Argumentation and Answer Set Programming

Francesca Toni and Marek Sergot

Department of Computing,
Imperial College London, UK
{ft,mjs}@imperial.ac.uk

Abstract. Argumentation and answer set programming are two of the main knowledge representation paradigms that have emerged from logic programming for non-monotonic reasoning. This paper surveys recent work on using answer set programming as a mechanism for computing extensions in argumentation. The paper also indicates some possible directions for future work.

1 Introduction

Argumentation was developed, starting in the early '90s [9,16,8], as a computational framework to reconcile and understand common features and differences amongst most existing approaches to non-monotonic reasoning. These include various alternative treatments of negation as failure in logic programming [32,26,52,19], Theorist [43], default logic [45], autoepistemic logic [38], non-monotonic modal logic [37] and circumscription [36]. Argumentation relies upon

- the representation of knowledge in terms of an argumentation framework; defining *arguments* and a binary *attack* relation between the arguments;
- *dialectical semantics* for determining *acceptable* sets of arguments;
- a computational machinery for determining the acceptability of a given (set of) argument(s) or for computing all acceptable sets of arguments (also referred to as *extensions*), according to some dialectical semantics.

Answer set programming (ASP) [31] constitutes one of the main current trends in logic programming and non-monotonic reasoning. ASP relies upon

- the representation of knowledge in terms of (possibly disjunctive) logic programs with negation as failure (possibly including explicit negation, various forms of constraints, aggregates etc);
- the interpretation of these logic programs under the stable model/answer set semantics [32,33] and its extensions (to deal with explicit negation, constraints, aggregates etc);
- efficient computational mechanisms (ASP solvers) to compute answer sets for grounded logic programs, and efficient 'grounders' to transform non-ground logic programs into ground (variable-free) ones.

Standard computational mechanisms for argumentation are defined using trees (e.g. [18]) or disputes (e.g. [20]) and only construct relevant parts of extensions. ASP can instead be used to support the full computation of extensions.

This paper provides a survey of recent work using ASP for computing extensions in abstract argumentation frameworks [16] and some other forms of argumentation. It also indicates possible directions for future work and cross-fertilisation between ASP and argumentation.

The paper is organised as follows. Section 2 gives some background on argumentation (focusing on abstract argumentation [16]) and ASP. Section 3 surveys existing approaches using ASP to compute extensions in argumentation (again focusing on abstract argumentation). Section 4 indicates some possible directions for future work. Section 5 concludes.

2 Background

2.1 Argumentation

An *abstract argumentation (AA) framework* [16] is a pair $\langle Arg, att \rangle$ where Arg is a finite set, whose elements are referred to as *arguments*, and $att \subseteq Arg \times Arg$ is a binary relation over Arg. Given $\alpha, \beta \in Arg$, α *attacks* β iff $(\alpha, \beta) \in att$. Given sets $X, Y \subseteq Arg$ of arguments, X *attacks* Y iff there exists $x \in X$ and $y \in Y$ such that $(x, y) \in att$. A set of arguments is referred to as an *extension*. An extension $X \subseteq Arg$ is

- *conflict-free* iff it does not attack itself;
- *stable* iff it is conflict-free and it attacks every argument it does not contain;
- *acceptable wrt* a set $Y \subseteq Arg$ of arguments iff for each β that attacks an argument in X, there exists $\alpha \in Y$ such that α attacks β;
- *admissible* iff X is conflict-free and X is acceptable wrt itself;
- *preferred* iff X is (subset) maximally admissible;
- *complete* iff X is admissible and X contains all arguments α such that $\{\alpha\}$ is acceptable wrt X;
- *grounded* iff X is (subset) minimally complete.

In addition, an extension $X \subseteq Arg$ is

- *ideal* [18] iff X is admissible and it is contained in every preferred set of arguments;
- *semi-stable* [12] iff X is complete and $X \cup X^+$ is (subset) maximal, where $X^+ = \{\beta \mid (\alpha, \beta) \in att \text{ for some } \alpha \in X\}$.

These notions of extensions constitute different alternative dialectical semantics, giving different approaches for determining what makes arguments dialectically viable. Arguments can be deemed to hold *credulously* wrt a given dialectical semantics if they belong to an extension sanctioned by that semantics. Arguments can be deemed to hold *sceptically* wrt a given dialectical semantics if they belong to all extensions sanctioned by that semantics. In some cases credulous and sceptical reasoning coincide, e.g. for grounded and ideal extensions, since these are unique.

For $AF = \langle Arg, att \rangle$, the *characteristic function* $\mathcal{F}_{AF}$ is such that $\mathcal{F}_{AF}(X)$ is the set of all acceptable arguments wrt X. Then, a conflict-free $X \subseteq Arg$ is

- an admissible extension iff $X \subseteq \mathcal{F}_{AF}(X)$,
- a complete extension iff it is a fixpoint of $\mathcal{F}_{AF}$, and
- a grounded extension iff X is the least fixpoint of $\mathcal{F}_{AF}$.

Several other argumentation frameworks have been given in the literature, concretely specifying arguments and attacks, some instantiating abstract argumentation, e.g. assumption-based argumentation [9,8,17] and logic programming-based argumentation frameworks such as [44], some equipped with dialectical semantics other than the ones proposed for abstract argumentation, e.g. [7,30]. Moreover, extensions of abstract argumentation have been proposed, e,g, value-based argumentation [5].

2.2 Answer Set Programming (ASP)

A logic program is a set of clauses of the form

$$p_1 \vee \ldots \vee p_k \leftarrow q_1 \wedge \ldots \wedge q_m \wedge not\, q_{m+1} \wedge \ldots \wedge not\, q_{m+n}$$

for $k \geq 0$, $m \geq 0$, $n \geq 0$, $k + m + n > 0$, where the p_i and q_j are *literals*, that is, of the form p or $\neg p$ where p is an atom. *not* denotes negation as failure. Expressions of the form $not\, q_j$ where q_j is a literal are called 'negation as failure literals', or *nbf-literals* for short. We will refer to $\{p_1, \ldots, p_k\}$ as the *head*, $\{q_1, \ldots, q_m, not\, q_{m+1}, \ldots not\, q_{m+n}\}$ as the *body* and $\{not\, q_{m+1}, \ldots not\, q_{m+n}\}$ as the *negative body* of a clause. We will also refer to clauses with $k = 0$ as *denial clauses*, clauses with $k > 1$ as *disjunctive* clauses, and clauses with $n = 0$ as *positive clauses.*

All variables in clauses in a logic program are implicitly universally quantified, with scope the individual clauses. A logic program stands for the set of all its ground instances over a given Herbrand universe. The semantics of logic programs is given for their grounded version over this Herbrand universe.

The answer sets of a (grounded) logic program are defined as follows [33].

Let X be a set of literals (that is, expressions p or $\neg p$ where p is an atom). A literal is true in X if it belongs to X. A nbf-literal $not\, p$ is true in X if the literal p does not belong to X. A clause is true in X if its head is true in X (there exists a literal in the head that is true in X) whenever its body is true in X (i.e., when all the literals and all the nbf-literals in that body are true in X). Thus, denial clauses are true in X if and only if their body is false in X (i.e., some literal in the body is not true in X). A set of literals X is closed under a set of clauses P if every clause in P is true in X. X is 'consistent' if it contains no pair of complementary literals p and $\neg p$ for some atom p; X is logically closed if it is consistent or if it is the set of all literals otherwise.

If P is a set of positive clauses, that is, a set of clauses containing no occurrences of negation as failure *not*, then X is an *answer set* of P if it is a (subset) minimal set of literals that is logically closed and closed under the clauses in P.

If P is any set of clauses (not necessarily positive ones) the Gelfond-Lifschitz reduct P^X of P wrt to the set of literals X is obtained from P by deleting (1) all clauses in P containing a nbf-literal $not\, p$ in the body where p is true in X, and (2) the negative body from all remaining clauses. P^X is a set of positive clauses. Then X is an *answer set* of P when X is an answer set of P^X.

None of the logic programs presented in section 3 of this paper contain occurrences of classical negation ($\neg$): all of the literals in all of the clauses are atoms. For programs P of this type, the answer sets of P are also the *stable models* [32] of P.

Several very efficient ASP solvers are widely available and can be used to compute answer sets and/or perform query answering wrt answer sets. These solvers include Smodels[1], DLV[2] and clasp[3]. These solvers incorporate, or are used in combination with, *grounders*, programs whose function is to generate, prior to computing answer sets, a finite ground logic program from a non-ground logic program (over a given, not necessarily finite Herbrand Universe).

3 ASP for Argumentation

Several approaches have been proposed for computing (several kinds of) extensions of *abstract argumentation* (AA) frameworks using ASP solvers [39,53,23,27]. All rely upon the mapping of an AA framework into a logic program whose answer sets are in one-to-one correspondence with the extensions of the original AA framework. All those summarised below use the DLV solver to compute these answer sets (and thus the extensions). The approaches differ in the kinds of extensions they focus on and in the mappings and correspondences they define, as we will see below. They fall into two groups: those which result in an AA framework-dependent logic program, and those which result in a logic program with an AA framework-dependent component and an AA framework-independent (meta-)logic program.

3.1 Nieves, Cortés and Osorio [39]: Preferred Extensions

Nieves et al [39] focus on the computation of **preferred** extensions. Their mapping relies upon the method of [6] using propositional formulas to express conditions for sets of arguments to be extensions of AA frameworks. The mapping produces a disjunctive logic program defining a predicate *def*, where $def(\alpha)$ can be read as 'argument α is defeated'. Intuitively,

– each pair (α, β) in the *att* component of an AA framework (Arg, att) is mapped to a disjunctive clause $def(\alpha) \vee def(\beta) \leftarrow$;

[1] `http://www.tcs.hut.fi/Software/smodels/`

[2] `http://www.dbai.tuwien.ac.at/proj/dlv/`

[3] `http://potassco.sourceforge.net/`

– for each pair $(\beta, \alpha) \in att$, a clause $def(\alpha) \leftarrow def(\gamma_1) \wedge \ldots \wedge def(\gamma_k)$ $(k \geq 0)$ is introduced, where $\gamma_1, \ldots, \gamma_k$ are all the 'defenders' of α against β (that is, where $(\gamma_1, \beta), \ldots, (\gamma_k, \beta) \in att$, and there are no other attacks against β in att).

$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & c \end{tikzcd}$$

Fig. 1. Graph representation for the AA framework $(\{a, b, c\}, \{(a, b), (b, c)\})$

For the AA framework of figure 1, the mapping returns

$$\begin{aligned} &def(a) \vee def(b) \leftarrow \\ &def(b) \vee def(c) \leftarrow \\ &def(c) \leftarrow def(a) \\ &def(b) \leftarrow \end{aligned}$$

The answer sets of the disjunctive logic program P^{pref}_{NCO} thus obtained are in one-to-one correspondence with the preferred extensions of the original AA framework $\langle Arg, att \rangle$, in that the complement

$$\mathcal{C}(AS) \ =_{\text{def}} \ \{\alpha \in Arg \mid def(\alpha) \notin AS\}$$

of each answer set AS of P^{pref}_{NCO} is a preferred extension of $\langle Arg, att \rangle$.

In the example of the AA framework of figure 1 and the resulting P^{pref}_{NCO} given earlier, the only answer set is $\{def(b)\}$, corresponding to the only preferred extension $\{a, c\} = \mathcal{C}(\{def(b)\})$ of the original AA framework.

$$\begin{tikzcd} a \arrow[loop left] \end{tikzcd}$$

Fig. 2. Graph representation for the AA framework $(\{a\}, \{(a, a)\})$

In the example of figure 2, P^{pref}_{NCO} is

$$\begin{aligned} &def(a) \vee def(a) \leftarrow \\ &def(a) \leftarrow def(a) \end{aligned}$$

with answer set $\{def(a)\}$ corresponding to the (only) preferred extension $\{\}$ of the original AA framework. In the case of the AA framework of figure 3, P^{pref}_{NCO} is

$$\begin{aligned} &def(a) \vee def(b) \leftarrow \\ &def(a) \leftarrow def(a) \\ &def(b) \leftarrow def(b) \end{aligned}$$

with answer sets $\{def(a)\}$ and $\{def(b)\}$ corresponding to the preferred extensions $\{b\}$ and $\{a\}$ (respectively) of the original AA framework.

$$\begin{tikzcd} a \arrow[r, leftrightarrow] & b \end{tikzcd}$$

Fig. 3. Graph representation for the AA framework $(\{a, b\}, \{(a, b), (b, a)\})$

3.2 Wakaki and Nitta [53]: Complete, Stable, Preferred, Grounded, and Semi-stable Extensions

Wakaki and Nitta [53] focus on the computation of complete, stable, preferred, grounded, and semi-stable extensions. Their mappings rely upon Caminada's *reinstatement labellings* [11] and correspondences between various kinds of constraints on such labellings and various notions of extensions in abstract argumentation. Intuitively, a reinstatement labelling is a total function from arguments to labels $\{in, out, undec\}$ such that (i) an argument is labelled *out* iff some argument attacking it is labelled *in*, and (ii) an argument is labelled *in* iff all arguments attacking it are labelled *out*.

All mappings given by Wakaki and Nitta result in a logic program that contains, for a given AA framework $\langle Arg, att\rangle)$, a set $P_{\langle Arg,att\rangle}$ of clauses $arg(\alpha) \leftarrow$ for all arguments $\alpha \in Arg$ and clauses $att(\alpha, \beta) \leftarrow$ for all pairs $(\alpha, \beta) \in att$. For example, in the case of the AA framework of figure 1, one obtains:

$$\begin{aligned}
&arg(a) \leftarrow\\
&arg(b) \leftarrow\\
&arg(c) \leftarrow\\
&att(a, b) \leftarrow\\
&att(b, c) \leftarrow
\end{aligned}$$

In the case of **complete** extensions, the logic program P^{compl}_{WN} resulting from the mapping includes in addition to $P_{\langle Arg,att\rangle}$ the following (AA framework-independent) clauses (directly corresponding to the notion of reinstatement labelling):

$$\begin{aligned}
&in(X) \leftarrow arg(X) \wedge not\, ng(X)\\
&ng(X) \leftarrow in(Y) \wedge att(Y, X)\\
&ng(X) \leftarrow undec(Y) \wedge att(Y, X)\\
&out(X) \leftarrow in(Y) \wedge att(Y, X)\\
&undec(X) \leftarrow arg(X) \wedge not\, in(X) \wedge not\, out(X)
\end{aligned}$$

The answer sets of P^{compl}_{WN} thus obtained are in one-to-one correspondence with the original AA framework $\langle Arg, att\rangle$, in that the *in* arguments

$$\mathcal{I}(AS) \ =_{\text{def}} \ \{\alpha \in Arg \mid in(\alpha) \in AS\}$$

of each answer set AS of P^{compl}_{WN} is a complete extension of $\langle Arg, att\rangle$.

In the example AA framework of figure 1, there is just one answer set of P^{compl}_{WN}: $\{in(a), in(c), out(b)\}$, corresponding to the only complete extension $\{a, c\} = \mathcal{I}(\{in(a), in(c), out(b)\})$ of the original AA framework. In the example

AA framework of figure 2, there is just one answer set of P_{WN}^{compl}: $\{undec(a)\}$, corresponding to the only complete extension $\{\} = \mathcal{I}(\{undec(a)\})$ of the original AA framework. In the example AA framework of figure 3, there are three answer sets of P_{WN}^{compl}: $\{in(a), out(b)\}$, $\{in(b), out(a)\}$, and $\{undec(a), undec(b)\}$, corresponding to the three complete extensions $\{a\}$, $\{b\}$ and $\{\}$, respectively, of the original AA framework.

In the case of **stable** extensions, the logic program defined by Wakaki and Nitta is P_{WN}^{stable} obtained by extending P_{WN}^{compl} with the clause

$$\leftarrow undec(X)$$

which imposes the further requirement that reinstatement labellings have an empty *undec* component. Thus, in the case of the AA framework of figure 3, there are only two answer sets of P_{WN}^{stable}, since $\{undec(a), undec(b)\}$ is not an answer set in this case. In the case of the AA framework of figure 2, there is also no answer set of P_{WN}^{stable}, since $\{undec(a)\}$ is no longer an answer set. The answer sets of P_{WN}^{stable} are in one-to-one correspondence with the original AA framework $\langle Arg, att\rangle$, in that the *in* arguments $\mathcal{I}(AS)$ of each answer set AS of P_{WN}^{stable} is a stable extension of $\langle Arg, att\rangle$, similarly to complete extensions.

Caminada [11] has proven that reinstatement labellings with a minimal *in* component, a maximal *in* component, and a minimal *undec* component correspond, respectively, to grounded, preferred and semi-stable extensions. In order to impose these maximality/minimality conditions and obtain logic programs with answer sets corresponding to grounded, preferred and semi-stable extensions, Wakaki and Nitta extend P_{WN}^{compl} to include meta-logic programs to be used to check answer sets of P_{WN}^{compl} (and thus reinstatement labellings). Such answer sets are determined in a 'guess & check' fashion [25]. The meta-logic programs are different for the three notions of extensions, but include a common core MP_{WN} consisting of the following (meta-)clauses

$$m_1(in_t(X)) \leftarrow in(X) \wedge arg(X)$$
$$m_1(undec_t(X)) \leftarrow undec(X) \wedge arg(X)$$

where $in_t(\alpha)$ and $undec_t(\alpha)$ are terms corresponding to atoms $in(\alpha)$ and $undec(\alpha)$ in P_{WN}^{compl} and m_1 is a meta-predicate expressing the candidate reinstatement labelling to be checked, as well as (meta-)clauses, for all answer sets AS of P_{WN}^{compl}:

$$m_2(in_t(X), \psi(AS)) \leftarrow in(X) \in AS$$
$$m_2(undec_t(X), \psi(AS)) \leftarrow undec(X) \in AS$$

ψ is a function assigning a unique natural number to every answer set of P_{WN}^{compl} and m_2 is a meta-predicate expressing alternative reinstatement labellings to be compared with the candidate reinstatement labelling being checked.

Then, P_{WN}^{pref} is $P_{WN}^{compl} \cup MP_{WN}$ extended with

$$\leftarrow d(Z) \wedge not\, c(Z)$$
$$d(\psi(AS)) \leftarrow m_2(in_t(X), \psi(AS)) \wedge not\, m_1(in_t(X))$$
$$c(\psi(AS)) \leftarrow m_1(in_t(X)) \wedge not\, m_2(in_t(X), \psi(AS))$$

$P_{WN}^{grounded}$ is $P_{WN}^{compl} \cup MP_{WN}$ extended with

$$\begin{aligned}
&\leftarrow c(Z) \wedge not\, d(Z)\\
&d(\psi(AS)) \leftarrow m_2(in_t(X), \psi(AS)) \wedge not\, m_1(in_t(X))\\
&c(\psi(AS)) \leftarrow m_1(in_t(X)) \wedge not\, m_2(in_t(X), \psi(AS))
\end{aligned}$$

Finally, P_{WN}^{semi} is $P_{WN}^{compl} \cup MP_{WN}$ extended with

$$\begin{aligned}
&\leftarrow d(Z) \wedge not\, c(Z)\\
&d(\psi(AS)) \leftarrow m_2(undec_t(X), \psi(AS)) \wedge not\, m_1(undec_t(X))\\
&c(\psi(AS)) \leftarrow m_1(undec_t(X)) \wedge not\, m_2(undec_t(X), \psi(AS))
\end{aligned}$$

As in the case of complete and stable extensions, preferred, grounded and semi-stable extensions correspond to the *in* arguments $\mathcal{I}(AS)$ of answer sets AS of the respective logic programs.

3.3 Egly, Gaggl and Woltran [23,24]: Conflict-Free, Admissible, Preferred, Stable, Semi-stable, Complete, Grounded Extensions

Egly et al [23] deal with the computation of conflict-free, admissible, preferred, stable, complete, and grounded extensions. Like Wakaki and Nitta [53] summarised in the previous section, an AA framework $\langle Arg, att\rangle$ is first mapped to a set of clauses $P_{\langle Arg,att\rangle}$ to be included in logic programs defined for computing the various notions of extension.

For **conflict-free** extensions, the logic program P_{EGW}^{cf} consists of $P_{\langle Arg,att\rangle}$ together with[4]

$$\begin{aligned}
&\leftarrow in(X) \wedge in(Y) \wedge att(X, Y)\\
&in(X) \leftarrow not\, out(X) \wedge arg(X)\\
&out(X) \leftarrow not\, in(X) \wedge arg(X)
\end{aligned}$$

The answer sets of P_{EGW}^{cf} are in one-to-one correspondence with the conflict-free extensions of the AA framework $\langle Arg, att\rangle$ mapped onto the $P_{\langle Arg,att\rangle}$ component of P_{EGW}^{cf}, in the same sense as in [53] (namely that the *in* arguments $\mathcal{I}(AS)$ of the answer sets AS correspond to conflict-free extensions).

A similar correspondence exists for the other kinds of extensions and the answer sets of the logic programs given below.

For **stable** extensions, the logic program P_{EGW}^{stable} consists of P_{EGW}^{cf} and

$$\begin{aligned}
&\leftarrow out(X) \wedge not\, defeated(X)\\
&defeated(X) \leftarrow in(Y) \wedge att(Y, X)
\end{aligned}$$

[4] Note that predicates *in* and *out* here are different from those used in [53] and, in particular, do not refer to the reinstatement labelling of [11].

For **admissible** extensions, the logic program P^{adm}_{EGW} consists of P^{cf}_{EGW} and

$$\begin{aligned}
&\leftarrow in(X) \wedge not\_defended(X)\\
&not\_defended(X) \leftarrow att(Y,X) \wedge not\ defeated(Y)\\
&defeated(X) \leftarrow in(Y) \wedge att(Y,X)
\end{aligned}$$

For **complete** extensions, the logic program P^{compl}_{EGW} consists of P^{adm}_{EGW} and

$$\leftarrow out(X) \wedge not\ not\_defended(X)$$

For **grounded** extensions, the logic program $P^{grounded}_{EGW}$ is obtained by mirroring the characteristic function presentation of this semantics (see section 2.1). The program makes use of an arbitrary ordering $<$ over arguments assumed to be given *a priori*. The program consists of three components. The first component $P^{<}_{EGW}$ uses the given ordering $<$ over arguments to define notions of infimum *inf*, supremum *sup* and successor *succ* over arguments, as follows:

$$\begin{aligned}
&succ(X,Y) \leftarrow lt(X,Y) \wedge not\ nsucc(X,Y)\\
&nsucc(X,Z) \leftarrow lt(X,Y) \wedge lt(Y,Z)\\
&lt(X,Y) \leftarrow arg(X) \wedge arg(Y) \wedge X < Y\\
&inf(X) \leftarrow arg(X) \wedge not\ ninf(X)\\
&ninf(Y) \leftarrow lt(X,Y)\\
&sup(X) \leftarrow arg(X) \wedge not\ nsup(X)\\
&nsup(X) \leftarrow lt(X,Y)
\end{aligned}$$

The second component computes all arguments defended (by all arguments currently *in*) in the layers obtained using *inf*, *sup* and *succ*, as follows:

$$\begin{aligned}
&defended(X) \leftarrow sup(Y) \wedge defended\_up\_to(X,Y)\\
&defended\_up\_to(X,Y) \leftarrow inf(Y) \wedge arg(X) \wedge not\ att(Y,X)\\
&defended\_up\_to(X,Y) \leftarrow inf(Y) \wedge in(Z) \wedge att(Z,Y) \wedge att(Y,X)\\
&defended\_up\_to(X,Y) \leftarrow succ(Z,Y) \wedge defended\_up\_to(X,Z) \wedge not\ att(Y,X)\\
&defended\_up\_to(X,Y) \leftarrow succ(Z,Y) \wedge defended\_up\_to(X,Z) \wedge in(V) \wedge\\
&\qquad\qquad att(V,Y) \wedge att(Y,X)
\end{aligned}$$

The third component of $P^{grounded}_{EGW}$ simply imposes that all defended arguments should be *in*:

$$in(X) \leftarrow defended(X)$$

Further, for **preferred** extensions, P^{pref}_{EGW} is $P^{adm}_{EGW} \cup P^{<}_{EGW}$ extended with a further component incorporating a maximality check on the *in* arguments. This is done by guessing a larger extension with more *in* arguments than the current extension, and checking that this is not admissible, again in a 'guess & check' fashion [25]. Membership in the guessed larger extension is defined using a new predicate *inN* (and corresponding new predicate *outN*). This additional component in P^{pref}_{EGW} is:

$$
\begin{aligned}
&\leftarrow \mathit{not\,spoil}\\
&\mathit{spoil} \leftarrow \mathit{eq}\\
&\mathit{eq} \leftarrow \mathit{sup}(Y) \wedge \mathit{eq\_up\_to}(Y)\\
&\mathit{eq\_up\_to}(Y) \leftarrow \mathit{inf}(Y) \wedge \mathit{in}(Y) \wedge \mathit{inN}(Y)\\
&\mathit{eq\_up\_to}(Y) \leftarrow \mathit{inf}(Y) \wedge \mathit{out}(Y) \wedge \mathit{outN}(Y)\\
&\mathit{eq\_up\_to}(Y) \leftarrow \mathit{succ}(Z,Y) \wedge \mathit{in}(Y) \wedge \mathit{inN}(Y) \wedge \mathit{eq\_up\_to}(Z)\\
&\mathit{eq\_up\_to}(Y) \leftarrow \mathit{succ}(Z,Y) \wedge \mathit{out}(Y) \wedge \mathit{outN}(Y) \wedge \mathit{eq\_up\_to}(Z)\\
&\mathit{spoil} \leftarrow \mathit{inN}(X) \wedge \mathit{inN}(Y) \wedge \mathit{att}(X,Y)\\
&\mathit{spoil} \leftarrow \mathit{inN}(X) \wedge \mathit{outN}(Y) \wedge \mathit{att}(Y,X) \wedge \mathit{undefeated}(Y)\\
&\mathit{undefeated}(X) \leftarrow \mathit{sup}(Y) \wedge \mathit{undefeated\_up\_to}(X,Y)\\
&\mathit{undefeated\_up\_to}(X,Y) \leftarrow \mathit{inf}(Y) \wedge \mathit{outN}(X) \wedge \mathit{outN}(Y)\\
&\mathit{undefeated\_up\_to}(X,Y) \leftarrow \mathit{inf}(Y) \wedge \mathit{outN}(X) \wedge \mathit{not\,att}(Y,X)\\
&\mathit{undefeated\_up\_to}(X,Y) \leftarrow \mathit{succ}(Z,Y) \wedge \mathit{undefeated\_up\_to}(X,Z) \wedge \mathit{outN}(Y)\\
&\mathit{undefeated\_up\_to}(X,Y) \leftarrow \mathit{succ}(Z,Y) \wedge \mathit{undefeated\_up\_to}(X,Z) \wedge \mathit{not\,att}(Y,X)\\
&\mathit{inN}(X) \leftarrow \mathit{spoil} \wedge \mathit{arg}(X)\\
&\mathit{outN}(X) \leftarrow \mathit{spoil} \wedge \mathit{arg}(X)
\end{aligned}
$$

Finally, for **semi-stable** extensions, Egly et al define P^{semi}_{EGW} as a variant of P^{pref}_{EGW} (see [24] for details).

3.4 Faber and Woltran [27]: Ideal Extensions

Faber and Woltran present an encoding of the computation of ideal extensions into so-called *manifold* answer set programs [27]. These programs allow various forms of meta-reasoning to be implemented within ASP, including credulous and sceptical reasoning. The manifold answer set program used for computing ideal extensions follows the algorithm of [21], which works as follows.

- Let *adm* be the set of all the admissible extensions of a given argumentation framework $\langle Arg, att\rangle$.
- Let $X^- = Arg \setminus \bigcup_{S\in adm} S$.
- Let $X^+ = \{\alpha \in Arg \mid \forall \beta, \gamma \in Arg : (\beta,\alpha),(\alpha,\gamma) \in att \Rightarrow \beta,\gamma \in X^-\} \setminus X^-$.
- Let $\langle Arg^*, att^*\rangle$ be the argumentation framework with $Arg^* = X^+ \cup X^-$ and $att^* = att \cap \{(\alpha,\beta),(\beta,\alpha) \mid \alpha \in X^+, \beta \in X^-\}$.
- Let adm^* be the set of all admissible extensions of $\langle Arg^*, att^*\rangle$.

Then, the ideal extension of $\langle Arg, att\rangle$ is $\bigcup_{S\in adm^*} S \cap X^+$.

The admissible extensions of (Arg^*, att^*) can be computed using a fixpoint iteration (which can be done in polynomial time since this argumentation framework is *bipartite* [21]). At the first iteration, X_1 is generated by eliminating all arguments in Arg^* that are attacked by unattacked arguments. At the second iteration, X_2 is X_1 minus all arguments that are attacked by arguments unattacked by X_1, and so on, until no more arguments can be eliminated (after at most $|X^+|$ iterations).

The logic program whose answer sets correspond to ideal extensions is obtained by using the manifold for credulous reasoning of the logic program for admissible extensions given by [23], further extended to identify (Arg^*, att^*) and to simulate the fixpoint algorithm outlined above. Details of this logic program can be found in [27].

3.5 DLV for ASP for Abstract Argumentation

All approaches described above have been implemented using the DLV ASP solver.

For the method of [39] (section 3.1) DLV can be used to perform credulous and sceptical reasoning under preferred extensions as follows. Given the logic program P^{pref}_{NCO} for an AA framework (Arg, att) and the query $\alpha?$ for an argument $\alpha \in Arg$, DLV used in `-brave` mode determines whether α belongs to a preferred extension of (Arg, att), and in `-cautious` mode whether α belongs to all preferred extensions of (Arg, att).

DLV is employed in a system[5] that can perform credulous and sceptical reasoning under the various semantics considered by [53] (section 3.2 above). DLV is also the core of the ASPARTIX system[6] [23,22]. This system supports the computation of admissible, stable, complete, grounded, preferred and ideal extensions, following the work of [23,27] (see section 3.3 above), as well as semi-stable extensions [12] and cf2 extensions [3] (see section 3.6 below) following encodings given in [24].

3.6 ASP for Other Forms of Argumentation

ASP has been used as a computational tool for abstract argumentation under other semantics, notably the cf2 extensions semantics [3], as well as for forms of argumentation other than abstract argumentation. In particular:

– Thimm and Kern-Isberner [48] present mappings of the DeLP argumentation framework [30] onto ASP.
– Egly et al [23] define mappings for value-based argumentation [5], a form preference-based abstract argumentation [1] and bipolar argumentation [2].
– Wakaki and Nitta [54] use ASP for computing extensions of a form of abductive argumentation they define.
– Devred et al [15] use ASP to compute extensions of abstract argumentation frameworks extended with constraints in the form of propositional formulas. Their mapping of constrained argumentation onto ASP uses lists and, as

[5] `http://www.ailab.se.shibaura-it.ac.jp/compARG.html`
[6] `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/`

a consequence, only ASP solvers capable of dealing with lists can be used with the outcome of this mapping. These include DLV-complex[7] [10] and ASPerIX[8] [35].
- Gaggl and Woltran [29] and Egly et al [24] propose encodings for abstract argumentation under the cf2 extensions semantics [3]. These are incorporated within the ASPARTIX system (see section 3.5).
- Osorio et al [40] propose an alternative mapping for abstract argumentation under the cf2 extensions semantics and use the DLV system to compute these extensions under the given mapping.

4 Some Future Directions

The approaches presented in section 3 employ the DLV system as the underlying ASP solver of choice. The approach of [39] (section 3.1) makes use of the capability of DLV to deal with disjunctive clauses and deploy the `brave` and `cautious` modes of DLV. The approach of [23,24] (section 3.3) makes use of the $<$ ordering that is built into DLV. The approach of [53] (section 3.2) uses the `brave` and `cautious` modes of DLV. It would be interesting to see whether other ASP solvers, e.g. the clasp and claspD (for disjunctive clauses)[9] solvers, could be beneficially used to support the computation of extensions and query answering in abstract argumentation. It would also be interesting to perform a comparative performance analysis of the methods presented in section 3 to identify the most efficient method to support various kinds of applications.

With the exception of a few works (see section 3.6) ASP has been deployed to support *abstract* argumentation. However, the majority of applications of argumentation, e.g. medical decision making [28] and legal reasoning [4], require concrete argumentation frameworks, where arguments and attacks are built from knowledge bases of rules and facts and are constructed on demand, that is to say, as determined by the needs of a given query/claim to be argued for or against. In particular, there are a number of approaches to argumentation based on logic programming (e.g [44,47,42]), extending logic programming (e.g. assumption-based argumentation [9,8,17]), or based upon classical logic (e.g. [7]). It would be useful to see whether these other forms of argumentation could be fruitfully computed using ASP.

Evidently this can always be done in the sense that these concrete forms of argumentation are (usually) also instances of abstract argumentation frameworks: having generated the relevant arguments and attacks, one could compute the extensions of the resulting abstract argumentation framework using any of the methods described in previous sections. What we have in mind however is rather different. What we would really like is a mapping from the rule base R used to construct arguments and attacks to a logic program P used to compute extensions, but one which retains some well defined correspondence between the argument-generating rules in R and (some of) the clauses in P.

[7] `http://www.mat.unical.it/dlv-complex`

[8] `http://www.info.univ-angers.fr/pub/claire/asperix/`

[9] `http://potassco.sourceforge.net/`

As has often been remarked, ASP computations are oriented primarily towards the generation of models (answer sets) rather than to proofs or chains of reasoning as in other forms of logic programming. In several of the existing applications of argumentation, however, it is precisely the explanation/justification of answers to queries, in the form of arguments for and against a particular claim, that matters more than the answers themselves. This is an essential feature, for example, when argumentation is used in a collaborative setting, e.g. as in [46,13], where arguments are constructed and evaluated across agents with possibly different knowledge bases, and explanations serve to inform and share information. In applications concerned with the (internal) resolution of dilemmas, conflicts between defeasible rules of conduct, decisions about vague or uncertain outcomes, it is often the explanations/justifications that are of primary interest. In an argumentation setting, explanations/justifications take the form of a *dialectical structure* which presents the relevant arguments, identifies the attacks between them, possibly classifying them into different types, and provides some representation of how arguments are defended against attacks. The specific details vary according to the concrete form of argumentation employed and the chosen dialectical semantics.

It would be extremely valuable to investigate to what extent dialectical explanations could be meaningfully extracted from extensions computed by means of ASP. A step in this direction has been made with the ASPARTIX system (see section 3.5) which includes a graphical interface labelling nodes of argumentation graphs to provide a visualisation of abstract argumentation frameworks. In general, however, and especially when the graph is large, only relevant parts of the graph should be presented (as the presentation of full extensions tends to obscure matters).

The survey presented in this paper was motivated in part by an interest in how argumentation frameworks could be extended with *priorities* (also referred to as 'preferences' in the literature). We are interested in how priorities (relative strengths of rules and arguments) could be used in the resolution of conflicts between defeasible rules of belief, and between defeasible rules of conduct in practical reasoning. There is reason to think that these forms of reasoning, though similar, might nevertheless be different in some important respects. See, for example, the recent discussion in [41] (though that is not presented in argumentation terms). There is some existing work on incorporating forms of priority/preference in (non-abstract) argumentation. See for example [44,34,49,51,50]. It is fair to say that it is still fragmented, however, certainly when compared to the long standing investigations of priorities/preferences in nonmonotonic reasoning. See [14] for a survey and classification of approaches, including the computation of answer sets. As observed in [14], a major difficulty in adding a treatment of priority into a (rule-based) argumentation framework lies in defining a suitable ordering on the strength of arguments based on a priority ordering of the rules from which they are constructed. It may be that it is better to approach the problem by ordering the strengths of attacks instead of the strengths of arguments, or some combination of the two. These questions remain to be investigated, as well as possible relationships to existing work on priorities/preferences in ASP. There is then the further issue, essential for the kind of applications we have in mind, of finding some way of extracting dialectical explanations from the computed extensions.

5 Conclusions

Argumentation and answer set programming are two of the main knowledge representation paradigms that have emerged from logic programming for non-monotonic reasoning.

We have surveyed a number of existing approaches to using ASP for computing extensions in argumentation. The majority of these approaches focus on abstract argumentation, and rely upon mapping abstract argumentation frameworks onto logic programs whose answer sets correspond to (various kinds of) extensions for abstract argumentation. We have seen that approaches exist for computing the majority of existing notions of extensions (including conflict-free, admissible, stable, preferred, complete, semi-stable, and ideal extensions). All presented approaches have been implemented in DLV.

We have also indicated some possible directions for future research on using ASP for argumentation, including: (i) deploying ASP solvers other than DLV, e.g. claspD, (ii) considering concrete (rather than abstract) argumentation frameworks in support of applications, and (iii) developing methods for explaining answers to queries (claims) in dialectical terms, drawing from relevant parts of answer sets corresponding to extensions where the claims hold true.

Acknowledgements

We thank Luis Moniz Pereira for useful comments on an earlier version of this paper.

References

1. Amgoud, L., Cayrol, C.: A reasoning model based on the production of acceptable arguments. Annals of Mathematics and Artificial Intelligence 34(1-3), 197–215 (2002)
2. Amgoud, L., Cayrol, C., Lagasquie-Schiex, M.-C., Livet, P.: On bipolarity in argumentation frameworks. International Journal of Intelligent Systems 23(10), 1062–1093 (2008)
3. Baroni, P., Giacomin, M., Guida, G.: SCC-recursiveness: a general schema for argumentation semantics. Artificial Intelligence 168(1-2), 162–210 (2005)
4. Bench-Capon, T., Prakken, H., Sartor, G.: Argumentation in legal reasoning. In: Rahwan, I., Simari, G. (eds.) Argumentation in Artificial Intelligence, pp. 363–382. Springer, Heidelberg (2009)
5. Bench-Capon, T.J.M.: Persuasion in practical argument using value-based argumentation frameworks. Journal of Logic and Computation 13(3), 429–448 (2003)

6. Besnard, P., Doutre, S.: Characterization of semantics for argument systems. In: Dubois, D., Welty, C.A., Williams, M.-A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004), pp. 183–193. AAAI Press, Menlo Park (2004)
7. Besnard, P., Hunter, A.: Elements of Argumentation. MIT Press, Cambridge (2008)
8. Bondarenko, A., Dung, P., Kowalski, R., Toni, F.: An abstract, argumentation-theoretic approach to default reasoning. Artificial Intelligence 93(1-2), 63–101 (1997)
9. Bondarenko, A., Toni, F., Kowalski, R.: An assumption-based framework for non-monotonic reasoning. In: Nerode, A., Pereira, L. (eds.) Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning, pp. 171–189. MIT Press, Cambridge (1993)
10. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in asp: Theory and implementation. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 407–424. Springer, Heidelberg (2008)
11. Caminada, M.: On the issue of reinstatement in argumentation. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 111–123. Springer, Heidelberg (2006)
12. Caminada, M.: Semi-stable semantics. In: Dunne, P.E., Bench-Capon, T.J.M. (eds.) Proceedings of the Second International Conference on Computational Models of Argument (COMMA 2006). Frontiers in Artificial Intelligence and Applications, vol. 144, pp. 121–130. IOS Press, Amsterdam (2006)
13. de Almeida, I.C., Alferes, J.J.: An argumentation-based negotiation for distributed extended logic programs. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS (LNAI), vol. 4371, pp. 191–210. Springer, Heidelberg (2007)
14. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Computational Intelligence 20(2), 308–334 (2004)
15. Devred, C., Doutre, S., Lefèvre, C., Nicolas, P.: Dialectical proofs for constrained argumentation. In: Baroni, P., Cerutti, F., Giacomin, M., Simari, G. (eds.) Proceedings of the Third International Conference on Computational Models of Argument (COMMA 2010), vol. 216. IOS Press, Amsterdam (2010)
16. Dung, P.: On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. Artificial Intelligence 77, 321–357 (1995)
17. Dung, P., Kowalski, R., Toni, F.: Assumption-based argumentation. In: Rahwan, I., Simari, G. (eds.) Argumentation in Artificial Intelligence, pp. 199–218. Springer, Heidelberg (2009)
18. Dung, P., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artificial Intelligence, Special Issue on Argumentation in Artificial Intelligence 171(10-15), 642–674 (2007)
19. Dung, P.M.: Negations as hypotheses: An abductive foundation for logic programming. In: Proceedings of 8th International Conference on Logic Programming, ICLP 1991, pp. 3–17 (1991)
20. Dung, P.M., Thang, P.M.: A unified framework for representation and development of dialectical proof procedures in argumentation. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, pp. 746–751 (2009)
21. Dunne, P.E.: The computational complexity of ideal semantics. Artificial Intelligence 173(18), 1559–1591 (2009)

22. Egly, U., Gaggl, S.A., Wandl, P., Woltran, S.: ASPARTIX conquers the web. In: Baroni, P., Giacomin, M., Simari, G. (eds.) Proceedings of the Second International Conference on Computational Models of Argument (COMMA 2010). Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (2010)
23. Egly, U., Gaggl, S.A., Woltran, S.: ASPARTIX: Implementing argumentation frameworks using answer-set programming. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 734–738. Springer, Heidelberg (2008)
24. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. In: Argument and Computation (2010) (accepted for publication)
25. Eiter, T., Polleres, A.: Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. Theory and Practice of Logic Programming 6(1-2), 23–60 (2006)
26. Eshghi, K., Kowalski, R.A.: Abduction compared with negation by failure. In: Proceedings of 6th International Conference on Logic Programming, ICLP 1989, pp. 234–254 (1989)
27. Faber, W., Woltran, S.: Manifold answer-set programs for meta-reasoning. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 115–128. Springer, Heidelberg (2009)
28. Fox, J., Glasspool, D., Grecu, D., Modgil, S., South, M., Patkar, V.: Argumentation-based inference and decision making–a medical perspective. IEEE Intelligent Systems 22(6), 34–41 (2007)
29. Gaggl, S., Woltran, S.: CF2 semantics revisited. In: Baroni, P., Cerutti, F., Giacomin, M., Simari, G. (eds.) Proceedings of the Third International Conference on Computational Models of Argument (COMMA 2010), vol. 216, pp. 243–254. IOS Press, Amsterdam (2010)
30. Garcia, A., Simari, G.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming 4(1-2), 95–138 (2004)
31. Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation, ch. 7, pp. 285–316. Elsevier, Amsterdam (2007)
32. 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 Logic Programming, pp. 1070–1080. MIT Press, Cambridge (1988)
33. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–386 (1991)
34. Kowalski, R.A., Toni, F.: Abstract argumentation. Journal of Artificial Intelligence and Law, Special Issue on Logical Models of Argumentation 4(3-4), 275–296 (1996)
35. Lefèvre, C., Nicolas, P.: The first version of a new asp solver: Asperix. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 522–527. Springer, Heidelberg (2009)
36. McCarthy, J.: Circumscription—a form of non-monotonic reasoning. Artificial Intelligence 13, 27–39 (1980)
37. McDermott, D.V.: Nonmonotonic logic ii: Nonmonotonic modal theories. J. ACM 29(1), 33–57 (1982)
38. Moore, R.C.: Semantical considerations on nonmonotonic logic. Artif. Intell. 25(1), 75–94 (1985)
39. Nieves, J.C., Cortés, U., Osorio, M.: Preferred extensions as stable models. TPLP 8(4), 527–543 (2008)

40. Osorio, M., Nieves, J.C., Gómez-Sebastià, I.: CF2-extensions as answer-set models. In: Baroni, P., Cerutti, F., Giacomin, M., Simari, G. (eds.) Proceedings of the Third International Conference on Computational Models of Argument (COMMA 2010), vol. 216. IOS Press, Amsterdam (2010)
41. Parent, X.: Moral particularism and deontic logic. In: Governatori, G., Sartor, G. (eds.) DEON 2010. LNCS, vol. 6181, pp. 84–97. Springer, Heidelberg (2010)
42. Pereira, L.M., Pinto, A.M.: Approved models for normal logic programs. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 454–468. Springer, Heidelberg (2007)
43. Poole, D.: A logical framework for default reasoning. Artificial Intelligence 36(1), 27–47 (1988)
44. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-classical Logics 7, 25–75 (1997)
45. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13(1-2), 81–132 (1980)
46. Schroeder, M., Schweimeier, R.: Fuzzy argumentation for negotiating agents. In: Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, pp. 942–943. ACM, New York (2002)
47. Schweimeier, R., Schroeder, M.: A parameterised hierarchy of argumentation semantics for extended logic programming and its application to the well-founded semantics. Theory and Practice of Logic Programming 5(1-2), 207–242 (2005)
48. Thimm, M., Kern-Isberner, G.: On the relationship of defeasible argumentation and answer set programming. In: Besnard, P., Doutre, S., Hunter, A. (eds.) Proceedings of the Second International Conference on Computational Models of Argument (COMMA 2008). Frontiers in Artificial Intelligence and Applications, vol. 172, pp. 393–404. IOS Press, Amsterdam (2008)
49. Toni, F.: Assumption-based argumentation for closed and consistent defeasible reasoning. In: Satoh, K., Inokuchi, A., Nagao, K., Kawamura, T. (eds.) JSAI 2007. LNCS (LNAI), vol. 4914, pp. 390–402. Springer, Heidelberg (2008)
50. Toni, F.: Assumption-based argumentation for epistemic and practical reasoning. In: Casanovas, P., Sartor, G., Casellas, N., Rubino, R. (eds.) Computable Models of the Law. LNCS (LNAI), vol. 4884, pp. 185–202. Springer, Heidelberg (2008)
51. Toni, F.: Assumption-based argumentation for selection and composition of services. In: Sadri, F., Satoh, K. (eds.) CLIMA VIII 2007. LNCS (LNAI), vol. 5056, pp. 231–247. Springer, Heidelberg (2008)
52. Van Gelder, A., Ross, K., Schlifp, J.: The well-founded semantics for general logic programs. Journal of ACM 38(3), 620–650 (1991)
53. Wakaki, T., Nitta, K.: Computing argumentation semantics in answer set programming. In: Hattori, H., Kawamura, T., Idé, T., Yokoo, M., Murakami, Y. (eds.) JSAI 2008. LNCS, vol. 5447, pp. 254–269. Springer, Heidelberg (2009)
54. Wakaki, T., Nitta, K., Sawamura, H.: Computing abductive argumentation in answer set programming. In: McBurney, P., Rahwan, I., Parsons, S., Maudet, N. (eds.) ArgMAS 2009. LNCS, vol. 6057, pp. 195–215. Springer, Heidelberg (2010)