Abstract Argumentation and Answer-Set Programming

Sarah Alice Gaggl, Stefan Woltran
Vienna University of Technology, Austria

Abstract. In this article, we overview encodings for problems associated to abstract argumentation frameworks (AFs) in the language of Answer-Set Programming (ASP). Our encodings are formulated as fixed queries, such that the input is the only part depending on the actual AF to process. We illustrate the functioning of this approach, which is underlying the argumentation system ASPARTIX, briefly report on our experimental experiences, and give links to the relevant articles in the literature.

1 Motivation

In Artificial Intelligence (AI), the area of argumentation (the survey by [1] gives an excellent overview) has become one of the central issues during the last decade. Argumentation provides a formal treatment for reasoning problems arising in a number of application fields, including Multi-Agent Systems and Law Research. In a nutshell, so-called abstract argumentation frameworks (AFs) formalize statements together with a relation denoting rebuttals between them, such that the semantics gives an abstract handle to solve the inherent conflicts between statements by selecting acceptable subsets of them. The reasoning underlying such argumentation frameworks turned out to be a very general principle capturing many other important formalisms from the areas of AI and Knowledge Representation.

Argumentation problems are in general intractable, for instance deciding if an argument is contained in some preferred extension is known to be NP-complete. Therefore, developing dedicated algorithms for the different reasoning problems is non-trivial. A promising way to implement such systems is to use a reduction method, where the given problem is translated into another language, for which sophisticated systems already exist. It turned out that Answer-Set Programming (ASP) is especially well suited for this purpose.

Earlier work already proposed reductions from argumentation problems to certain target formalisms. Most notably are encodings in terms of (quantified) propositional logic [2, 6] and logic programs [10–12, 15] (see [14] for a survey). The main difference of this earlier work compared to our approach is the necessity to compile (at least, for some of the semantics) each problem instance into a different instance of the target formalism (e.g., into a different logic program). In our approach, all semantics are encoded

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-028.
within a fixed query (independent from the concrete AF to process). Thus, we are more in the tradition of a classical implementation, because we construct an interpreter in ASP which takes an AF given as input.

2 ASP-Encodings for Abstract Argumentation Frameworks

Abstract argumentation frameworks have been first introduced by Dung [3] in 1995. It is a very simple but also very powerful formalism to reason over conflicting knowledge. The syntax only consists of a set of statements called arguments and a binary relation between them, the attacks denoting the conflicts between the arguments. As we are on the abstract level, we do not concentrate on the internal structure of the arguments but only on their relation to each other.

An argumentation framework (AF) is a pair \( F = (A, R) \), where \( A \) is a finite set of arguments and \( R \subseteq A \times A \). The pair \((a, b) \in R\) means that \( a \) attacks \( b \). A set \( S \subseteq A \) of arguments attacks \( b \) (in \( F \)), if there is an \( a \in S \), such that \((a, b) \in R\). An argument \( a \in A \) is defended by \( S \subseteq A \) (in \( F \)) iff, for each \( b \in A \), it holds that, if \((b, a) \in R\), then \( S \) attacks \( b \) (in \( F \)).

Such an AF can be represented as a directed graph as in the following example.

Example 1. Consider the AF \( F = (A, R) \), consisting of the set of arguments \( A = \{a, b, c, d, e, f, g\} \) and the set of attack relations \( R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, f), (f, f), (f, g), (g, e)\} \) as illustrated in Figure 1.

The inherent conflicts between the arguments are solved by selecting subsets of arguments, where a semantics \( \sigma \) assigns a collection of sets of arguments to an AF \( F \). The basic requirement for all semantics is that none of the selected arguments attack each other; these sets are then called conflict-free. Then admissible extensions of an AF are those conflict-free sets which defend their arguments against all attacks.

In the following we present the ASP-encodings for admissible semantics as used in the system ASPARTIX (see [5] for a detailed description of most of the argumentation semantics and the corresponding encodings). First the input AF from \( F = (A, R) \) is defined as,

\[
\hat{F} = \{\text{arg}(a) \mid a \in A\} \cup \{\text{att}(a, b) \mid (a, b) \in R\}.
\] (1)
Table 1. Complexity for decision problems in argumentation frameworks.

<table>
<thead>
<tr>
<th></th>
<th>adm</th>
<th>pref</th>
<th>semi</th>
<th>stage</th>
<th>grd+</th>
<th>ground</th>
<th>cf2</th>
<th>stage2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cred</td>
<td>NP-c</td>
<td>NP-c</td>
<td>ΣP2-c</td>
<td>ΣP2-c</td>
<td>NP-c</td>
<td>in P</td>
<td>NP-c</td>
<td>ΣP2-c</td>
</tr>
<tr>
<td>Skept</td>
<td>trivial</td>
<td>ΠP2-c</td>
<td>ΠP2-c</td>
<td>ΠP2-c</td>
<td>coNP-c</td>
<td>in P</td>
<td>coNP-c</td>
<td>ΠP2-c</td>
</tr>
</tbody>
</table>

Then, the program πadm computes admissible extensions by first guessing all subsets \( S \subseteq A \) of arguments with \( \text{in}(\cdot) \) (resp. \( \text{out}(\cdot) \)) denoting the arguments in (resp. not in) the set \( S \). Then the constraints rule out those guesses which are not conflict free, or which do not defend their arguments.

\[
\pi_{\text{adm}} = \{ \text{in}(X) : \neg \text{out}(X), \text{arg}(X); \\
\text{out}(X) : \neg \text{in}(X), \text{arg}(X); \\
\text{in}(X), \text{in}(Y), \text{att}(X, Y); \\
\text{defeated}(X) : \neg \text{in}(Y), \text{att}(Y, X); \\
\text{defeated}(X) : \neg \text{in}(X), \text{att}(Y, X), \neg \text{defeated}(Y) \}.
\]

Typical reasoning tasks for an argumentation semantics \( \sigma \) are credulous and skeptical reasoning which are supported by most ASP solvers:

- Cred\(_{\sigma}\): Given \( AF = (A, R) \) and \( a \in A \). Is \( a \) contained in some \( S \in \sigma(F) \)?
- Skept\(_{\sigma}\): Given \( AF = (A, R) \) and \( a \in A \). Is \( a \) contained in each \( S \in \sigma(F) \)?

Depending on the computational complexity of the different semantics, ASPARTIX uses different techniques in the encodings.

- Stratified programs for grounded semantics [5];
- Normal programs for admissible, stable, complete (all in [5]) and \( cf2 \) [8] semantics;
- Disjunctive programs for preferred, semi-stable (both in [5]), stage [4] and stage2 semantics (on system-page);
- Manifold programs for ideal semantics [7];
- metasp optimization techniques (see [9]) for preferred, semi-stable, stage and resolution-based grounded semantics (grd∗) [4].

3 Conclusion

An experimental evaluation of the encodings with different solvers like dlv, lpars, smodels, cmodels, clasp, claspD and gnt showed that AFs with up to 140 arguments can be used as input for most of the semantics [13]. For most of the encodings gringo/clasp or gringo/claspD outperformed the other solvers. Except for semi-stable semantics dlv performed better.

Furthermore, an evaluation of handcrafted saturation encodings versus the metasp optimization technique showed that the latter one not only makes the encodings easier but also performs surprisingly well [4].

All encodings incorporated in ASPARTIX are available at
http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/

and a web-application of ASPARTIX is provided under:

http://rull.dbai.tuwien.ac.at:8080/ASPARTIX
Bibliography


