# ASPARTIX: Implementing Argumentation Frameworks Using Answer-Set Programming*

Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran

Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Vienna, Austria

**Abstract.** The system ASPARTIX is a tool for computing acceptable extensions for a broad range of formalizations of Dung's argumentation framework and generalizations thereof. ASPARTIX relies on a fixed disjunctive datalog program which takes an instance of an argumentation framework as input, and uses the answer-set solver DLV for computing the type of extension specified by the user.

## 1 Motivation

The area of argumentation (see [1] for an excellent summary) has become one of the central issues in Artificial Intelligence (AI) within the last decade, providing a formal treatment for reasoning problems arising in a number of interesting applications fields, including Multi-Agent Systems and Law Research. In a nutshell, argumentation frameworks 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 admissible 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 (KR).

The increasing interest in argumentation led to numerous proposals for formalizations of argumentation. These approaches differ in many aspects. First, there are several ways how "admissibility" of a subset of statements can be defined; second, the notion of rebuttal has different meanings (or even additional relationships between statements are taken into account); finally, statements are augmented with priorities, such that the semantics yields those admissible sets which contain statements of higher priority.

Argumentation problems are in general intractable, thus developing dedicated algorithms for the different reasoning problems is non-trivial. Instead, a more promising approach is to use a reduction method, where the given problem is translated into another language, for which sophisticated systems already exist.

The system we present in this paper follows this approach and provides solutions for reasoning problems in different types of argumentation frameworks (AFs) by means of computing the answer sets of a datalog program. To be more specific, the system is capable to compute the most important types of extensions (i.e., admissible, preferred, stable, complete, and grounded) in Dung's original AF [2], the preference-based AF [3], the value-based AF [4], and the bipolar AF [5]. Hence our system can be used to

* This work was partially supported by the Austrian Science Fund (FWF) under project P20704.

compare different argumentation semantics in a profound and novel way, and thus can be used by researchers to compare the different semantics on concrete examples within a uniform setting. Our approach is to use a *fixed* logic program which is capable of computing the different forms of extension from a given framework which is given as input. Hence, the burden of efficient computation is delegated to systems which evaluates this logic program. Due to this simple architecture, our system is easily extendible and suitable for rapid prototyping.

To the best of our knowledge, so far no system is available which supports such a broad range of different semantics, although nowadays a number of implementations exists[1], including Dungine [6] a Java reasoner capable of reasoning with grounded and preferred extensions; an epistemic and practical reasoner which also supports preferred credulous semantics for practical arguments (see [7]); PARMENIDES, a system for e-democracy based on value-based AFs [8]; and CASAPI, a Prolog implementation that combines abstract and assumption-based argumentation [9].

The work which is closest related to ours is by Nieves *et al.* [10] who also suggest to use answer-set programming for computing extensions of argumentation frameworks. The most important difference is that in their work the program has to be re-computed for each new instance, while our system relies on a *single fixed* interpreter program which just requires the actual instance as an input database. Although there is no advantage of the interpreter approach from a theoretical point of view (as long as the reductions are polynomial-time computable), there are several practical ones. The interpreter is easier to understand, easier to debug, and easier to extend. We believe that our approach thus is more reliable and easier extendible to further formalisms. An evaluation of the practical efficiency of the approach is subject of ongoing work.

Our system makes use of the prominent answer-set solver DLV [11]. All necessary programs to run ASPARTIX and some illustrating examples are available at `http://www.kr.tuwien.ac.at/research/systems/argumentation/`

## 2 Background and System Specifics

The declarative programming paradigm of *Answer Set Programming* (ASP) [11,12] under the stable-models semantics [13] is nowadays recognized as well suited for modeling and solving problems which involve common sense reasoning, and has been fruitfully applied to a range of applications including data integration, configuration, or diagnosis using advanced ASP solvers [14] such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT. The basic idea of ASP is to encode solutions to a problem into the intended models of a logic program, in a way such that the solutions are described in terms of rules and constraints instead of specifying a concrete algorithm which singles out the solutions. The problem encoding is then given to an ASP solver, which computes some or multiple answer set(s) of the program together with the input. The solutions of the problem can then be easily read off from the answer sets.

We will use ASP to compute several kinds of extensions in different argumentation frameworks. An *argumentation framework* $AF$ is a pair $(A, R)$ where $A$ is a set of arguments and $R \subseteq A \times A$. The pair $(a, b) \in R$ means that $a$ attacks $b$. A set $S \subseteq A$

[1] See also http://www.csc.liv.ac.uk/~azwyner/software.html for an overview.

of arguments attacks $b$, if $b$ is attacked by some $a \in S$. An argument $a \in A$ is *defended* by $S \subseteq A$ iff for each $b \in A$, it holds that, if $(b, a) \in R$, then $b$ is attacked by $S$. A set $S \subseteq A$ is said to be *conflict-free (in AF)*, if there are no $a, b \in S$, such that $(a, b) \in R$. A conflict-free set $S \subseteq A$ is *admissible* (for $AF$), iff each $a \in S$ is defended by $S$. A *preferred extension* of $AF$ is a maximal (w.r.t. set inclusion) admissible set of $AF$.

As an example of a generalization, we give the definitions for value-based argumentation frameworks (VAFs). VAFs are given by tuples $(A, R, V, val, valpref)$, where $A$ and $R$ are as for a standard argumentation framework, $V$ is a non-empty set of values, $val$ assigns to each $a \in A$ an element of $V$, and $valpref \subseteq V \times V$ is a preference relation (transitive, irreflexive and asymmetric). An argument $a \in A$ attacks or *defeats* an argument $b \in A$ iff both $(a, b) \in R$ and $(val(b), val(a)) \notin valpref$. Using this new notion of an attack provides the definitions of conflict-free sets, and admissible and preferred extensions for VAFs in the same way as for the basic framework.

Computing extensions of argumentation frameworks is quite straight forward within the ASP paradigm. First we guess candidates for extensions (i.e., subsets of arguments) and then rule out via constraints those candidates which do not match the particular requirements of the extension (for instance, constraints can easily eliminate those candidates which are not conflict-free). Typical language extensions in ASP are helpful for different problems. For instance, a suitable use of disjunction in rule heads allow us to formalize how to compute preferred extensions (which are known to be computationally more involved).

The architecture of the system ASPARTIX is as follows: We have a single ASP program which provides all necessary rules to compute extensions of the different argumentation formalisms. The task of the user is just to set up the input database which contains (i) the instance of the argumentation frameworks (ii) the type of extension ASPARTIX should compute. Then, invoking DLV with this input facts together with the ASPARTIX-program provides answer sets which are in a one-to-one correspondence with the specified extensions of the given framework. Technical details of the encoding are provided in a companion paper [15].

## 3 Applying ASPARTIX

The example we will consider to demonstrate our system is adapted from [16] and describes the case "Popov v. Hayashi" as decided by the honorable Kevin McCarthy in 2002. The case concerned the possession of the baseball which Barry Bonds hit for his record breaking 73rd home run in the 2001 season. When the ball was struck into the crowd, Popov caught it in the upper part of the webbing of his baseball glove. But Popov was not given the chance to complete his catch since, as it entered his glove, he was tackled and thrown to the ground by others trying to secure the ball, which became dislodged from his glove. Hayashi (himself innocent of the attack on Popov), then picked up the ball and put it in his pocket, so securing possession.

After the examination of all testimonies and videotapes neither Popov nor Hayashi were able to establish possession to the baseball. But McCarthy had to make a decision which is fair to both parties. Therefore he considered the following arguments on which he also assigned different values:

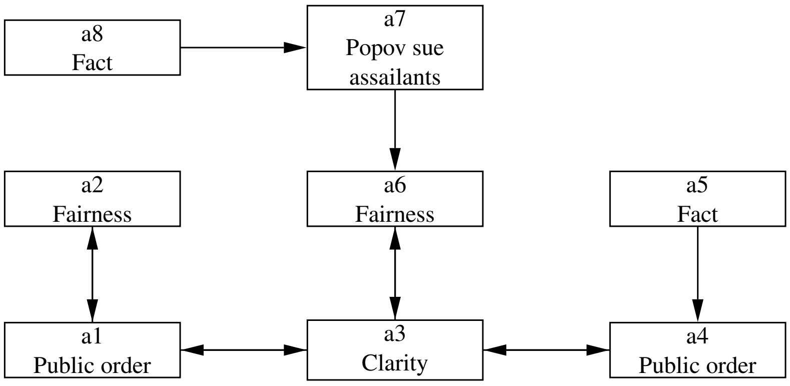

**Fig. 1.** Value-based Argument Graph for Popov v. Hayashi

1. Where interruption of completing the catch so establishing possession was illegal; decide for Popov; to prevent assault being rewarded; promoting the value of public order.
2. Where it has not been shown that Hayashi did not have possession and did nothing wrong; do not decide for Popov; which would punish Hayashi; demoting the value of fairness.
3. Where Hayashi had unequivocal control of the baseball; decide for Hayashi; to provide a bright line; promoting clarity of law.
4. Where interruption of completing the catch so establishing possession was illegal; do not insist on unequivocal control; which would reward assault; demoting the value of public order.
5. Since Hayashi was not an assailant, finding for Hayashi would not reward assault.
6. Where it has not been shown that Popov did not have possession and did nothing wrong; do not decide for Hayashi; which would punish Popov; demoting the value of fairness.
7. Where interruption of completing the catch so establishing possession was illegal; Popov should sue the assailants of the assault; which would not punish Popov; promoting the value of fairness.
8. Since assailants cannot be identified, suing those responsible for the assault is not a viable action.

Figure 1 shows arguments 1–8 from above together with the attack relation inbetween them (see [16] for details). The input file `input.dl` for this example (formulated as a simple framework without values) would contain the following facts:

```
prefex. arg(a1). arg(a2). ... arg(a8).
attacks(a1,a2). attacks(a1,a3). attacks(a2,a1). attacks(a3,a1).
attacks(a3,a4). attacks(a3,a6). attacks(a4,a3). attacks(a5,a4).
attacks(a6,a3). attacks(a7,a6). attacks(a8,a7).
```

The fact `prefex` specifies that the *preferred* extensions should be computed. Executing ASPARTIX thus consists of one call to DLV using `input.dl` and our program `aspartix.dl`, using filter options of DLV as follows:

```
./DLV input.dl aspartix.dl -filter=in,input_error
```

which produces the following output representing the preferred extensions

```
{in(a2),in(a3),in(a5),in(a8)}, {in(a2),in(a5),in(a6),in(a8)},
{in(a1),in(a5),in(a6),in(a8)}.
```

Hence, one still does not get a clear answer unless the framework is accordingly extended to a VAF. We add the following facts to our input file.

```
val(a1,public_order). val(a2,fairness). val(a3,clarity).
val(a4,public_order). val(a5,fact). val(a6,fairness).
val(a7,popov_sue_assailants). val(a8,fact).
vaf. valpref(fairness,public_order). valpref(fairness,clarity).
valpref(fairness,fact). valpref(fairness,popov_sue_assailants).
```

Here, predicate `vaf` tells ASPARTIX that the input is a VAF, predicate `val` specifies the values of the arguments, and predicate `valpref` defines a value ranking.

If we now invoke ASPARTIX as above with the now extended input file we receive as output a single answer set representing the preferred extension of the specified VAF, {`a2,a5,a6,a8`}. Since the result is that both `a2` and `a6` are contained in the preferred extension, McCarthy can decide neither for Popov nor for Hayashi. Therefore, he decided that the ball should be sold and the proceeds divided between the two.

## References

1. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artif. Intell. 171, 619–641 (2007)
2. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77, 321–358 (1995)
3. Amgoud, L., Cayrol, C.: A reasoning model based on the production of acceptable arguments. Ann. Math. Artif. Intell. 34, 197–215 (2002)
4. Bench-Capon, T.J.M.: Persuasion in practical argument using value-based argumentation frameworks. J. Log. Comput. 13, 429–448 (2003)
5. Amgoud, L., Cayrol, C., Lagasquie, M.C., Livet, P.: On bipolarity in argumentation frameworks. International Journal of Intelligent Systems 23, 1–32 (2008)
6. South, M., Vreeswijk, G., Fox, J.: Dungine: A java dung reasoner. In: Proceedings of COMMA 2008, pp. 360–368 (2008)
7. Visser, W.: (2008), `http://www.wietskevisser.nl/research/epr/`
8. Cartwright, D., Atkinson, K.: Political engagement through tools for argumentation. In: Proceedings of COMMA 2008, pp. 116–127 (2008)
9. Gaertner, D., Toni, F.: (2008), `http://www.doc.ic.ac.uk/~dg00/casapi.html`
10. Nieves, J.C., Osorio, M., Cortés, U.: Preferred extensions as stable models. Theory and Practice of Logic Programming 8, 527–543 (2008)
11. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM ToCL 7, 499–562 (2006)
12. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. 25, 241–273 (1999)
13. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput 9, 365–386 (1991)
14. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
15. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. Technical Report DBAI-TR-2008-62, Technische Universität Wien (2008)
16. Wyner, A., Bench-Capon, T.J.M., Atkinson, K.: Arguments, values and baseballs: Representation of Popov v. Hayashi. In: Proceedings of JURIX 2007, pp. 151–160 (2007)