Argument & Computation -1 (2024) 1–7
DOI 10.3233/AAC-230019
IOS Press
CORRECTED PROOF



# pygarg: A Python engine for argumentation

Jean-Guy Mailly
*IRIT, Université Toulouse Capitole, France*
*E-mail: jean-guy.mailly@irit.fr*

**Abstract.** Recent advancements in algorithms for abstract argumentation make it possible now to solve reasoning problems even with argumentation frameworks of large size, as demonstrated by the results of the various editions of the International Competition on Computational Models of Argumentation (ICCMA). However, the solvers participating to the competition may be hard to use for non-expert programmers, especially if they need to incorporate these algorithms in their own code instead of simply using the command-line interface. Moreover, some ICCMA solvers focus on the ICCMA tracks, and do not implement algorithms for other problems. In this paper we describe `pygarg`, a Python implementation of the SAT-based approach used in the argumentation solver `CoQuiAAS`. Contrary to `CoQuiAAS` and most of the participants to the various editions of ICCMA, `pygarg` incorporates all problems that have been considered in the main track of any edition of ICCMA. We show how to easily use `pygarg` via a command-line interface inspired by ICCMA competitions, and then how it can be used in other Python scripts as a third-party library.

Keywords: Abstract argumentation, SAT-based solver, Python software

## 1. Introduction

Abstract argumentation [11] provides a simple framework for representing conflicting pieces of information and deducing which of them can be accepted. In his seminal paper, Dung shows how it can be used to represent problems such as non-monotonic reasoning or stable marriage problems. Recent works show various other applications like fair allocation of resources [20] or explainability of (black box) classification models [1]. Despite the generally high complexity of argumentation reasoning [13], recent advancements in SAT-based solving techniques for argumentation [18] have permitted to handle harder instances and problems, as can be seen from the results of the International Competition on Computational Models of Argumentation (ICCMA) [15]. However, from a practical point of view, the solvers participating to the competition tend to focus on the problems that are part of the competition track. This means, for instance, that most of the solvers that participated to the last editions of ICCMA (in 2021 and 2023) may not solve the extension enumeration task. If a user requires several types of reasoning tasks, he may need to use different argumentation solvers in the same project, which may increase the difficulty especially for non-expert programmers. This may make it difficult for some part of the community to actually implement and test their ideas. This is why we propose `pygarg`, a Python implementation of the SAT-based algorithms initially proposed in `CoQuiAAS` [18],[1] the solver that won the first edition of ICCMA in 2015.[2] While `pygarg` can be used with a command-line interface inspired by the ICCMA requirements, it is also easy to incorporate it as a third-party library in any Python script. So, anyone in

[1] Notice that most of these SAT-based techniques are also incorporated in `Crustabri`, which won several tracks at ICCMA 2023. See https://github.com/crillab/crustabri.

[2] http://argumentationcompetition.org/2015/index.html

need of solving problems for abstract argumentation can use `pygarg` for problems such as deciding the (credulous or skeptical) acceptability of an argument, computing one or all the extensions, or counting the extensions, for Dung's semantics [11], the semi-stable [8] and ideal semantics [12], with a single tool.

Section 2 recalls basic notions of abstract argumentation. Section 3 describes the design of `pygarg`, and how to use it either as a command-line tool or in one's own script. Section 4 concludes the paper.

## 2. Background notions

We start with a reminder of basic notions of abstract argumentation.

**Definition 1.** An *abstract argumentation framework* (AF) [11] is a directed graph $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is the (finite) set of *arguments* and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is the *attack relation*.

Dung does not assume the finiteness of the set of arguments, however it is the case for our implementation. Notice that the set of argument can be empty. We say that $a \in \mathcal{A}$ (respectively $S \subseteq \mathcal{A}$) *attacks* $b \in \mathcal{A}$ when $(a, b) \in \mathcal{R}$ (respectively some $a \in S$ attacks $b$). Then, a set of arguments $S$ *defends* an argument $a$ if $S$ attacks all the arguments attacking $a$. Classical reasoning with AFs is based on the notion of *extensions*, *i.e.* sets of jointly acceptable arguments. An extension must usually satisfy two basic properties: $S \subseteq \mathcal{A}$ is *conflict-free* if $\forall a, b \in S$, $(a, b) \notin \mathcal{R}$; and $S \subseteq \mathcal{A}$ is *self-defending* if $S$ defends all its elements. A set satisfying both these properties is *admissible*. We write $\mathbf{cf}(\mathcal{F})$ and $\mathbf{ad}(\mathcal{F})$ for the sets of conflict-free and admissible sets of $\mathcal{F}$. Then, classical Dung's semantics [11] are defined as follows.

**Definition 2.** Let $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ be an AF. Then, the set $S \subseteq \mathcal{A}$ is:

- a *complete* extension if $S \in \mathbf{ad}(\mathcal{F})$ and $S$ contains all the arguments that it defends;
- a *preferred* extension if $S$ is a $\subseteq$-maximal admissible set;
- a *stable* extension if $S \in \mathbf{cf}(\mathcal{F})$ and $S$ attacks all the arguments in $\mathcal{A} \setminus S$;
- a *grounded* extension if $S$ is a $\subseteq$-minimal complete extension.

We use $\mathbf{co}(\mathcal{F})$, $\mathbf{pr}(\mathcal{F})$, $\mathbf{st}(\mathcal{F})$ and $\mathbf{gr}(\mathcal{F})$ for these sets of extensions. It is well-known [11] that $|\mathbf{gr}(\mathcal{F})| = 1$ for any AF, that $\mathbf{st}(\mathcal{F}) \subseteq \mathbf{pr}(\mathcal{F})$, and preferred extensions also correspond to $\subseteq$-maximal complete extensions. Finally, from all the semantics studied in this paper, only the stable semantics may collapse, *i.e.* for any $\sigma \neq \mathbf{st}$, $\sigma(\mathcal{F}) \neq \emptyset$ for any AF. From the preferred semantics, one can define a "more skeptical" semantics as follows.

**Definition 3.** Let $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ be an AF. Then, the set $S \subseteq \mathcal{A}$ is an *ideal* extension [12] if it is a $\subseteq$-maximal admissible set included in all the preferred extensions.

We write $\mathbf{id}(\mathcal{F})$ the set of ideal extensions of an AF. Similarly to the grounded semantics, the ideal extension is unique for any AF. Finally, we also focus on one last semantics, based on the notion of range. Given an AF $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ and $a \in \mathcal{A}$, we write $a^+ = \{b \in \mathcal{A} | (a, b) \in \mathcal{R}\}$ the set of arguments attacked by $a$. We generalize it to sets, with $S^+ = \bigcup_{a \in S} a^+$ for the set of arguments attacked by $S$. The *range* of $S$ is $S^\oplus = S \cup S^+$, *i.e.* the set of arguments which are either members of $S$ or attacked by $S$.

**Definition 4.** Let $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ be an AF. Then, the set $S \subseteq \mathcal{A}$ is a *semi-stable* extension [8] if $S \in \mathbf{co}(\mathcal{F})$ and the range of $S$ is $\subseteq$-maximal among the ranges of all complete extensions of $\mathcal{F}$.

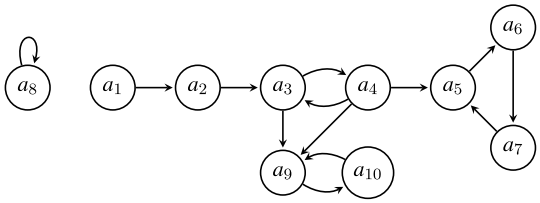


Fig. 1. An example of AF $\mathcal{F}$.

We write $\mathbf{sst}(\mathcal{F})$ for the semi-stable extensions. Notice that $\mathbf{st}(\mathcal{F}) \subseteq \mathbf{sst}(\mathcal{F})$, and if $\mathbf{st}(\mathcal{F}) \neq \emptyset$ then both semantics coincide, but $\mathbf{sst}(\mathcal{F}) \neq \emptyset$ even when there is no stable extension.

**Example 1.** Let $\mathcal{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ be the AF shown in Fig. 1. Its complete extensions are $\mathbf{co}(\mathcal{F}) = \{\{a_1\}, \{a_1, a_{10}\}, \{a_1, a_3, a_{10}\}, \{a_1, a_4, a_6, a_{10}\}\}$. Among them, the preferred extensions are $\mathbf{pr}(\mathcal{F}) = \{\{a_1, a_3, a_{10}\}, \{a_1, a_4, a_6, a_{10}\}\}$ (the $\subseteq$-maximal ones), and the grounded extension is $\mathbf{gr}(\mathcal{F}) = \{\{a_1\}\}$ (the $\subseteq$-minimal one). Among the preferred extensions, there is no stable extension (because of the self-attacking argument $a_8$), and only one semi-stable extension $\mathbf{st}(\mathcal{F}) = \{\{a_1, a_4, a_6, a_{10}\}\}$, which is also the unique semi-stable extension in this case. Finally, since the intersection of the preferred extensions is $\{a_1, a_{10}\}$, which is an admissible set, we deduce that the ideal extension is $\mathbf{id}(\mathcal{F}) = \{\{a_1, a_{10}\}\}$.

Reasoning with these semantics is generally hard, with complexity up to the second level of the polynomial hierarchy, depending on the semantics and the exact decision problem [13]. However, various implementations based on SAT solvers have been proposed for reasoning with abstract argumentation, mainly based on the logical encoding by Besnard and Doutre [3]. Describing in details these algorithms is out of the scope of this paper, but we refer the interested reader to [18], since our work is essentially a Python implementation of the SAT-based approach from `CoQuiAAS`.

## 3. Describing `pygarg`

`pygarg` is an open-source software,[3] implemented in Python, relying on `PySAT` [16] for performing calls to SAT solvers. In this section, we describe how `pygarg` can be used, either as a command-line tool, or as a library integrated to another Python code.

### *3.1. Using `pygarg` as a command-line tool*

The source code of `pygarg` is based on five Python files:

- `__main__.py` (in the main package) provides the command-line interface of the software,
- in the `dung` package,
  - `apx_parser.py` provides tools for parsing an APX file [14] (as used notably in ICCMA 2015, 2017, 2019 and 2021) into an argumentation framework usable by the solvers,
  - `dimacs_parser.py` provides tools for parsing a Dimacs file [17] (as used in ICCMA 2023) into an argumentation framework usable by the solvers,

[3] `pygarg` is available online: https://github.com/jgmailly/pygarg, under the GNU Lesser General Public License version 3. It is also available in the Python Package Index and can be installed with the commande line `pip install pygarg`.

```
$ python3 main.py -p EE-CO -fo apx -f test.apx
w a1
w a1 a4 a6
w a1 a3
```

Fig. 2. Enumerating extensions with `pygarg` on the command-line.

```
args = ["a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8", "a9", "a10"]
atts = [["a1", "a2"], ["a2", "a3"], ["a3", "a4"], ["a3", "a9"],
        ["a4", "a3"], ["a4", "a5"], ["a4", "a9"], ["a5", "a6"],
        ["a6", "a7"], ["a7", "a5"], ["a8", "a8"], ["a9", "a10"],
        ["a10", "a9"]]
```

Listing 1. Python data structure for the sets of arguments and attacks.

* `encoding.py` provides the tools used to translate argumentation problems into SAT solving,
* `solver.py` provides the functions for solving argumentation reasoning tasks.

Using the `__main__.py` file, one can use a command-line interface reminiscent of the ICCMA requirements, which is described in details in the `README` file of the software. In summary, one can use the command-line options:

- `-p PROBLEM` to define the problem to be solved, with `PROBLEM` being `XX-YY` where `XX` is one of `DC`, `DS`, `SE`, `EE`, `CE` (for credulous acceptability, skeptical acceptability, computing some extension, enumerating extensions and counting extensions) and `YY` must be one of `CF`, `AD`, `ST`, `CO`, `PR`, `GR`, `ID`, `SST` corresponding to $\sigma \in \{$**cf**, **ad**, **st**, **co**, **pr**, **gr**, **id**, **sst**$\}$,
- `-fo FORMAT` to define the format of the input file describing an AF, which must be equal to `apx` or `dimacs`,
- `-f FILENAME` to specify the path to the input file,
- `-a ARGNAME` to specify the name of the argument to be checked (for `DC` and `DS` problems).

The output of these commands follows the requirements of ICCMA 2023. This means that (when possible), an extension is provided as a witness for the (non-)acceptability of an argument, in a line starting with `w`. The same syntax is used for providing one (or each) extension of the AF. For instance, if `test.apx` corresponds to the AF from Fig. 1, we obtain the result presented at Fig. 2.

In case there is an empty extension, a line with only `w` will be printed.

### 3.2. *Using `pygarg` in another Python program*

Now we focus on how to use `pygarg` in one's own Python code. The data structures used to represent an AF are simply a list of strings (representing the arguments names), and a list of lists of strings (representing the attacks). For instance, the AF from Fig. 1 corresponds to the structure from Listing 1.

Instead of manually constructing the list of arguments and attacks, one can use the `parse(filename)` function provided in both `apx_parser.py` and `dimacs_parser.py`. For instance, Listing 2 shows how to parse an `apx` file.

```
import pygarg.dung.apx_parser

args, atts = apx_parser.parse("test.apx")
```

Listing 2. Parsing a text file with pygarg.

```
from pygarg.dung import solver as solver
from pygarg.dung import apx_parser as parser

args, atts = parser.parse("test.apx")
for sem in ["CO", "PR", "GR", "ST", "SST", "ID"]:
  print(f"{sem}-extensions: ", end='')
  print(solver.extension_enumeration(args, atts, sem))
```

Fig. 3. Enumerating extensions with `pygarg` imported in one's own code.

```
CO-extensions: [['a1'], ['a1', 'a10'], ['a1', 'a4', 'a6', 'a10'], ['a1', 'a3', 'a10']]
PR-extensions: [['a1', 'a4', 'a6', 'a10'], ['a1', 'a3', 'a10']]
GR-extensions: [['a1']]
ST-extensions: []
SST-extensions: [['a1', 'a4', 'a6', 'a10']]
ID-extensions: [['a1', 'a10']]
```

Fig. 4. Result of running the extension enumeration.

Then, one needs to focus on some functions provided in the file `solver.py`:

- `credulous_acceptability(args, atts, argname, sem)` determines whether the argument `argname` is credulously accepted under the semantics `sem`,
- `skeptical_acceptability(args, atts, argname, sem)` determines whether the argument `argname` is skeptically accepted under the semantics `sem`,
- `compute_some_extension(args, atts, sem)` computes one `sem`-extension,
- `extension_enumeration(args, atts, sem)` enumerates all the `sem`-extensions,
- `extension_counting(args, atts, sem)` counts the number of `sem`-extensions.

In these functions, the `sem` argument must be a string describing the semantics, using the same conventions as the command-line interface (for instance, the complete semantics is described by `"CO"`).

**Example 2.** Continuing the previous example, Fig. 3 shows how we enumerate the extensions of the AF from Fig. 1, for the semantics $\sigma \in \{$**co**, **pr**, **gr**, **st**, **sst**, **id**$\}$. The result is shown in Fig. 4.

## 4. Discussion

As far as we know, the most similar implementation of reasoning tasks for abstract argumentation is `PyArg` [6,22]. However, the focus of `PyArg` is not on efficient algorithms, but rather on its graphical interface[4] and various other advanced features like computing explanations of acceptability or structured argumentation, which are not included in the current version of `pygarg`. For this reason, the algorithms

[4] See https://pyarg.npai.science.uu.nl.

included in this platform are more "naive" (for instance they are not based on SAT solving techniques), and thus they do not scale up as well as SAT-based algorithms. This is not a major problem for the purpose of `PyArg`, which is visualisation (and one can assume that users interested in visualising graphs do not use large graphs with dozens or hundreds of arguments). An empirical evaluation [21] shows that `pygarg` clearly outperforms `PyArg` for all tasks except the ones related to the (polynomially computable) grounded semantics.

For future work, we envision various possible directions. A first one would be to replace "naive" SAT-based algorithms by more efficient ones when possible (for instance, the current implementation of skeptical acceptability under the preferred semantics and reasoning with the ideal semantics is based on the enumeration of preferred extensions, but it could benefit from the techniques proposed by [24]). We are also interested in implementing algorithms for other semantics (like the stage semantics [25], or the more challenging semantics based on weak admissibility [2]), or other problems (like verifying if a set of arguments is an extension). Still in line with recent ICCMA competitions, we would like to incorporate techniques for dynamic re-computation of extensions when an AF evolves [4], or approximation algorithms (in the spirit of [10]). Other problems related to extension-based semantics could be added, like counting the number of extensions (not) containing a given argument. The labelling-based [7] counterpart of the problems already implemented could also be added. Finally, more long term projects include the integration of gradual and ranking-based semantics [5], as well as more general argumentation frameworks like Bipolar AFs [9], Strength-based AFs [23] or Incomplete AFs [19].

## Acknowledgement

This work is supported by the ANR project AGGREEY (ANR-22-CE23-0005) and the chair AIDAL (ANR-22-CPJ1-0061-01).

## References

[1] L. Amgoud, Non-monotonic explanation functions, in: *Proceedings of the 16th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2021*, Lecture Notes in Computer Science, Vol. 12897, Springer, 2021, pp. 19–31. doi:10.1007/978-3-030-86772-0_2.

[2] R. Baumann, G. Brewka and M. Ulbricht, Shedding new light on the foundations of abstract argumentation: Modularization and weak admissibility, *Artif. Intell.* **310** (2022), 103742. doi:10.1016/j.artint.2022.103742.

[3] P. Besnard and S. Doutre, Checking the acceptability of a set of arguments, in: *10th International Workshop on Non-monotonic Reasoning (NMR 2004)*, 2004, pp. 59–64, http://www.pims.math.ca/science/2004/NMR/papers.html.

[4] S. Bistarelli, L. Kotthoff, F. Santini and C. Taticchi, Containerisation and dynamic frameworks in ICCMA'19, in: *Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) Co-Located with the 7th International Conference on Computational Models of Argument (COMMA 2018)*, CEUR Workshop Proceedings, Vol. 2171, CEUR-WS.org, 2018, pp. 4–9, https://ceur-ws.org/Vol-2171/paper_1.pdf.

[5] E. Bonzon, J. Delobelle, S. Konieczny and N. Maudet, A comparative study of ranking-based semantics for abstract argumentation, in: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI Press, 2016, pp. 914–920. doi:10.1609/aaai.v30i1.10116.

[6] A. Borg and D. Odekerken, PyArg for solving and explaining argumentation in Python: Demonstration, in: *Computational Models of Argument – Proceedings of COMMA 2022*, Frontiers in Artificial Intelligence and Applications, Vol. 353, IOS Press, 2022, pp. 349–350. doi:10.3233/FAIA220167.

[7] M. Caminada, On the issue of reinstatement in argumentation, in: *Proceedings of the 10th European Conference on Logics in Artificial Intelligence, JELIA 2006*, Lecture Notes in Computer Science, Vol. 4160, Springer, 2006, pp. 111–123. doi:10.1007/11853886_11.

[8] M.W.A. Caminada, W.A. Carnielli and P.E. Dunne, Semi-stable semantics, *J. Log. Comput.* **22**(5) (2012), 1207–1254. doi:10.1093/logcom/exr033.

[9] C. Cayrol and M.-C. Lagasquie-Schiex, Bipolarity in argumentation graphs: Towards a better understanding, *Int. J. Approx. Reason.* **54**(7) (2013), 876–899. doi:10.1016/j.ijar.2013.03.001.
[10] J. Delobelle, J.-G. Mailly and J. Rossit, Revisiting approximate reasoning based on grounded semantics, in: *Seventeenth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2023)*, 2023.
[11] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artif. Intell.* **77**(2) (1995), 321–358. doi:10.1016/0004-3702(94)00041-X.
[12] P.M. Dung, P. Mancarella and F. Toni, Computing ideal sceptical argumentation, *Artif. Intell.* **171**(10–15) (2007), 642–674. doi:10.1016/j.artint.2007.05.003.
[13] W. Dvorák and P.E. Dunne, Computational problems in formal argumentation and their complexity, in: *Handbook of Formal Argumentation*, College Publications, 2018, pp. 631–688.
[14] U. Egly, S.A. Gaggl and S. Woltran, ASPARTIX: Implementing argumentation frameworks using answer-set programming, in: *Proceedings of the 24th International Conference on Logic Programming, ICLP 2008*, Lecture Notes in Computer Science, Vol. 5366, Springer, 2008, pp. 734–738. doi:10.1007/978-3-540-89982-2_67.
[15] S.A. Gaggl, T. Linsbichler, M. Maratea and S. Woltran, Design and results of the second international competition on computational models of argumentation, *Artif. Intell.* **279** (2020). doi:10.1016/j.artint.2019.103193.
[16] A. Ignatiev, A. Morgado and J. Marques-Silva, PySAT: A Python toolkit for prototyping with SAT Oracles, in: *SAT*, 2018, pp. 428–437. doi:10.1007/978-3-319-94144-8_26.
[17] M. Järvisalo, T. Lehtonen and A. Niskanen, Design of ICCMA 2023, 5th international competition on computational models of argumentation: A preliminary report (invited paper), in: *Proceedings of the First International Workshop on Argumentation and Applications (Arg&App 2023) Co-Located with 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, CEUR Workshop Proceedings, Vol. 3472, CEUR-WS.org, 2023, pp. 4–10, https://ceur-ws.org/Vol-3472/invited1.pdf.
[18] J.-M. Lagniez, E. Lonca and J.-G. Mailly, CoQuiAAS: A constraint-based quick abstract argumentation solver, in: *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015*, IEEE Computer Society, 2015, pp. 928–935. doi:10.1109/ICTAI.2015.134.
[19] J.-G. Mailly, Yes, no, maybe, I don't know: Complexity and application of abstract argumentation with incomplete knowledge, *Argument Comput.* **13**(3) (2022), 291–324. doi:10.3233/AAC-210010.
[20] J.-G. Mailly, Abstract argumentation applied to fair resources allocation: A preliminary study, in: *Proceedings of the First International Workshop on Argumentation and Applications (Arg&App 2023) Co-Located with 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, CEUR Workshop Proceedings, Vol. 3472, CEUR-WS.org, 2023, pp. 85–91, https://ceur-ws.org/Vol-3472/short2.pdf.
[21] J.-G. Mailly, pygarg: A Python engine for argumentation, Technical report, IRIT/RR–2024–02–FR, IRIT, Institut de Recherche en Informatique de Toulouse, 2024.
[22] D. Odekerken, A. Borg and M. Berthold, Accessible algorithms for applied argumentation, in: *Proceedings of the First International Workshop on Argumentation and Applications (Arg&App 2023) Co-Located with 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, CEUR Workshop Proceedings, Vol. 3472, CEUR-WS.org, 2023, pp. 92–98, https://ceur-ws.org/Vol-3472/short3.pdf.
[23] J. Rossit, J.-G. Mailly, Y. Dimopoulos and P. Moraitis, United we stand: Accruals in strength-based argumentation, *Argument Comput.* **12**(1) (2021), 87–113. doi:10.3233/AAC-200904.
[24] M. Thimm, F. Cerutti and M. Vallati, Skeptical reasoning with preferred semantics in abstract argumentation without computing preferred extensions, in: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, ijcai.org, 2021, pp. 2069–2075. doi:10.24963/ijcai.2021/285.
[25] B. Verheij, Two approaches to dialectical argumentation: Admissible sets and argumentation stages, in: *Proceedings of the Eighth Dutch Conference on Artificial Intelligence (NAIC'96)*, 1996, pp. 357–368.