B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 159

asprin: Answer Set Programming with Preferences

Javier Romero[1]

Abstract: Answer Set Programming (ASP) is a well established approach to *declarative problem solving*, combining a rich yet simple modeling language with high-performance solving capacities. In this talk we present *asprin*, a general, flexible and extensible framework for preferences in ASP. *asprin* is general and captures many of the existing approaches to preferences. It is flexible, because it allows for the combination of different types of preferences. It is also extensible, allowing for an easy implementation of new approaches to preferences. Since it is straightforward to capture propositional theories and constraint satisfaction problems in ASP, the framework is also relevant to optimization in Satisfiability Testing and Constraint Processing.

Keywords: Preference, Optimization, Answer Set Programming

1 Introduction

Answer Set Programming (ASP; [Ba03]) is a well established approach to *declarative problem solving*. Rather than solving a problem by telling a computer *how to solve the problem*, the idea is to simply describe *what the problem is* and leave its solution to the computer. The success of ASP is due to the combination of a rich yet simple modeling language with high-performance solving capacities. Modeling has its roots in the fields of Knowledge Representation and Logic Programming, while solving is based in methods from Deductive Databases and Satisfiability Testing (SAT; [Bi09]). ASP programs resemble Prolog programs, but they are interpreted according to the stable models semantics [GL88], and the underlying solving techniques are closely related to those of modern SAT solvers.

For solving real-world applications it is often necessary to represent and reason about preferences. This was realized quite early in ASP, leading to many approaches to preferences [BNT03; Br04; SP06]. Departing from there, we have developed our approach [Br15a; Br15b] and the resulting *asprin*[2] system providing a general and flexible framework for quantitative and qualitative preferences in ASP. Our framework is *general* and captures many of the existing approaches to preferences. It is *flexible*, providing means for the combination of different types of preferences. And it is also *extensible*, allowing for an easy implementation of new approaches to preferences. Since it is straightforward to capture propositional theories and constraint satisfaction problems in ASP, the approach is also relevant to optimization in Satisfiability Testing and Constraint Processing.

[1] University of Potsdam, javier@cs.uni-ptosdam.de
[2] *asprin* stands for "*AS*P for *pr*eference handl*in*g".

2 The Travelling Salesperson Problem in Answer Set Programming

The basic idea of ASP is to represent a given problem by a logic program whose stable models correspond to solutions, and then to use an ASP solver for finding stable models of the program. As an illustration, let us consider the Travelling Salesperson Problem (TSP). In our example, the salesperson starts in the city `a`, and should visit the cities `b`, `c`, and `d`. We are given the information about the existing roads between the cities, and their corresponding distances. In ASP, this could be represented by the following set of facts:

```
start(a). city(a). city(b). city(c). city(d).
road(a,b,10). road(b,c,20). road(c,d,25). road(d,a,40).
road(b,d,30). road(d,c,25). road(c,a,35).
```

where, for instance, `road(a,b,10)` means that there is a road from `a` to `b` of `10` kilometres. In the most basic formulation of the problem, a solution is a route visiting once every city and returning to the starting city. The following rules capture the problem:

```
{ travel(X,Y) : road(X,Y,D) }.

visited(Y) :- travel(X,Y), start(X).
visited(Y) :- travel(X,Y), visited(X).

:- city(X), not visited(X).
:- city(X), 2 { travel(X,Y) }.
:- city(X), 2 { travel(Y,X) }.
```

Line 1 chooses a set of `travel(X,Y)` atoms, where `X` and `Y` are two cities connected through a `road`. The next rules describe when a city has been `visited`: if we `travel` to it from the `start` city (Line 3), or if we `travel` from an already `visited` city (Line 4). The last lines represent constraints on the solutions. Line 6 says that it cannot be the case that a `city` is not `visited`, line 7 forbids travelling from one `city` to two or more cities and, similarly, line 8 forbids reaching a city from two or more cities. Putting together the previous facts and rules in a logic program, it turns out that the solutions of the original problem correspond to the stable models of the program, which can be computed by an ASP solver. For example, stable model M_1 contains $\{$`travel(a,b)`, `travel(b,c)`, `travel(c,d)`, `travel(d,a)`$\}$ and corresponds to the solution where first `a` is visited, then `b`, `c`, `d` and `a` again. Another stable model M_2 contains $\{$`travel(a,b)`, `travel(b,d)`, `travel(d,c)`, `travel(c,a)`$\}$.

3 *asprin*: Answer Set Programming with Preferences

asprin extends ASP with preference relations among the stable models of a logic program. Formally, a *preference relation* is a strict partial order $\succ$ over the stable models of a logic program P. Given two stable models X and Y of P, $X \succ Y$ means that X is preferred to Y with respect to $\succ$. Then, a stable model X of P is *optimal* with respect to $\succ$ if there is no other stable model Y such that $Y \succ X$.

In *asprin*, preference relations are declared by *preference statements*, composed of an identifier, a type, and a set of preference elements. The identifier names the preference relation, whereas its type and elements define the relation. Coming back to the classical formulation of the TSP, the problem is to find a route of minimum distance. The preference for shorter routes can be represented as follows:

```
#preference(distance,less(weight)){
  D :: travel(X,Y), road(X,Y,D)
}.
```

This statement[3] declares a preference relation named `distance` of type `less(weight)` with a single preference element `D :: travel(X,Y), road(X,Y,D)`. The preference type aims at reducing the sum of the weights, and the preference element says that if we travel from `X` to `Y` and the distance is `D`, we obtain a weight of `D`. Hence, the resulting relation `distance` prefers stable models that induce the minimum sum of distances. In our example, M_1, whose sum of distances is `95`, is optimal with respect to `distance`, and is preferred to M_2, whose sum of distances is `100`. Indeed, M_1 is a solution to the classical TSP.

Extending the problem definition, we could consider a conditional preference of type `aso` (standing for Answer Set Optimization, [BNT03]):

```
#preference(balance,aso){
  travel(Y,Z), road(Y,Z,D'), D'<25 || travel(X,Y),
                                      road(X,Y,D), D>25
}.
```

expressing that, whenever we travel through a long road (`D>25`), we prefer afterwards to travel through a short one (`D'<25`). And going further, both preferences could be combined:

```
#preference(all,pareto){
  **distance; **balance
}.
```

defining a new preference relation `all` which is the Pareto ordering of the two preferences `distance` and `balance`.

One important feature of *asprin* is that the semantics of preference types are not predefined. The idea is that we may define the semantics of a new preference type, and implement those semantics in *asprin* simply by writing a logic program, called a *preference program*. This program takes two (reified) stable models and decides whether one is preferred to the other following the semantics of the preference type. Then we may write a preference statement of the new type, and use it importing the corresponding preference program.

For computing optimal models, *asprin* does repeated calls to an underlying ASP solver: first, an arbitrary stable model is generated; then this stable model is "fed" to the preference

[3] In ASP, meta statements are preceded by '#'.

program to produce a preferred one, etc. Once the program becomes unsatisfiable, the last stable model obtained is an optimal one.

asprin is implemented in Python, and available for download at `https://github.com/potassco/asprin`. *asprin* provides a *library* containing a number of common, preference types along with the necessary preference programs. Users happy with what is available in the library can thus use the available types without having to bother with preference programs at all. However, if the predefined preference types are insufficient, users may define their own ones (and so become preference engineers). In this case, they also have to provide the preference programs *asprin* needs to cope with the new preference relations.

References

[Ba03] Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.

[Bi09] Biere, A.; Heule, M.; van Maaren, H.; Walsh, T., eds.: Handbook of Satisfiability. IOS Press, 2009.

[BNT03] Brewka, G.; Niemelä, I.; Truszczyński, M.: Answer Set Optimization. In (Gottlob, G.; Walsh, T., eds.): Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03). Morgan Kaufmann Publishers, pp. 867–872, 2003.

[Br04] Brewka, G.: Complex Preferences for Answer Set Optimization. In (Dubois, D.; Welty, C.; Williams, M., eds.): Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04). AAAI Press, pp. 213–223, 2004.

[Br15a] Brewka, G.; Delgrande, J.; Romero, J.; Schaub, T.: asprin: Customizing Answer Set Preferences without a Headache. In (Bonet, B.; Koenig, S., eds.): Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI'15). AAAI Press, pp. 1467–1474, 2015.

[Br15b] Brewka, G.; Delgrande, J.; Romero, J.; Schaub, T.: Implementing Preferences with asprin. In (Calimeri, F.; Ianni, G.; Truszczyński, M., eds.): Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15). Vol. 9345. Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 158–172, 2015.

[GL88] Gelfond, M.; Lifschitz, V.: The Stable Model Semantics for Logic Programming. In (Kowalski, R.; Bowen, K., eds.): Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88). MIT Press, pp. 1070–1080, 1988.

[SP06] Son, T.; Pontelli, E.: Planning with Preferences using Logic Programming. Theory and Practice of Logic Programming 6/5, pp. 559–608, 2006.