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 the two main knowledge representation paradigms 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 directions for future work.

1 Introduction

Argumentation was developed, starting in the early '90s [9, 14, 8], as a computational framework to reconcile and understand common features and differences amongst most existing approaches to non-monotonic reasoning, including various alternative treatments of negation as failure in logic programming [30, 24, 42, 17], theorist [38], default logic [40], autoepistemic logic [35], non-monotonic modal logic [34] and circumscription [33]. 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) [29] constitutes one of the main current trends in logic programming and non-monotonic reasoning. ASP relies upon

- the representation of knowledge in terms of 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 [30, 31] 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 turn non-ground logic programs into grounded ones.

Standard computational mechanisms for argumentation are defined using trees (e.g. see [16]) or disputes (e.g. see [18]) 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 of abstract argumentation frameworks [14] 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 [14]) 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* [14] 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 x such that $\{x\}$ is acceptable wrt X;
- *grounded* iff X is (subset) minimally complete.

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

- *ideal* [16] 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 $X^+ = \{\beta | (\alpha, \beta) \in att \text{ for some } \alpha \in X\}$.

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, 15] and logic programming-based argumentation frameworks such as [39], some equipped with dialectical semantics other than the ones proposed for abstract argumentation, e.g. [7, 28]. 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 $k \geq 0$, $m \geq 0$, $n \geq 0$, $k+m+n > 0$, p_i, q_j atoms, and *not* negation as failure. 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 *standard clauses*, clauses with $k > 1$ as *disjunctive* clauses, 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 [30, 31].

An *interpretation* is a set of literals (namely atoms and negation as failure of atoms) that is consistent (namely it does not contain an atom and its negation as failure). A literal is true in an interpretation if it belongs to it. A literal is false in an interpretation if its complement belongs to it (the complement of an atom is its negation as failure, the complement of *not* a is the atom a). A clause is true in an interpretation I if its head is true in I (namely there exists an atom in the head that is true in I) whenever its body is true in I (namely all the literals in the body are true in I). Thus, denial clauses are true in an interpretation I only if their body is false in I (namely some literal in the body is false in I).

A *model* of a logic program P is an interpretation M that is total (each literal is either true or false in M) and such that all clauses in P are true in M.

If P is a set of positive clauses, then an *answer set* of P is a model M such that the set of all atoms in M is (subset) minimal amongst all the models of P.

If P is a set of any clauses, let the Gelfond-Lifschitz transform [30] of P wrt an interpretation I be P^I obtained from P by deleting 1) all clauses in P whose negative body is false in I (namely with some literal in the body false in I) and 2) the negative body from all remaining clauses. P^I is a set of positive clauses. Then an *answer set* of P is a model M that is an answer set of P^M.

Several ASP solvers have been proposed, 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 combinations with grounders generating, prior to computing answer sets, ground logic programs (over a given, typically finite Herbrand Universe) from non-ground logic programs.

3 ASP for Argumentation

3.1 ASP for abstract argumentation

Several approaches have been proposed for computing (several kinds of) extensions of AA frameworks using ASP solvers [36, 43, 21, 25]. 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 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. The mapping defined by the first approach results into an AA framework-dependent logic program. The mappings defined by the other approaches result into logic programs with an AA framework-dependent component and an AA framework-independent (meta-)logic program.

Nieves et al [36] for preferred extensions. Nieves et al [36] 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. It results into 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 onto a disjunctive clause $def(\alpha) \vee def(\beta) \leftarrow$; moreover
- for each α being attacked (namely occurring in some pair $(\beta, \alpha) \in att$), a standard clause $def(\alpha) \leftarrow def(\gamma_1) \wedge \ldots \wedge def(\gamma_k)$ $(k \geq 0)$ is introduced for each argument β attacking α, where $\gamma_1, \ldots, \gamma_k$ are all "defenders" of α against β (namely $(\delta_1, \beta), \ldots, (\delta_k, \beta) \in att$, and there are no more attacks against β).

[1] `http://www.tcs.hut.fi/Software/smodels/`
[2] `http://www.dbai.tuwien.ac.at/proj/dlv/`
[3] `http://potassco.sourceforge.net/`

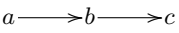

Fig. 1. Graph representation for the AA framework $(\{a, b, c\}, \{(a, b), (b, c)\})$.

For the AA framework of figure 1, the mapping returns

$$def(a) \vee def(b) \leftarrow$$
$$def(b) \vee def(c) \leftarrow$$
$$def(c) \leftarrow def(a)$$
$$def(b) \leftarrow$$

The answer sets of the disjunctive logic program P_{NCO}^{pref} 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" of each answer set AS of P_{NCO}^{pref} is a preferred extension of $\langle Arg, att\rangle$. This "complement" can be defined as follows:

- $\mathcal{C}(AS) = \{\alpha \in Arg | def(\alpha) \notin AS\}$

In the case of the AA framework of figure 1 and the resulting P_{NCO}^{pref} 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.

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

As further illustrations, in the case of the AA framework of figure 2, P_{NCO}^{pref} is

$$def(a) \vee def(a) \leftarrow$$
$$def(a) \leftarrow def(a)$$

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

$$def(a) \vee def(b) \leftarrow$$
$$def(a) \leftarrow def(a)$$
$$def(b) \leftarrow def(b)$$

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

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

Wakaki and Nitta [43] for complete, stable, preferred, grounded, and semi-stable extensions. Wakaki and Nitta [43] focus on the computation of complete, stable, preferred, grounded, and semi-stable extensions. Their mappings rely upon Caminada's *reinstatement labellings* and correspondence between various kinds of constraints on such labellings and various notions of extensions in abstract argumentation [11]. 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 including a logic program $P_{\langle Arg,att\rangle}$ with standard clauses $arg(\alpha) \leftarrow$ for all arguments $\alpha \in Arg$ and $att(\alpha, \beta) \leftarrow$ for all pairs $(\alpha, \beta) \in att$ (for a given AA framework $\langle Arg, att\rangle$). 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 addition, in the case of **complete** extensions, the logic program P_{WN}^{compl} resulting from the mapping also includes the following (AA framework-independent) standard 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_{WN}^{compl} thus obtained are in one-to-one correspondence with the original AA framework $\langle Arg, att\rangle$, in that the "in" arguments of each answer set AS of P_{WN}^{compl} is a complete extension of $\langle Arg, att\rangle$. These "in" arguments can be defined as follows

- $\mathcal{I}(AS) = \{\alpha \in Arg | in(\alpha) \in AS\}$

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

$$\leftarrow undec(X)$$

corresponding to imposing that reinstatements labelling 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. Also, in the case of the AA framework of figure 2, there is 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 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, 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) while these are determined, in a "guess&check" fashion [23]. These meta-logic programs are different for the three notions of extensions, but include a common core MP_{WN} consisting of (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 la-

belling to be checked, as well as (meta-)clauses, for all answer sets AS of P_{WN}^{compl}:

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

where ψ is a function univocally assigning a natural number to answer sets 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

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

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

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

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

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

As in the case of complete and stable extensions, preferred, grounded and semi-stable extensions correspond to the "in" arguments in answer sets of the respective logic programs.

Egly et al [21, 22] for conflict-free, admissible, preferred, stable, semi-stable, complete, grounded extensions. Egly et al [21] focus on the computation of conflict-free, admissible, preferred, stable, complete, and grounded extensions. Like Wakaki and Nitta [43], they map an AA framework $\langle Arg, att \rangle$ onto a logic program $P_{\langle Arg,att \rangle}$, included in all logic programs they define for computing the various notions of extension.

For **conflict-free** extensions, they define a logic program P_{EGW}^{cf} consisting of $P_{\langle Arg,att \rangle}$ and [4]

[4] Note that predicates *in* and *out* here are different from those used in [43] and, in particular, do not refer to the reinstatement labelling of [11].

$$\leftarrow in(X) \wedge in(Y) \wedge att(X,Y)$$
$$in(X) \leftarrow not\, out(X) \wedge arg(X)$$
$$out(X) \leftarrow not\, in(X) \wedge arg(X)$$

The answer sets of P^{cf}_{EGW} 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^{cf}_{EGW}, in the same sense as in [43] (namely the "in" arguments in the answer sets 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^{stable}_{EGW} consists of P^{cf}_{EGW} and

$$\leftarrow out(X) \wedge not\, defeated(X)$$
$$defeated(X) \leftarrow in(Y) \wedge att(Y,X)$$

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

$$\leftarrow in(X) \wedge not\_defended(X)$$
$$not\_defended(X) \leftarrow att(Y,X) \wedge not\, defeated(Y)$$
$$defeated(X) \leftarrow in(Y) \wedge att(Y,X)$$

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 as 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:

$$succ(X,Y) \leftarrow lt(X,Y) \wedge not\, nsucc(X,Y)$$
$$nsucc(X,Z) \leftarrow lt(X,Y) \wedge lt(Y,Z)$$
$$lt(X,Y) \leftarrow arg(X) \wedge arg(Y) \wedge X < Y$$
$$inf(X) \leftarrow arg(X) \wedge not\, ninf(X)$$
$$ninf(Y) \leftarrow lt(X,Y)$$
$$sup(X) \leftarrow arg(X) \wedge not\, nsup(X)$$
$$nsup(X) \leftarrow lt(X,Y)$$

The second component to compute all arguments "defended" (by all arguments currently "in") in the layers obtained using inf, sup and $succ$, as follows:

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

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

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

Further, for **preferred** extensions, P_{EGW}^{pref} is $P_{EGW}^{adm} \cup P_{EGW}^{<}$ extended with a further component incorporating a maximality check on the "in" arguments, 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 [23]. Membership in the guessed larger extension is defined using a new predicate inN (and corresponding new predicate $outN$). This additional component in P_{EGW}^{pref} is:

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

Finally, for **semi-stable** extensions, Egly et al define P^{semi}_{EGW} as a variant of P^{pref}_{EGW} (see [22] for details).

Faber and Woltran [25] for ideal extensions. Faber and Woltran propose an encoding of the computation of ideal extensions into manifold answer set programs [25]. These programs allow to implement various forms of meta-reasoning within ASP, including credulous and sceptical reasoning. The manifold answer set programs used for computing ideal extensions follows the algorithm of [19], 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 $X^+ = \{\alpha \in Arg | \forall \beta, \gamma \in Arg : (\beta, \alpha), (\alpha, \gamma) \in att \Rightarrow \beta, \gamma \in X^-\} \setminus X^-$
- let $\langle Arg^*, att^*\rangle$ be the argumentation framework with $Arg^* = X^+ \cup X^-$ and $att^* = att \cap \{(\alpha, \beta), (\beta, \alpha) | \alpha \in X^+, \beta \in X^-\}$
- 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 in polynomial time using a fixpoint iteration, since this argumentation framework is bipartite. 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 extension given by [21] extended to identify (Arg^*, att^*) and to simulate the fixpoint algorithm outlined above. Details of this logic program can be found in [25].

3.2 DLV for ASP for abstract argumentation

All approaches described in section 3.1 have been implemented using the DLV ASP solver.

DLV can be used to perform credulous and sceptical reasoning under preferred extensions using the approach of [36] as follows. Let `file_lp` be a file with the logic program P^{pref}_{NCO} obtained for an AA framework (Arg, att) and `file_arg` be a file with α? for an argument $\alpha \in Arg$. Then

- `$ dlv -brave file_lp file_arg` can be used to determine whether α belongs to a preferred extension of (Arg, att)
- `$ dlv -cautious file_lp file_arg` can be used to determine whether α belongs to all preferred extensions of (Arg, att)

DLV is used to perform credulous and sceptical reasoning under the various semantics considered by [43] within the system avavilble at

`http://www.ailab.se.shibaura-it.ac.jp/compARG.html`.

DLV is also the core of the ASPARTIX system [21, 20] available at

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

This system allows to compute admissible, stable, complete, grounded, preferred and ideal extensions, following the work of [21, 25], as well as semi-stable extensions [12] and cf2 extensions [3] (see also section 3.3, following encodings given in [22].

3.3 ASP for other forms of argumentation

ASP has been proposed 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 [41] propose mappings of the DeLP argumentation framework [28] onto ASP.
- Egly et al [21] define mappings for value-based argumentation [5], preference-based argumentation [1] and bipolar argumentation [2].
- Wakaki and Nitta [44] use ASP for computing extensions of a form of abductive argumentation they define.
- Devred et al [13] 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 a consequence, only ASP solvers capable of dealing with lists can be used with the outcome of this mapping. These include DLV-complex[5] [10] and ASPerIX[6] [32].
- Gaggl and Woltran [27] and Egly et al [22] propose encodings for abstract argumentation under the cf2 extensions semantics [3]. These are incorporated withing the ASPARTIX system (see section 3.2).
- Osorio et al [37] 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

All approaches defined in section 3.1 rely upon the DLV system as the underlying ASP solver of choice. The approach of [36] makes use of the capability of DLV to deal with disjunctive clauses and deploy the `brave` and `cautious` modes of DLV. The approach of [21, 22] makes use of the $<$ ordering that is built into DLV. The approach of [43] uses the `brave` and `cautious` modes of DLV. It would be interesting to see whether other ASP solver, e.g. the recently proposed clasp

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

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

and claspD (for disjunctive clauses) [7] could be beneficially used to support the computation of extensions and query answering in abstract argumentation.

It would be interesting to perform a comparative performance analysis of the methods presented in section 3.1 to identify the most efficient method to support applications.

With the exception of few works (see section 3.3) ASP has been deployed to support *abstract argumentation*. However, the majority of applications of argumentation, e.g. medical decision making [26] and legal reasoning [4], require concrete argumentation frameworks, where arguments and attacks are built from knowledge bases "on demand" (namely depending on given queries/claims to be argued for or against). It would thus be useful to see whether other forms of argumentation could be fruitfully computed using ASP. In particular, there are a number of approaches to argumentation based on logic programming (e.g. [39]), extending logic programming (e.g. assumption-based argumentation [9, 8, 15]), or based upon logic (e.g. [7]) that may benefit from computational models supported by ASP.

In several of the existing applications of argumentation, the "explanation" of answers given to specific queries (namely claims) matters more than the actual answers. These explanations are given in terms of arguments in favour and arguments against the claims. Then, it would be useful to see whether these explanations could be usefully extracted from the extensions computed by means of ASP. A step in this direction has been made with the ASPARTIX system (see section 3.2), that includes a graphical interface labelling nodes of argumentation graphs that visuale abstract argumentation frameworks. In general, however, and especially when the graph is large, only relevant parts of the graph should be presented to be useful to users (as the presentation of full extensions may "cloud" matters).

5 Conclusions

Argumentation and answer set programming are the two main knowledge representation paradigms 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. These approaches 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.

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

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.

References

1. L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Ann. Math. Artif. Intell.*, 34(1-3):197–215, 2002.
2. L. Amgoud, C. Cayrol, M.-C. Lagasquie-Schiex, and P. Livet. On bipolarity in argumentation frameworks. *Int. J. Intell. Syst.*, 23(10):1062–1093, 2008.
3. P. Baroni, M. Giacomin, and G. Guida. Scc-recursiveness: a general schema for argumentation semantics. *Artif. Intell.*, 168(1-2):162–210, 2005.
4. T. Bench-Capon, H. Prakken, and G. Sartor. Argumentation in legal reasoning. In I. Rahwan and G. Simari, editors, *Argumentation in AI*, pages 363–382. Springer, 2009.
5. T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.
6. P. Besnard and S. Doutre. Characterization of semantics for argument systems. In *KR*, pages 183–193. AAAI Press, 2004.
7. P. Besnard and A. Hunter. *Elements of Argumentation.* MIT Press, 2008.
8. A. Bondarenko, P. Dung, R. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93(1-2):63–101, 1997.
9. A. Bondarenko, F. Toni, and R. Kowalski. An assumption-based framework for non-monotonic reasoning. In *Proc. LPRNR'93*, pages 171–189. MIT Press, 1993.
10. F. Calimeri, S. Cozza, G. Ianni, and N. Leone. Computable functions in asp: Theory and implementation. In M. G. de la Banda and E. Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 407–424. Springer, 2008.
11. M. Caminada. On the issue of reinstatement in argumentation. In M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, editors, *JELIA*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.
12. M. Caminada. Semi-stable semantics. In P. E. Dunne and T. J. M. Bench-Capon, editors, *COMMA*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press, 2006.
13. C. Devred, S. Doutre, C. Lefvre, and P. Nicolas. Dialectical proofs for constrained argumentation. In P. Baroni, F. Cerutti, M. Giacomin, and G. Simari, editors, *Proceedings of the Third International Conference on Computational Models of Argument (COMMA'10)*, volume 216, pages 159–170. IOS Press, 2010.
14. P. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77:321–357, 1995.
15. P. Dung, R. Kowalski, and F. Toni. Assumption-based argumentation. In I. Rahwan and G. Simari, editors, *Argumentation in AI*, pages 199–218. Springer, 2009.
16. P. Dung, P. Mancarella, and F. Toni. Computing ideal sceptical argumentation. *Artif. Intell., Special Issue on Argumentation in Artificial Intelligence*, 171(10–15):642–674, 2007.

17. P. M. Dung. Negations as hypotheses: An abductive foundation for logic programming. In *ICLP*, pages 3–17, 1991.
18. P. M. Dung and P. M. Thang. A unified framework for representation and development of dialectical proof procedures in argumentation. In C. Boutilier, editor, *IJCAI*, pages 746–751, 2009.
19. P. E. Dunne. The computational complexity of ideal semantics. *Artif. Intell.*, 173(18):1559–1591, 2009.
20. U. Egly, S. A. Gaggl, P. Wandl, and S. Woltran. ASPARTIX conquers the web. In *COMMA*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2010.
21. U. Egly, S. A. Gaggl, and S. Woltran. ASPARTIX: Implementing argumentation frameworks using answer-set programming. In M. G. de la Banda and E. Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.
22. U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 2010. Accepted for publication.
23. T. Eiter and A. Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *TPLP*, 6(1-2):23–60, 2006.
24. K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *ICLP*, pages 234–254, 1989.
25. W. Faber and S. Woltran. Manifold answer-set programs for meta-reasoning. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2009.
26. J. Fox, D. Glasspool, D. Grecu, S. Modgil, M. South, and V. Patkar. Argumentation-based inference and decision making–a medical perspective. *IEEE Intelligent Systems*, 22(6):34–41, 2007.
27. S. Gaggl and S. Woltran. CF2 semantics revisited. In P. Baroni, F. Cerutti, M. Giacomin, and G. Simari, editors, *Proceedings of the Third International Conference on Computational Models of Argument (COMMA'10)*, volume 216, pages 243–254. IOS Press, 2010.
28. A. Garcia and G. Simari. Defeasible logic programming: An argumentative approach. *Journal of Theory and Practice of Logic Programming*, 4(1-2):95–138, 2004.
29. M. Gelfond. Answer sets. In *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier, 2007.
30. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.
31. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
32. C. Lefèvre and P. Nicolas. The first version of a new asp solver : Asperix. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 522–527. Springer, 2009.
33. J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
34. D. V. McDermott. Nonmonotonic logic ii: Nonmonotonic modal theories. *J. ACM*, 29(1):33–57, 1982.
35. R. C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.

36. J. C. Nieves, U. Cortés, and M. Osorio. Preferred extensions as stable models. *TPLP*, 8(4):527–543, 2008.
37. M. Osorio, J. C. Nieves, and I. Gómez-Sebastià. CF2-extensions as answer-set models. In P. Baroni, F. Cerutti, M. Giacomin, and G. Simari, editors, *Proceedings of the Third International Conference on Computational Models of Argument (COMMA'10)*, volume 216, pages 159–170. IOS Press, 2010.
38. D. Poole. A logical framework for default reasoning. *Artif. Intell.*, 36(1):27–47, 1988.
39. H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.
40. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.
41. M. Thimm and G. Kern-Isberner. On the relationship of defeasible argumentation and answer set programming. In P. Besnard, S. Doutre, and A. Hunter, editors, *COMMA*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 393–404. IOS Press, 2008.
42. A. Van Gelder, K. Ross, and J. Schlifp. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.
43. T. Wakaki and K. Nitta. Computing argumentation semantics in answer set programming. In H. Hattori, T. Kawamura, T. Idé, M. Yokoo, and Y. Murakami, editors, *JSAI*, volume 5447 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2009.
44. T. Wakaki, K. Nitta, and H. Sawamura. Computing abductive argumentation in answer set programming. In P. McBurney, I. Rahwan, S. Parsons, and N. Maudet, editors, *ArgMAS*, volume 6057 of *Lecture Notes in Computer Science*, pages 195–215. Springer, 2010.