This manuscript was published as:
Wolfgang Dvořák, Anna Rapberger, Johannes P. Wallner, and Stefan Woltran. ASPARTIX-V19 - An Answer-set Programming based System for Abstract Argumentation. In Herzig A., Kontinen J., editors, *Proceedings of the 11th International Symposium on Foundations of Information and Knowledge Systems, FoIKS 2020*, volume 12012 of *Lecture Notes in Computer Science*, pages 79–89, 2020. Springer. The final authenticated version is available online at `https://doi.org/10.1007/978-3-030-39951-1_5`.

ASPARTIX-V19 - An Answer-set Programming based System for Abstract Argumentation

Wolfgang Dvořák[0000−0002−2269−8193], Anna Rapberger[0000−0003−0355−3535], Johannes P. Wallner[0000−0002−3051−1966], and Stefan Woltran[0000−0003−1594−8972]

Institute of Logic and Computation, TU Wien, Vienna, Austria
{dvorak,arapberg,wallner,woltran}@dbai.tuwien.ac.at

Abstract. We present ASPARTIX-V, a tool for reasoning in abstract argumentation frameworks that is based on answer-set programming (ASP), in its 2019 release. ASPARTIX-V participated in this year's edition of the International Competition on Computational Models of Argumentation (ICCMA'19) in all classical (static) reasoning tasks. In this paper we discuss extensions the ASPARTIX suite of systems has undergone for ICCMA'19. This includes incorporation of recent ASP language constructs (e.g. conditional literals), domain heuristics within ASP, and multi-shot methods. In particular, with this version of ASPARTIX-V we partially deviate from an earlier focus on monolithic approaches (i.e., one-shot solving via a single ASP encoding) to further enhance performance. We also briefly report on the results achieved by ASPARTIX-V in ICCMA'19.

Keywords: abstract argumentation · argumentation system · answer-set programming.

1 Introduction

Abstract argumentation frameworks (AFs) as introduced by Dung [4] are a core formalism for many problems and applications in the field of formal argumentation. In a nutshell, AFs formalize statements as arguments together with a relation denoting conflicts between arguments. Semantics of these AFs give a handle to resolve the conflicts between statements by selecting coherent subsets of the arguments. This selection is solely based on the relation between the arguments and considers arguments as abstract entities. Several different semantics to select coherent subsets of arguments have already been proposed by Dung [4] but numerous other semantics have been introduced later on which lead to a multitude of argumentation semantics (see [1]).

A prominent line of research in the field of computational argumentation has focused on implementations of reasoning procedures for abstract argumentation (see, e.g., [2]) and cumulated in the biennial International Competition on Computational Models of Argumentation (ICCMA)[1] which has been established in

[1] www.argumentationcompetition.org

2015. There are two kinds of approaches to such systems. First, the direct approach of implementing dedicated algorithms for argumentation problems which are often based one some kind of labelling propagation (see, e.g., [20]). Second, the reduction-based approach where the argumentation problem is encoded in some other formalism for which sophisticated solvers already exist. Prominent target formalisms for the later are answer-set programming (ASP) [17,18] and propositional logic with SAT-solving technology; see [3] for an overview.

In this paper we consider the ASPARTIX[2] system that exploits ASP technology to solve argumentation reasoning problems and describe the ASPARTIX-V (Answer Set Programming Argumentation Reasoning Tool - Vienna) version in its 2019 edition which is dedicated to the reasoning tasks of ICCMA'19. We discuss the specifics of ASPARTIX-V19 and differences to earlier versions of ASPARTIX. This includes incorporation of recent ASP language constructs (e.g. conditional literals), domain heuristics within ASP, and multi-shot methods. In particular, with this version of ASPARTIX-V we partially deviate from an earlier focus on monolithic approaches (i.e., one-shot solving via a single ASP encoding) to further enhance performance. Moreover, we give a first analysis of the results achieved by ASPARTIX-V in ICCMA'19.

In the remainder of the paper we first recall the necessary argumentation background and the tracks of this years ICCMA competition. We then give an overview on the ASPARTIX system and explain the aim of our ASPARTIX-V19 edition. In the main part we discuss technical specifics of the ASPARTIX-V19 edition. Finally, we briefly discuss the performance of our system at ICCMA'19.

2 Preliminaries

In this section we briefly introduce the necessary background on abstract argumentation and discuss the tracks of the ICCMA'19 competition.

2.1 Abstract Argumentation

Let us introduce argumentation frameworks [4] and recall the semantics relevant for this work (for a comprehensive introduction, see [1]).

Definition 1. *An* argumentation framework (AF) *is a pair* $F = (A, R)$ *where* A *is a finite set of arguments and* $R \subseteq A \times A$ *is the attack relation. The pair* $(a, b) \in R$ *means that* a *attacks* b*, and we say that a set* $S \subseteq A$ *attacks (in* F*) an argument* b *if* $(a, b) \in R$ *for some* $a \in S$*. An argument* $a \in A$ *is* defended *(in* F*) by a set* $S \subseteq A$ *if each* b *with* $(b, a) \in R$ *is attacked by* S *in* F*.*

Semantics for argumentation frameworks are defined as functions σ which assign to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$, with each set $S \in \sigma(F)$ called an extension. We consider for σ the functions cf, grd, stb, adm, com, $ideal$, prf,

[2] www.dbai.tuwien.ac.at/research/argumentation/aspartix/

sem and *stg* which stand for conflict-free, grounded, stable, admissible, complete, ideal, preferred, semi-stable and stage extensions, respectively. Towards the definition of these semantics we introduce the following notation. For a set $S \subseteq A$, we denote the set of arguments attacked by (resp. attacking) S in F as $S_F^+ = \{x \mid S \text{ attacks } x \text{ in } F\}$ (resp. $S_F^- = \{x \mid x \text{ attacks some } s \in S \text{ in } F\}$), and define the *range of* S in F as $S_F^\oplus = S \cup S_F^+$.

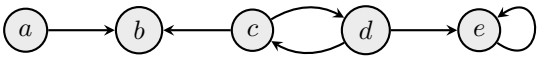
We are now prepared to give the formal definitions of the abstract argumentation semantics we will consider.

Definition 2. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in F)*, if there are no $a, b \in S$, such that $(a, b) \in R$. $cf(F)$ denotes the collection of conflict-free sets of F. For a conflict-free set $S \in cf(F)$, it holds that*

- *$S \in stb(F)$, if each $a \in A \setminus S$ is attacked by S in F;*
- *$S \in adm(F)$, if each $a \in S$ is defended by S in F;*
- *$S \in com(F)$, if $S \in adm(F)$ and each $a \in A$ defended by S in F is contained in S;*
- *$S \in grd(F)$, if $S \in com(F)$ and there is no $T \subset S$ such that $T \in com(F)$;*
- *$S \in prf(F)$, if $S \in adm(F)$ and there is no $T \supset S$ such that $T \in adm(F)$;*
- *$S \in ideal(F)$, if S is a $\subseteq$-maximal admissible set that is contained in each preferred extension of F;*
- *$S \in sem(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ with $S_R^\oplus \subset T_R^\oplus$;*
- *$S \in stg(F)$, if there is no $T \in cf(F)$, with $S_R^\oplus \subset T_R^\oplus$.*

Notice that $grd(F)$, $ideal(F)$ respectively, always yields a unique extension, the grounded, ideal respectively, extension of F.

Example 1. Consider the AF $F = (A, R)$, with arguments $A = \{a, b, c, d, e\}$ and attacks $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. The graph representation of F is as follows.



Considering the extensions of F, we have $stb(F) = stg(F) = sem(F) = \{\{a, d\}\}$. The admissible sets of F are $\emptyset$, $\{a\}$, $\{c\}$, $\{d\}$, $\{a, c\}$ and $\{a, d\}$ and thus the set of preferred extensions is $prf(F) = \{\{a, c\}, \{a, d\}\}$ and the complete extensions are $\{a\}$, $\{a, c\}$ and $\{a, d\}$. Finally, the grounded extension is $\{a\}$ and coincides with the ideal extension. ◇

2.2 Tracks of ICCMA'19

ICCMA'19 [3] is the third edition of the International Competition on Computational Models of Argumentation (ICCMA) and had two types of tracks, the classical tracks and the novel dynamic tracks. In the classical tracks the solver

[3] https://www.iccma2019.dmi.unipg.it/

is given an argumentation framework and has to solve a specific reasoning task while in the dynamic tracks the solver is given an initial argumentation frameworks and a list of updates to that framework and the reasoning task has to be evaluated after each update to the framework. As the ASPARTIX-V system supports only the classical tracks we will focus on these tracks here.

For the classical tracks ICCMA'19 considers the following four reasoning tasks, that correspond to the standard reasoning problems studied in the literature (see, e.g., [7]).

- DC-σ: Decide Credulous acceptance of an argument w.r.t. a semantics σ: Given $F = (A, R)$, $a \in A$ decide whether $a \in E$ for some extension $E \in \sigma(F)$.
- DS-σ: Decide Skeptical acceptance of an argument w.r.t. a semantics σ: Given $F = (A, R)$, $a \in A$ decide whether $a \in E$ for all extensions $E \in \sigma(F)$.
- SE-σ: compute Some σ-Extension: Given $F = (A, R)$ return some $E \in \sigma(F)$.
- EE-σ: Enumerate all σ-Extensions: Given $F = (A, R)$ enumerate all $E \in \sigma(F)$.

For σ, seven semantics were considered, namely complete, preferred, stable, semi-stable, stage, grounded and ideal. This resulted in a total number of 24 classical tracks, as for $\sigma \in \{ideal, grd\}$ (the semantics with a unique extension) we have DC-σ = DS-σ and SE-σ = EE-σ.

3 The ASPARTIX System and its V19 Edition

The ASPARTIX system was one of the first systems that supported efficient reasoning for a broad collection of abstract argumentation semantics starting with the work of Gaggl et al. (see, e.g., [11]) and has been continuously expanded and improved since then (see, e.g., [8,9,13,21,10]). However, the system is not limited to abstract argumentation frameworks but also supports enhancements of AFs by, e.g., preferences or recursive attacks. It is thus frequently used as reference system in the literature.

ASPARTIX is based on answer-set programming (ASP) and the idea of characterizing argumentation semantics via ASP encodings. With such an encoding of a semantics one can easily apply state-of-art systems for ASP to solve diverse reasoning tasks or to enumerate all extensions of a given AF. Given an AF as input, in the `apx` format of ICCMA, ASPARTIX delegates the main reasoning to an answer set programming solver (e.g., [15]), with answer set programs encoding the argumentation semantics and reasoning tasks. The basic workflow is shown in Figure 1, i.e., the AF is given in `apx` format (facts in the ASP language), and the AF semantics and reasoning tasks are encoded via ASP rules, possibly utilizing further ASP language constructs. For more information on the ASPARTIX system and its derivatives in general the interested reader is referred to the systems web-page:

www.dbai.tuwien.ac.at/research/argumentation/aspartix/

In this work we shall focus on ASPARTIX-V19 which is a derivative of ASPARTIX tuned towards the tracks of ICCMA'19. That is, ASPARTIX-V19 is

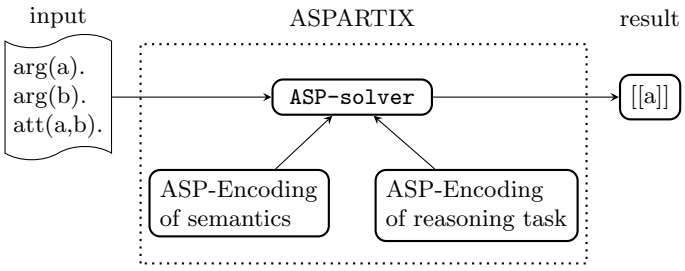


Fig. 1. Basic workflow of ASPARTIX

restricted to AFs and supports all the standard tasks of ICCMA'19, i.e. credulous/skeptical acceptance and computing all/some extension(s) for complete, preferred, stable, semi-stable, stage, grounded, and ideal semantics. In the following we highlight specifics of the current version and in particular differences to prior versions. In this instance of the argumentation competition, the software systems were collected as docker containers. The competition version ASPARTIX-V19 is available at

https://hub.docker.com/r/aspartix19/aspartix19-repo.

In this competition version of the ASPARTIX system we deviate from classical ASPARTIX design virtues. First, while traditional ASPARTIX encodings are modular in the sense that fixed encodings for semantics can be combined with the generic encodings of reasoning tasks, we use semantics encodings specific to a reasoning task. Second, when appropriate, we apply multi-shot methods for reasoning, which is in contrast to the earlier focus on so-called monolithic encodings, where one uses a single ASP-encoding and runs the solver only once (as illustrated in Figure 1). Third we make use of advanced features of the ASP-language, and utilize clingo v5.3.0 and v4.4.0 [4] [15].

Next, we list and overview some of the ASP-techniques novel to the ASPARTIX system. First, we exploit the concept of conditional literals [14, Section 3.1.11], which has first been applied for ASP-encodings of argumentation semantics in [13]. For example we simplified the encoding of grounded semantics (cf. Listing 1.1). Moreover, conditional literals enable us to give ASPARTIX style encodings of the translations from AF semantics to ASP semantics provided in [22]. Second, we exploit clingo domain heuristics [16] (see also [14, Chapter 10]), in order to compute subset-maximal extensions while only specifying constraints for the base semantics [12].

4 Implementation Details

When not stated otherwise, for a supported semantics we provide an ASP-encoding such that when combined with an AF in the `apx` format the answer-sets

[4] https://potassco.org/

of the program are in a one-to-one correspondence with the extensions of the AF. Given an answer-set of such an encoding the corresponding extension is given by the $\texttt{in}(\cdot)$ predicate, i.e., an argument $\texttt{a}$ is in the extensions iff $\texttt{in}(\texttt{a})$ is in the answer-set. With such an encoding we can exploit a standard ASP-solver to compute some extension (SE) by computing an answer-set; enumerate all extensions (EE) by enumerating all answer-sets; decide credulous acceptance (DC) of an argument a by adding the constraint $\leftarrow \texttt{in}(\texttt{a})$ to the program and testing whether the program is satisfiable, i.e., a is credulously accepted if there is at least one answer set; and decide skeptical acceptance (DS) of an argument a by adding the constraint $\leftarrow \mathit{not}\ \texttt{in}(\texttt{a})$ to the program and testing whether the program is unsatisfiable, i.e., a is skeptically accepted if there is no answer set.

4.1 Conditional Literals

We make use of the *conditional literal* [14]. In the head of a disjunctive rule literals may have conditions, e.g. consider the head of rule "$\mathbf{p}(X) : \mathbf{q}(X) \leftarrow$". Intuitively, this represents a head of disjunctions of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true. Rules might as well have conditions in their body, e.g. consider the body of rule "$\leftarrow \mathbf{p}(X) : \mathbf{q}(X)$", which intuitively represents a conjunction of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true.

A bottleneck of previous encodings for grounded semantics was the grounding step of the solver, i.e., the instantiation of variables with constants typically produces large programs. By utilizing conditional literals we were able to provide a compact encoding (cf. Listing 1.1) with significant smaller grounded programs.

Listing 1.1. Encoding for grounded semantics (using conditional literals)

```
in(X) ← arg(X), defeated(Y) : att(Y,X).
defeated(X) ← arg(X), in(Y), att(Y,X).
```

Moreover, conditional literals allow for an ASPARTIX style implementation of the translations from argumentation framework to grounded logic programs provided in [22]. For example consider our one line encoding of stable semantics in Listing 1.2 and the encoding of preferred semantics in Listing 1.3.

Listing 1.2. Encoding for stable semantics (using conditional literals)

```
in(Y) ← arg(Y), not in(X) : att(X,Y).
```

Listing 1.3. Encoding for preferred semantics (using conditional literals)

```
defended(X) | defeated(X) ← arg(X).
defended(X) ← arg(X), defeated(Y) : att(Y,X).
defeated(X) ← defended(Y), att(Y,X).
← defended(X), not defeated(Y), att(Y,X).
← defeated(X), not defended(Y) : att(Y,X).
in(X) ← defended(X), not defeated(X).
```

4.2 Domain Heuristics

Clingo provides an option to specify user-specific domain heuristics in the ASP-program which guides the ASP-solver. In particular one can define heuristics in order to select the answer-sets that are subset-maximal/minimal w.r.t. a specified predicate. Inspired by [12] we use such heuristics to compute preferred extensions by utilizing an encoding for complete semantics and identifying the subset-maximal answer-sets w.r.t. the `in(·)` predicate (cf. Listing 1.4). Moreover, we use domain heuristics and three-valued labelling-based characterizations of complete semantics via the predicates `in(·)`, `out(·)`, and `undec(·)` in order to compute the subset-maximal ranges of complete and conflict-free sets, i.e. we compute the subset-minimal answer-sets w.r.t. the `undec(·)` predicate. This can be exploited for computing some semi-stable or stage extensions. However, the domain heuristics only return one witnessing answer-set for each minima and thus this technique is not directly applicable to the corresponding enumerations tasks (we would miss some extensions if several extensions have the same range). In the next section we present a multi-shot method addressing this problem.

Listing 1.4. Encoding for preferred semantics (using domain heuristics)

```
%% Complete labellings
in(X) | out(X) | undec(X) ← arg(X).
in(X) ← arg(X), out(Y) : att(Y,X).
out(X) ← in(Y), att(Y,X).
← in(X), not out(Y), att(Y,X).
← out(X), not in (Y): att(Y,X).
← in(X), out(X).
← undec(X), out(X).
← undec(X), in(X).
%% We now apply heuristics to get the complete labeling with subset−maximal in(.) set
#heuristic in(X) : arg(X). [1,true]
```

4.3 Multi-shot Methods

We utilize multi-shot strategies and pre-processing of the AF for several semantics and reasoning tasks. In the current section, we briefly describe these methods.

For credulous and skeptical reasoning with complete, preferred, grounded, and ideal semantics we do not need to consider the whole framework but only those arguments that have a directed path to the query argument (notice that this does not hold true for stable, semi-stable and stage semantics). We perform pre-processing on the given AF that removes arguments without a directed path to the queried argument before starting the reasoning with an ASP-solver.

For computing the ideal extension we follow a two-shot strategy that is inspired by algorithms proposed earlier for ideal semantics [5,6]. That is, we first use an encoding for complete semantics and the brave reasoning mode of clingo to compute all arguments that are credulously accepted/attacked w.r.t. preferred

semantics. Second, we use the outcome of the first call together with an encoding that computes a fixed-point corresponding to the ideal extension. For reasoning with ideal semantics we use an encoding for ideal sets and perform credulous reasoning on ideal sets in the standard way.

Semi-stable extensions correspond to those complete labellings for which the set of undecided arguments is subset-minimal. In our approach, we utilize an encoding for complete semantics extended by an `undec(·)` predicate and process the answer-sets. We check whether models without an `undec(·)` predicate have been computed; in that case, semi-stable extensions coincide with stable extensions. In the other case, we compute all subset-minimal sets among all undecided sets using the set class in python and return the corresponding models.

For enumerating stage extensions we use a multi-shot strategy. First we use the domain heuristics to compute the maximal ranges w.r.t. naive semantics (as each range maximal conflict-free set is also subset-maximal it is sufficient to only consider naive sets, i.e. subset-maximal conflict free sets). Second, for each of the maximal ranges we start another ASP-encoding which computes conflict-free sets with exactly that range (this is equivalent to computing stable extension of a restricted framework). Each of these extensions corresponds to a different stage extension of the AF.

For reasoning with semi-stable and stage semantics we use a multi-shot strategy similar to that for enumerating the stage extensions. First we use domain heuristics to compute the maximal ranges w.r.t. complete or naive semantics. In the second step we iterate over these ranges and perform skeptical (credulous) reasoning over complete extensions (conflict-free sets) with the given range. For skeptical acceptance, we answer negatively as soon as a counterexample to a positive answer is found when iterating the extensions; otherwise, after processing all maximal ranges we answer with YES. Analogously, for credulous acceptance, we check in each iteration whether we can report a positive answer; otherwise, after processing all maximal ranges, we return NO.

5 Discussion

We next briefly discuss the performance of our system at ICCMA'19 (detailed results of the competition are published at https://www.iccma2019.dmi.unipg.it/results/results-main.html). The competition was dominated by the μ-toksia system by Niskanen and Järvisalo [19], an optimized system based on modern SAT-solving technology which won all the tracks of the competition and only failed to solve two of the benchmark instances in the given time-limit of 600 seconds.

The ASPARTIX-V19 system scored third in the overall evaluation of the competition, scored second in 8 of the 24 tracks and scored second in the aggregated evaluation of complete and stable semantics. Moreover, for 16 tracks ASPARTIX-V19 solved all instances of the competition within the given time-limit. Noteworthy, ASPARTIX-V19 was to only system to solve the enumeration task under stage semantics for the `n256p3q08n.apx` instance.

The ICCMA'19 results also reported different kinds of errors in the results of the ASPARTIX-system, which we investigated and shall discuss in the following. This errors include wrong results, malformed output, crashed computations and for enumeration tasks incomplete list of extensions which are not due to a timeout. The affected tasks are skeptical acceptance under preferred and semi-stable semantics, credulous acceptance under semi-stable semantics, stage and ideal semantics and enumeration of semi-stable and stage semantics.

The main reason for these errors seems to be side-effects of concurrent calls to the solver. Towards understanding the erroneous results, we performed additional experiments. For these experiments we considered all skeptical and credulous acceptance instances of the competition where ASPARTIX-V19 returned an erroneous result or crashed and reran the ASPARTIX-V19 docker on these instances in an isolated setting. For all but one instance we got the correct results. In this isolated setting ASPARTIX-V19 only reported one wrong result for skeptical reasoning with semi-stable semantics on the `Small-result-b86.apx` instance. This seems to be due to a bug in the used ASP solver, which can be resolved by using an earlier version of the solver (we got correct results with clingo 4.4.0). We maintain an updated and extended version of ASPARTIX-V19, available at the systems web-page [5]. For the enumeration tasks we investigated selected instances with erroneous / incomplete results and again got correct results when running them in an isolated setting and on the other hand could generate erroneous results by concurrent calls to the solver.

From our development work and the results achieved in the international competition, we conclude that (i) a performance increase was achieved by utilizing advanced language features of ASP, across multiple reasoning tasks covering several levels of complexity of the polynomial hierarchy (e.g., argumentative reasoning tasks considered in the ICCMA range from polynomial-time decidable to being complete for a class on the second level of the polynomial hierarchy), (ii) said language features, furthermore, provide means for compact and accessible modeling of problem shortcuts in the ASP language, however care needs to be taken when designing systems that interface ASP solvers, and (iii) while our prototype was outperformed by the SAT based approach of μ-toksia, performance of ASPARTIX-V19 does not lag behind for several cases. Indeed, as witnessed by the uniquely solved instance only by ASPARTIX-V19, certain shortcuts included in ASPARTIX-V19 can lead to complementary performance for families of instances.

Acknowledgments. The authors are grateful to a reviewer for suggesting directions for further improvements in the encodings.

This work has been funded by the Austrian Science Fund (FWF): P30168-N31, W1255-N23, and I2854.

[5] https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/dung.html#iccma_interface.

References

1. Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. Abstract argumentation frameworks and their semantics. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 4. College Publications, February 2018.
2. Federico Cerutti, Sarah A. Gaggl, Matthias Thimm, and Johannes P. Wallner. Foundations of implementations for formal argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15. College Publications, February 2018. Also available as an article in the IfCoLog Journal of Logics and their Applications 4(8):2623–2706.
3. Günther Charwat, Wolfgang Dvořák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.
4. Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artif. Intell.*, 77(2):321–358, 1995.
5. Paul E. Dunne. The computational complexity of ideal semantics. *Artif. Intell.*, 173(18):1559–1591, 2009.
6. Paul E. Dunne, Wolfgang Dvořák, and Stefan Woltran. Parametric properties of ideal semantics. *Artif. Intell.*, 202:1–28, 2013.
7. Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14. College Publications, February 2018. Also available as an article in the IfCoLog Journal of Logics and their Applications 4(8):2557–2622.
8. Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proc. INAP, Revised Selected Papers*, volume 7773 of *Lecture Notes in Artificial Intelligence*, pages 114–133. Springer, 2013.
9. Wolfgang Dvořák, Sarah Alice Gaggl, Thomas Linsbichler, and Johannes Peter Wallner. Reduction-based approaches to implement Modgil's extended argumentation frameworks. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2015.
10. Wolfgang Dvořák, Alexander Greßler, and Stefan Woltran. Evaluating SETAFs via answer-set programming. In Matthias Thimm, Federico Cerutti, and Mauro Vallati, editors, *Proc. SAFA co-located with COMMA 2018*, volume 2171 of *CEUR Workshop Proceedings*, pages 10–21. CEUR-WS.org, 2018.
11. Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.
12. Wolfgang Faber, Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Enumerating preferred extensions using ASP domain heuristics: The ASPrMin solver. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors,

Proc. COMMA, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 459–460. IOS Press, 2018.
13. Sarah Alice Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. Improved answer-set programming encodings for abstract argumentation. *Theory and Practice of Logic Programming*, 15(4-5):434–448, 2015.
14. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Philipp Wanko Sven Thiele. Potassco guide version 2.2.0. https://github.com/potassco/guide/releases/tag/v2.2.0, 2019.
15. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.
16. Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In Marie desJardins and Michael L. Littman, editors, *Proc. AAAI*, pages 350–356. AAAI Press, 2013.
17. Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.
18. Ilkka Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.
19. Andreas Niskanen and Matti Järvisalo. μ-toksia participating in ICCMA 2019. https://www.iccma2019.dmi.unipg.it/papers/ICCMA19_paper_11.pdf, 2019.
20. Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.*, 207:23–51, 2014.
21. Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. ASPARTIX-V: utilizing improved ASP encodings. http://argumentationcompetition.org/2015/pdf/paper_11.pdf, 2015.
22. Chiaki Sakama and Tjitze Rienstra. Representing argumentation frameworks in answer set programming. *Fundam. Inform.*, 155(3):261–292, 2017.