O \subseteq S^r$ is such that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$, then $t(\mathbf{p}_1) < \ldots < t(\mathbf{p}_r)$, $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) \subseteq S$, and for all $\mathbf{q} \in A(\mathbf{p}_1, \ldots, \mathbf{p}_r)$, $t(\mathbf{q}) > t(\mathbf{p}_r)$. The idea is that if for each i, B_i is satisfied at $\mathbf{p}_i$, then the algorithm A can be applied to $(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ to produce a set $O' \subseteq S$ such that if $\mathbf{q} \in O'$, then $t(\mathbf{q}) > t(\mathbf{p}_r)$ and $(a, \mathbf{q})$ holds. **Stationary rules** are of the form

$$\frac{B_1; B_2; \ldots; B_r : H, O}{a}$$

where each B_i is as above, $a \in At$, $O \subseteq S^r$ such that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$, then $t(\mathbf{p}_1) < \cdots < t(\mathbf{p}_r)$, and H is a Boolean algorithm such that for all $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$, $H(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ is defined. The idea is that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$ and for each i, B_i is satisfied at $\mathbf{p}_i$, and $H(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ is true, then $(a, \mathbf{p}_r)$ holds.

Modeling Preferences as SC Atoms

SC atoms were first introduced by Marek and Remmel in [18]. A SC atom is a pair $\langle X, \mathcal{F} \rangle$ where X is a finite set and $\mathcal{F} \subseteq 2^X$. A *set constraint clause* (SC clause) is a string of the form $\langle X, \mathcal{F} \rangle \leftarrow \langle Y_1, \mathcal{G}_1 \rangle, \ldots, \langle Y_n, \mathcal{G}_n \rangle$, where $\langle X, \mathcal{F} \rangle$, $\langle Y_1, \mathcal{G}_1 \rangle, \ldots, \langle Y_n, \mathcal{G}_n \rangle$ are SC atoms. A *set constraint program* (SC program) is a finite set of SC clauses. Let M be a set of atoms and $K = \langle X, \mathcal{F} \rangle$ be a SC atom. Define $M \models K$ if $X \cap M \in \mathcal{F}$. Marek and Remmel in [18] defined the stable model semantics for SC program using the notion of NSS transform.

The basic idea of using SC atoms to define preferences is to consider triples of the from $\langle X, \mathcal{F}, wt \rangle$ or $\langle X, \mathcal{F}, \preccurlyeq \rangle$ where X is a finite set of atoms, $\mathcal{F} \subseteq 2^X$, $wt : \mathcal{F} \to [0, \infty) \subseteq \mathbb{R}$, $\preccurlyeq$ is a partial order in $\mathcal{F}$.

The triples of the form $\langle X, \mathcal{F}, wt \rangle$ are called *weight preference set constraint atoms* and triples of the form $\langle X, \mathcal{F}, \preccurlyeq \rangle$ are called *partially ordered preference set constraint atoms.* A set of atoms M is satisfies $\langle X, \mathcal{F}, wt \rangle$ or $\langle X, \mathcal{F}, \preccurlyeq \rangle$ if and only if M satisfies $\langle X, \mathcal{F} \rangle$. Now suppose that we have a SC program P and an additional finite set of clauses T of the form $\langle X_i, \mathcal{F}_i, wt_i \rangle \leftarrow$, $i \in \{1, \ldots, n\}$. Suppose that M is a stable model of $P \cup T$. Then we can define the weight of the model M as $W(M) = \sum_{i=1}^{n} wt_i(X_i \cap M)$. The weight functions can be used to describe user's preferences for what the user wants $M \cap X_i$ to be by saying that for $F_1, F_2 \in \mathcal{F}_i$, F_1 is preferred over F_2 if $wt_i(F_1) < wt_i(F_2)$. A stable model M_1 of $P \cup T$ is preferred over the stable model M_2 of $P \cup T$ if $W(M_1) < W(M_2)$. Thus the introduction of weight preference set constraint atoms can lead to a natural weighting of stable models which can be used to model preferences. Similarly, suppose that there is an SC program P which in addition has a finite set of clauses T of the form $\langle X_i, \mathcal{F}_i, \preccurlyeq_i \rangle \leftarrow$ for $i \in \{1, \ldots, n\}$. Now suppose that two stable models M_1 and M_2 of $P \cup T$ are given. Then $M_1 \preccurlyeq M_2$ if and only if $M_1 \cap X_i \preccurlyeq_i M_2 \cap X_i$ for $i = 1, \ldots, n$. Thus the introduction of partial order preference set constraint atoms can lead to a natural partial order on stable models which can be used to model preferences.

Computing Stable Models of Logic Programs Using Metropolis Type Algorithms

The use of Metropolis type algorithms to compute stable models of logic programs is based on the Forward Chaining (FC) algorithm introduced by Marek et al.[17]. The FC algorithm provides two ingredients necessary for any procedure to be used with the Metropolis type algorithms: a way to produce a "next" candidate given a current candidate, and a measure of

merit that assigns a numeric score to a candidate (based on how closely it corresponds to a stable model).

Given a Markov chain $K(x,y)$ and a probability distribution $\pi(x)$, let $G(x,y) = \pi(y)K(y,x)/\pi(x)K(x,y)$. The Metropolis-Hastings algorithm defines a new Markov chain $M(x,y)$, where the probability $M(x,y)$ is equal to the probability of drawing $x_{t+1} = y$ given $x_t = x$ using the following procedure:

1. given current state $x_t = x$, draw y based on the Markov chain $K(x,\cdot)$;
2. draw U from the uniform distribution on $[0,1]$;
3. set $x_{t+1} = y$ if $U \leq G(x_t, y)$ and set $x_{t+1} = x_t$ otherwise.

The key result about the Metropolis-Hasting algorithm is the following.

▶ **Theorem 1.** *Let X be a finite set and $K(x,y)$ be a Markov chain on X such that $\forall x, y \in X$ $K(x,y) > 0$ iff $K(y,x) > 0$. Let $\pi(x)$ be a probability distribution on X. Let $M(x,y)$ be the Metropolis-Hastings chain as defined above. Then $\pi(x)M(x,y) = \pi(y)M(y,x)$ for all x, y. In particular, for all x, $y \in X$ $\lim_{n\to\infty} M^n(x,y) = \pi(y)$.*

The relevance of this result to the task of finding stable models of normal propositional logic programs is in the following: after sampling from M for sufficiently many steps the probability of being at y is $\pi(y)$ regardless of the starting state. If $\pi(y)$ is defined to be relatively large whenever y corresponds to a stable model then for a normal propositional logic program P which has a stable model, samples generated from M will eventually include those corresponding to the stable models and moreover the sampling from M will be biased towards those samples that correspond to the stable models of P.

There are various approaches to computing stable models of logic programs. Systems such as *smodels* [20] and *clasp* [6] use complete algorithms that will either find stable models if they exist or will report that stable models do not exist. These systems are not based on the Metropolis type algorithms. The use of the Metropolis type algorithm for the purpose of finding stable models of logic programs was investigated in [13], [14], and [15]. The Metropolis algorithm is also used in SAT solvers [19].

3 Goal of the Research

Each of the 3 areas studied has its own goal. Thus the goal of the research in the area of Hybrid ASP is to produce theoretical machinery necessary to build an H-ASP software system that is able to reason about dynamical systems that exhibit both discrete and continuous behavior, and to produce a prototype of such a software system.

In action languages like H, the goal is to compile an H program into a variant ASP program that can be processed with variant ASP solvers. A long term goal of this research is to develop extensions of ASP solvers that can process Hybrid ASP programs. This would allow the development of extensions of action languages like H that could be compiled to Hybrid ASP programs which, in turn would be processed by Hybrid ASP solvers.

The goal of the research in the area of modeling preferences is to produce theoretical machinery necessary to build a software system for finding preferred stable models of SC logic programs and to produce a prototype of such a software system.

The goal of the research in the area of computing stable models of logic programs using Metropolis type algorithms is to produce a theoretical machinery necessary to build a software system for finding stable models of logic programs using Metropolis type algorithms and to produce a prototype of such a software system.

Finally, it would be interesting to combine the research to produce a theoretical foundation and software system for solving the following problem: what is a next action that an agent acting in a dynamical system has to perform in order to maximize the total expected reward.

4 Current Status of the Research

Hybrid ASP. A paper on H-ASP [3] was accepted as a technical communication to ICLP 2011. Proof of concept software prototypes are developed and successfully tested to research the feasibility of combining H-ASP with numerical algorithms for solving PDEs and with decision making algorithms (see [22] for a review of decision making algorithms).
Preferences as SC atoms. The framework of theoretical definitions is created and various examples of the use of SC atoms to model preferences are being investigated.
Computing stable models of logic programs using Metropolis type algorithms. The subject is discussed in [2]. The paper discusses Metropolized Forward Chaining (MFC) algorithm and some computer experiments performed using the algorithm.

5 Preliminary Results Accomplished

Hybrid ASP. see section 4.
Preferences as SC atoms. Preliminary results include being able to successfully model preferences as defined in [21], as well as to model preferences in various toy examples.
Computing stable models of logic programs using Metropolis type algorithms. Preliminary results include software prototypes, both single processor and parallel versions, as well as a size 300 $(2, 6)$ Van der Waerden certificate found by the software as discussed in [2]. The certificate was found using a 288 processor parallel run in under 2 weeks. To illustrate the difficulty of finding size 300 $(2, 6)$ Van der Waerden certificate - a single processor version of smodels has failed to find size 210 certificate (while running for over 3 weeks), and a single processor version of clasp 1.3.3 has failed to produce size 240 certificate (while running for over 2 weeks). The experiments demonstrate that the method is feasible for the problem of finding stable models of logic programs and merits additional research. A different set of experiments was used to validate the use of MFC for the problem of finding maximal subprograms that have stable models of the programs that don't have stable models. Two Metropolis type algorithm were tested: Metropolis algorithm and Stochastic Approximation Monte Carlo (SAMC) algorithm [1], [12]. Preliminary experiments indicate that for difficult problems SAMC performs significantly better than the Metropolis algorithm.

6 Open Issues and Expected Achievements

Hybrid ASP. It is expected that the theoretical foundation necessary for computations with H-ASP will be produced as well as a software proof of concept prototype.
Preferences as SC atoms. It is expected that a theoretical foundation necessary for modeling preferences using SC atoms will be produced. It is not clear whether a software proof of concept prototype will be produced as part of the dissertation.
Computing stable models of logic programs using Metropolis type algorithms. The initial goal of this area is already mostly achieved. However research produced many open issues. Here are some of them: 1. what are the relative merits of using various Metropolis type algorithms? 2. What other approaches to using Metropolis type algorithms for finding

stable models of logic programs are there? 3. How does MFC compare in performance to the existing algorithms?

Regarding combining the 3 areas: it is not clear whether a fusion of 3 areas will be accomplished, and a proof of concept prototype will be produced as part of the dissertation.

Acknowledgments. The author is grateful to his thesis advisor Jeffrey Remmel for collaboration and guidance. The author is grateful to Adriano Garsia for helpful discussion and for providing a summer research assistantship from NSF grant DMS-0800273 in order to conduct some of the computer experiments. This research was supported in part by NSF MRI Award 0821816 and by other computing resources provided by the Center for Computational Mathematics (http://ccom.ucsd.edu/).

References

1 Y. F. Atchade, J. S. Liu. The Wang-Landau Algorithm in General State Spaces: Applications and Convergence Analysis. *Technical Report, Department of Statistics, University of Michigan*, (2007).

2 A. Brik and J.B. Remmel. Computing Stable Models of Logic Programs Asing Metropolis Type Algorithm. Preprint, ASPOCP ICLP 2011, (2011).

3 A. Brik, J.B. Remmel. Hybrid ASP. Preprint, technical communications ICLP 2011, (2011).

4 Sandeep Chintabathina. Towards Answer Set Programming Based Architectures for Intelligent Agents. *PhD thesis, Texas Tech University*, (2010).

5 P. Diaconis. The Markov Chain Monte Carlo Revolution. *Bulletin of the American Mathematical Society*, Nov. (2008).

6 M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-Driven Answer Set Solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, (2007), 386–392.

7 M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16), (1998).

8 W. Hastings, Monte Carlo sampling methods using Markov chains and their applications. *Biometrica*, 57 (1970), 97-109.

9 P.R. Herwig, M. J. H. Heule, P. M. van Lambalgen, and H. van Maaren. A New Method to Construct Lower Bounds for van der Waerden Numbers. *Electronic Journal of Combinatorics*, **14** (2007), #R6.

10 S. M. Jachec, O. B. Fringer, M. G. Gerritsen, R. L. Street. Numerical Simulation of Internal Tides and the Resulting Energetics within Monterey Bay and the Surrounding Area. *Geophysical Research Letters,* (2006) Vol. 33, L12605, doi: 10.1029/2006GL026314.

11 R. Kerckhoffs, M. Neal, Q. Gu, J. Bassingthwaighte, J. Omens, A. McCulloch. Coupling of a 3D Finite Element Model of Cardiac Ventricular Mechanics to Lumped Systems Models of the Systemic and Pulmonic Circulation. *Annals of Biomedical Engineering*, 35 (1), (2007), 1-18.

12 F. Liang, C. Liu, and R. J. Carroll. Stochastic Approximation in Monte Carlo Computation. *Journal of the American Statistical Association*, 102 (2007), 305-320.

13 L. Liu and M. Truszczyński. Local-search techniques in propositional logic extended with cardinality atoms. *Proceedings of the 9-th International Conference on Principles and Practice of Constraint Programming*, LNCS 2833 (2003), 495-509.

14 L. Liu and M. Truszczyński. Local search techniques for Boolean combinations of psuedo-Boolean constraints. *Proceedings of the Twentieth National Conference on Artificial Intelligence*, AAAI Press (2006), 98-103.

15 L. Liu and M. Truszczyński. Statisfiability testing of Boolean combinations of pseudo-boolean constraints using local search techniques. *Constraints*, **12**(3), (2007), 345-369.

16 N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth. A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21 (1953), 1087-1092.

17 W. Marek, A. Nerode, and J.B. Remmel. Logic Programs. Well Orderings, and Forward Chaining. *Annals of Pure and Applied Logic*, 96 (1999), 231-276.

18 W. Marek and J. B. Remmel. Set Constraint in Logic Programming. *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 7th International Conference*, LNCS 2923, (2004), 154–167.

19 B. Selman, H. Levesque, D. Mitchel. A New Method for Solving Hard Satisfiability Problems. *Proceedings of the 10th National Conference on Artificial Intelligence*, (1992), 440-446.

20 P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138 (2002), 181–234.

21 T. C. Son and E. Pontelli. Planning with Preferences using Logic Programming. *TPLP*, 6 (5), (2006), 559-608.

22 R. S. Sutton, A. G. Barto. Reinforcement Learning. 1998. *The MIT Press.*