# The Third International Competition on Computational Models of Argumentation (ICCMA'19)*

Solver Requirements

Stefano Bistarelli Lars Kotthoff Theofrastos Mantadelis Francesco Santini
Carlo Taticchi

January 24, 2019

This document contains requirements for solvers participating at ICCMA'19. In particular, this document provides some formal background on abstract argumentation, a list of computation problems (tasks) considered in the competition, the input formats of benchmark instances, the expected output format of results, and a description of the interface participating solvers are required to provide in a Docker container.

## 1 Abstract Argumentation

An *abstract argumentation framework* (AF, for short) (Dung, 1995) is a tuple $\mathcal{F} = (\mathsf{A}, \rightarrow)$ where $\mathsf{A}$ is a set of arguments and $\rightarrow$ is a relation $\rightarrow \subseteq \mathsf{A} \times \mathsf{A}$. For two arguments $a, b \in \mathsf{A}$ the relation $a \rightarrow b$ means that argument $a$ *attacks* argument $b$. An argument $a \in \mathsf{A}$ is *defended* by $S \subseteq \mathsf{A}$ (in $\mathcal{F}$) if for each $b \in \mathsf{A}$ such that $b \rightarrow a$ there is some $c \in S$ such that $c \rightarrow b$. A set $E \subseteq \mathsf{A}$ is *conflict-free* (in $\mathcal{F}$) if and only if there are no $a, b \in E$ with $a \rightarrow b$. $E$ is *admissible* (in $\mathcal{F}$) if and only if it is conflict-free and each $a \in E$ is defended by $E$. Finally, the range of $E$ (in $\mathcal{F}$) is given by $E^+_{\mathcal{F}} = E \cup \{a \in \mathsf{A} \mid \exists b \in E : b \rightarrow a\}$.

Semantics are used to determine sets of jointly acceptable arguments by mapping each AF $\mathcal{F} = (\mathsf{A}, \rightarrow)$ to a set of extensions $\sigma(\mathcal{F}) \subseteq 2^{\mathsf{A}}$. The extensions under complete, preferred, stable, semi-stable (Caminada *et al.*, 2012), stage (Verheij, 1996), grounded and ideal (Dung *et al.*, 2007) semantics are defined as follows. Given an AF $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and a set $E \subseteq \mathsf{A}$,

- $E \in \mathbf{CO}(\mathcal{F})$ iff $E$ is admissible in $\mathcal{F}$ and if $a \in \mathsf{A}$ is defended by $E$ in $\mathcal{F}$ then $a \in E$,
- $E \in \mathbf{PR}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and there is no $E' \in \mathbf{CO}(\mathcal{F})$ s.t. $E' \supset E$,
- $E \in \mathbf{ST}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and $E^+_{\mathcal{F}} = \mathsf{A}$,
- $E \in \mathbf{SST}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and there is no $E' \in \mathbf{CO}(\mathcal{F})$ s.t. $E'^+_{\mathcal{F}} \supset E^+_{\mathcal{F}}$,
- $E \in \mathbf{STG}(\mathcal{F})$ iff $E$ is conflict-free in $\mathcal{F}$ and there is no $E'$ that is conflict-free in $\mathcal{F}$ s.t. $E'^+_{\mathcal{F}} \supset E^+_{\mathcal{F}}$,

*Adapted from last edition's version by Sarah A. Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran.

- $E \in \mathbf{GR}(\mathcal{F})$ iff $E \in \mathbf{CO}(\mathcal{F})$ and there is no $E' \in \mathbf{CO}(\mathcal{F})$ s.t. $E' \subset E$,
- $E \in \mathbf{ID}(\mathcal{F})$ if and only if $E$ is admissible, $E \subseteq \bigcap \mathbf{PR}(\mathcal{F})$ and there is no admissible $E' \subseteq \bigcap \mathbf{PR}(\mathcal{F})$ s.t. $E' \supset E$.

For more discussion on these semantics see (Baroni *et al.*, 2011).

Note that both grounded and ideal extensions are uniquely determined and always exist (Dung, 1995; Dung *et al.*, 2007). Thus, they are also called *single-status* semantics. The other semantics introduced are *multi-status* semantics. That is, there is not always a unique extension induced by the semantics. In order to reason with multi-status semantics, usually, one takes either a credulous or skeptical perspective.

That is, given an AF $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and a semantics $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{GR}, \mathbf{ID}\}$, argument $a \in \mathsf{A}$ is

- *credulously accepted* in $\mathcal{F}$ under semantics $\sigma$ if there is a $\sigma$-extension $E \in \sigma(\mathcal{F})$ with $a \in E$, and
- *skeptically accepted* in $\mathcal{F}$ with semantics $\sigma$ if for all $\sigma$-extensions $E \in \sigma(\mathcal{F})$ it holds that $a \in E$.

Recall that stable semantics is the only case where an AF might possess no extension. In such a situation, each argument is defined to be skeptically accepted.

## 2 Dockerization

Solvers need to be packaged by participants in a Docker container (`https://www.docker.com`). The main advantage is to allow each solver to be delivered with its complete and intended run time environment, with the purpose to make setup and deployment easier; differences between participants' and organizers' running environment is thus avoided. Moreover, a dockerized application can be launched on different platforms (e.g., Windows, Linux, macOS, and in the cloud), making it possible to rerun the same experiments anywhere.

For more detailed information and guide to how to dockerize your solver, push it to a repository, and obtain a public link to it (to be submitted), please refer to `http://iccma19.dmi.unipg.it/res/ICCMA19_docker_manual.pdf`.

## 3 Dynamic Frameworks

In previous ICCMA editions, all the frameworks in each database were considered static in the sense that all the AFs were sequentially passed as input to solvers, representing different and independent information: all tasks are computed from scratch without taking any potentially useful knowledge from previous runs into account.

However, in many practical applications, an AF represents only a temporary situation: arguments and attacks can be added/retracted to take into account new knowledge that becomes available. For instance, in disputes among users of online social networks (Kökciyan *et al.*, 2016), arguments/attacks are repeatedly added/retracted by users to express their point of view in response to the last move made by the adversaries in the current digital polylogue (often disclosing as few arguments/attacks as possible).

The dynamics of frameworks has attracted recent and wide interest in the Argumentation community. We describe some related work, which also points to the research groups interested

in the organisation of such a track. In (Boella *et al.*, 2009), the authors investigate the principles according to which a grounded extension of a Dungs AF does not change when the set of arguments/attacks are changed. The work of (Cayrol *et al.*, 2010) studies how the extensions can evolve when a new argument is considered. The authors focus on adding one argument interacting with one starting argument (i.e., an argument which is not attacked by any other argument). In further work (Xu and Cayrol, 2015), the authors study the evolution of the set of extensions after performing a change operation (addition/removal of arguments/interaction). In (Baroni *et al.*, 2014), the authors propose a division-based method to divide the updated framework into two parts: "affected" and "unaffected". Only the status of affected arguments is recomputed after updates. A matrix-reduction approach that resembles the previous division method is presented in (Xu and Cayrol, 2015). A recent work that tests complete, preferred, stable, and grounded semantics on an AF and a set of updates is (Alfano *et al.*, 2017). This approach finds a reduced (updated) AF sufficient to compute an extension of the whole AF, and uses state-of-the-art algorithms to recompute an extension of the reduced AF only.

Modifications of AFs are also studied in the literature as a base to compute *robustness* measures of frameworks (Bistarelli *et al.*, 2018). In particular, by adding/removing an argument/attack, the set of extensions satisfying a given semantics may or may not change. For instance, one could be interested in computing the number of modifications needed to bring a change in this set, or measure the number of modifications needed to have a different set of extensions satisfying a desired semantics. In the latter case, the user is interested in having an estimate on how distant two different points of views are; this kind of approach has also been proposed in (Baumann and Brewka, 2010).

# 4 Computational problems

ICCMA'19 offers the participation to 7 classical tracks (exactly the same standard tracks in ICCMA'17), and 4 new dynamic tracks.

## 4.1 Classical Tracks

Let $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{GR}, \mathbf{ID}\}$ be some semantics. We consider the following computational tasks[1]:

**SE-$\sigma$** **Given** $\mathcal{F} = (\mathsf{A}, \rightarrow)$, **return** some set $E \subseteq \mathsf{A}$ that is a $\sigma$-extension of $\mathcal{F}$.

**EE-$\sigma$** **Given** $\mathcal{F} = (\mathsf{A}, \rightarrow)$, **enumerate** all sets $E \subseteq \mathsf{A}$ that are $\sigma$-extensions of $\mathcal{F}$.

**DC-$\sigma$** **Given** $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and $a \in \mathsf{A}$, **decide** whether $a$ is credulously accepted in $\mathcal{F}$ under $\sigma$.

**DS-$\sigma$** **Given** $\mathcal{F} = (\mathsf{A}, \rightarrow)$ and $a \in \mathsf{A}$, **decide** whether $a$ is skeptically accepted in $\mathcal{F}$ under $\sigma$.

The following tasks will be tested in ICCMA'19:

**CO.** Complete Semantics (SE, EE, DC, DS);

**PR.** Preferred Semantics (SE, EE, DC, DS);

[1] With the exception of **DS-GR**, **EE-GR**, **DS-ID**, and **EE-ID**. Note that **DC-CO** and **DC-PR** are equivalent tasks (as **EE-GR** and **SE-GR**, **DS-GR** and **DC-GR**, **EE-ID** and **SE-ID**, **DS-ID** and **DC-ID**), but in order to allow the participation in the preferred track without implementing any task on the complete semantics, we repeat the task.

**ST.** Stable Semantics (SE, EE, DC, DS);

**SST.** Semi-stable Semantics (SE, EE, DC, DS);

**STG.** Stage Semantics (SE, EE, DC, DS);

**GR.** Grounded Semantics (only (SE) and (DC));

**ID.** Ideal Semantics (only (SE) and (DC)).

The combination of problems with semantics amounts to a total number of 24 tasks. Each solver participating to ICCMA'19 can support, i.e. compete in, an arbitrary set of tasks. If a solver supports all the tasks of a track (e.g., the track on the complete semantics), it also automatically participates in the corresponding track.

### 4.2 Dynamic Tracks

In addition, 4 new tracks are dedicated to the solution of problems over dynamic argumentation frameworks. In this case, an instance consists of an initial framework and an additional file storing a sequence of additions/deletions of attacks. This file is provided through a simple text format, e.g., a sequence of *+att(a,b).* or *-att(d,e).* (see Section 5.3).

The dynamics tracks and their tasks are presented in the following list:

**CO.** Complete Semantics (SE, EE, DC, DS);

**PR.** Preferred Semantics (SE, EE, DC, DS);

**ST.** Stable Semantics (SE, EE, DC, DS);

**GR.** Grounded Semantics (only (SE) and (DC)).

The combination of problems with semantics amounts to a total number of 14 tasks. The final output needs to report the solution for the initial framework and as many outputs as the number of changes. The four new tracks involve the following semantics and problems.

**Remark.** Tasks in dynamic tracks are invoked by appending "$D$" at the end of the intended task: for instance, EE-$\sigma$-D points to the enumeration task with semantics $\sigma$, having as input a dynamic AF.

## 5 Input File Formats

Each benchmark instance is provided in two different file formats: trivial graph format (`tgf`) and aspartix format (`apx`). In the following we present the AF $\mathcal{F} = (\mathsf{A}, \rightarrow)$ where $\mathsf{A} = \{a1, a2, a3\}$ and $\rightarrow = \{(a1, a2), (a2, a3), (a2, a1)\}$ in each of these formats.

### 5.1 Trivial Graph Format

See `http://en.wikipedia.org/wiki/Trivial_Graph_Format`. In the following we will refer to the file containing this instance by `myFile.tgf`.

```
1
2
3
#
1 2
2 3
2 1
```

### 5.2 Aspartix Format

See (Egly *et al.*, 2010). In the following we will refer to the file containing this instance by `myFile.apx`.

```
arg(a1).
arg(a2).
arg(a3).
att(a1,a2).
att(a2,a3).
att(a2,a1).
```

### 5.3 Dynamic Track: Format of the File with Changes

For each (dynamic) problem instance, a solver requires to take as input two files: the starting framework (either in `apx` or `tgf` format) and a text file with a list of changes to be applied to it. The file with changes has to report a list of modifications (one per line) over the initial framework. The format of the file with changes has to follow the same format of the original file (either in `apxm` or `tgfm` format, see in the following of this section). Let us introduce an example in `apx` (Section 5.2).

**Example 5.1.** *The starting framework is provided in full. For instance, it can be `myFile.apx`:*

```
arg(a1).
arg(a2).
arg(a3).
att(a1,a2).
att(a2,a3).
```

*The second file is a text file containing the list of modifications to be* sequentially *performed on the starting file, one after another. The name of this file has to be the same as the starting framework, with extension* .apxm *instead of* .apx. *For this example, `myFile.apxm` is:*

```
+att(a3,a2).
-att(a1,a2).
+att(a1,a3).
```

*Applying these changes over the initial file corresponds in practice to three more full frameworks (besides the initial one), where the first one is:*

```
arg(a1).
arg(a2).
```

```
arg(a3).
att(a1,a2).
att(a2,a3).
att(a3,a2).
```

*The second framework is:*

```
arg(a1).
arg(a2).
arg(a3).
att(a2,a3).
att(a3,a2).
```

*The third framework is:*

```
arg(a1).
arg(a2).
arg(a3).
att(a2,a3).
att(a3,a2).
att(a1,a3).
```

We propose a second example by using the same exact framework expressed in the `tgf` format (see Section 5.1). The initial framework is here called `myFile.tgf`:

**Example 5.2.** `1`

```
2
3
#
1 2
2 3
```

*The text file with modification needs to have the same name of the the initial framework, with suffix* .tgfm *instead of* .tgf. *Hence, in this case, `myFile.tgfm`:*

```
+3 2
-1 2
+1 3
```

*Even in this case, applying these changes over the initial file corresponds in practice to three more full frameworks (besides the initial one), where the first one is:*

```
1
2
3
#
1 2
2 3
3 2
```

*The second framework is:*

```
1
2
3
#
2 3
3 2
```

*The third framework is:*

```
1
2
3
#
2 3
3 2
1 3
```

**Remark 1.** During these modifications, attacks have to be added only between existing arguments: no new argument can be introduced. In case all the attacks connected to an argument are removed, such argument is *not* to be removed from the framework.

**Remark 2.** Benchmarks and generators need to provide both the initial framework and the modification file for each instance. For each initial framework, a modification file with at least 15 attack additions/deletions is required to be submitted/generated.

If a modification file has $n$ changes (one per text line), a solver has to run $n$ different problems by applying such modifications in sequence (from the top of the file). Please do not consider $n$ as fixed, but just continue solving problems until there are no more modifications (i.e., lines) to apply.

# 6 Output Format

In the following two subsections we describe the format that the output need to follow, for both classical and dynamic tracks.

## 6.1 Classical Tracks

Solvers must write the result to standard output exactly in the format described below.

- **DC-$\sigma$** for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{GR}, \mathbf{ID}\}$:

  The output must be either

  ```
  YES
  ```

  expressing that the queried argument is credulously accepted in the given AF under $\sigma$, or

  ```
  NO
  ```

  if the queried argument is not credulously accepted in the given AF under $\sigma$.

- **DS**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}\}$:
  The output must be either

  ```
  YES
  ```

  expressing that the queried argument is skeptically accepted in the given AF under $\sigma$, or

  ```
  NO
  ```

  if the queried argument is not skeptically accepted in the given AF under $\sigma$.

- **SE**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{GR}, \mathbf{ID}\}$:
  The output must be of the form

  ```
  [a1,a3,a6]
  ```

  expressing that $\{a1, a3, a6\}$ is a $\sigma$-extension of the given AF, or

  ```
  NO
  ```

  expressing that there does not exist a $\sigma$-extension of the given AF.

- **EE**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}\}$:
  The output must be of the form

  ```
  [
    [a1,a2]
    [a1,a3]
    [a2,a3]
  ]
  ```

  expressing that the set of $\sigma$-extensions of the given AF is exactly $\{\{a1, a2\}, \{a1, a3\}, \{a2, a3\}\}$, or

  ```
  []
  ```

  expressing that there does not exist a $\sigma$-extension of the given AF. Note that if the empty set is the only $\sigma$-extension of the given AF, the output is required to be:

  ```
  [
    []
  ]
  ```

  White spaces (but not new line carriage (\n)) are ignored.

### 6.2 Dynamic Tracks

Concerning dynamic tracks, all the answers are in the form of a list where the first element represents the solution of the required task on the initial framework; each following element in this list is the answer returned on the $(i+1)^{th}$ framework obtained by sequentially applying the first $i$ changes in the modification file (see Section 5.3): $i \in [1..n]$ and $n$ is the total number of changes in the modification file.

- **DC**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{GR}\}$:

  The output must be a list of YES or NO. For instance, considering the files in Example 5.1 and the DC-CO task checking argument `a1`, the answer needs to be:

  ```
  [YES, YES, YES, YES]
  ```

- **DS**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$:

  The output must be a list of YES or NO. For instance, considering the files in Example 5.1 and the DC-CO task checking argument `a1`, the answer needs to be:

  ```
  [YES, YES, YES, YES]
  ```

- **SE**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{GR}\}$: The output must be a list of extensions. For instance, considering the files in Example 5.1 and the SE-CO task, the answer needs to be:

  ```
  [
    [a1,a3]
    [a1,a3]
    [a1]
    [a1,a2]
  ]
  ```

  An element in the list can be `[]` if there does not exist a $\sigma$-extension in the corresponding AF.

- **EE**-$\sigma$ for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$:

  The output must be a list of lists of extensions (i.e., the enumeration for each of the considered frameworks). For instance, considering the files in Example 5.1 and the EE-CO task, the answer needs to be:

  ```
  [
    [
      [a1,a3]
    ]
    [
      [a1,a3]
    ]
    [
      [a1]
      [a1,a3]
  ```

```
    [a1,a2]
  ]
  [
    [a1,a2]
  ]
]
```

Note that if the empty set is the only $\sigma$-extension of one of the given AF, the output is required to be

```
[
  [
  ]
]
```

for that framework.

# 7 Solver Interface

The single executable of a solver should be runnable from a command line and must provide the following behavior (let `solver` be the filename of the executable. The following example commands show the case a solver already implements the required interface[2]. Your solver must be executed by `docker run ...` command line, for docker commands please refer to the link in Section 2):

- `solver` (without any parameters)
  Prints author and version information of the solver on standard output.
  **Example:**

  ```
  user$ solver
  MySolver v1.0
  John Smith
  user$ _
  ```

- `solver --formats`
  Prints the supported formats of the solver in the form

  ```
  [supportedFormat1,supportedFormat2, ..., supportedFormatN]
  ```

  The possible values for each supported format are `tgf`, `apx`.
  **Example:**

  ```
  user$ solver --formats
  [tgf,apx]
  user$ _
  ```

- `solver --problems`
  Prints the supported computational problems (i.e., tasks) in the form

[2] In case your solver does not abide by the syntax, you can use the script *generic-interface-2019.sh* file in *conarg_dir.zip* as an interface to the solver.

`[supportedProblem1,supportedProblem2, ..., supportedProblemN]`

The possible values are `DC-CO`, `DC-PR`, `DC-ST`, `DC-SST`, `DC-STG`, `DC-GR`, `DC-ID`, `DS-CO`, `DS-PR`, `DS-ST`, `DS-SST`, `DS-STG`, `SE-CO`, `SE-PR`, `SE-ST`, `SE-SST`, `SE-STG`, `SE-GR`, `SE-ID`, `EE-CO`, `EE-PR`, `EE-ST`, `EE-SST`, `EE-STG`, and values for dynamic tracks are `DC-CO-D`, `DS-CO-D`, `SE-CO-D`, `EE-CO-D`, `DC-PR-D`, `DS-PR-D`, `SE-PR-D`, `EE-PR-D`, `DC-ST-D`, `DS-ST-D`, `SE-ST-D`, `EE-ST-D`, `DC-GR-D`, `SE-GR-D`.

**Example:**

```
user$ solver --problems
[DC-CO,DS-CO,EE-CO,SE-ST]
user$ _
```

- `solver -p <task> -f <file> -fo <fileformat> [-a <additional_parameter>]`
  Solves the given classical problem on the argumentation framework specified by the given file (represented in the given file format) and prints out the result. More specifically:
  - `solver -p DC-<semantics> -f <file> -fo <fileformat> -a <additional_parameter>`
    Solves the problem of deciding whether an argument (given as additional parameter) is credulously inferred and prints out either `YES` (if it is credulously inferred) or `NO` (if it is not credulously inferred).
    **Example:**

    ```
    user$ solver -p DC-CO -f myFile.apx -fo apx -a a1
    YES
    user$ _
    ```

  - `solver -p DS-<semantics> -f <file> -fo <fileformat> -a <additional_parameter>`
    Solves the problem of deciding whether an argument (given as additional parameter) is skeptically inferred and prints out either `YES` (if it is credulously inferred) or `NO` (if it is not credulously inferred).
    **Example:**

    ```
    user$ solver -p DC-ST -f myFile.apx -fo apx -a a2
    NO
    user$ _
    ```

  - `solver -p SE-<semantics> -f <file> -fo <fileformat>`
    Returns one extension wrt. the given semantics in the format `[A1,A2,...,AN]` (no further parameters needed).
    **Example:**

    ```
    user$ solver -p SE-PR -f myFile.tgf -fo tgf
    [a1,a3]
    user$ _
    ```

    If there does not exist any extension wrt. the given semantics (which can be the case for stable semantics) then the output `NO` is expected by the solver.
    **Example:**

    ```
    user$ solver -p SE-ST -f anotherFile.tgf -fo tgf
    NO
    user$ _
    ```

- `solver -p EE-<semantics> -f <file> -fo <fileformat>`
  Enumerates all sets that are extensions wrt. the given semantics in the format `[[A1,A2,...,AN],[B1,B2,...,BM],..., [Z1,Z2,...,ZN]]` (no further parameters needed).
  **Example:**

```
user$ solver -p EE-PR -f myFile.apx -fo apx
[
[a1,a3]
[a2]
]
user$ _
```

- `solver -p <task> -f <file> -m <modification file> -fo <fileformat> [-a <additional_parameter>]`
  Solves the given dynamic problem on the argumentation framework specified by the given file (represented in the given file format) and the file with the list of sequential modifications; finally, it prints out the result. More specifically:
  - `solver -p DC-<semantics>-D -f <file> -m <modification file> -fo <fileformat> -a <additional_parameter>`
    Solves the problem of deciding whether an argument (given as additional parameter) is credulously inferred and prints out either `YES` (if it is credulously inferred) or `NO` (if it is not credulously inferred) for the initial framework and each of the frameworks obtained by sequentially applying modifications.
    **Example:**

```
user$ solver -p DC-CO-D -f myFile.apx -m myFile_Modapx.apxm
-fo apx -a a1
[YES, YES, NO, YES]
user$ _
```

  - `solver -p EE-<semantics>-D -f <file> -m <modification file> -fo <fileformat>`
    Enumerates all the sets that are extensions according to the given semantics and prints out the result following the output format in Section 6.2.
    **Example:**

```
user$ solver -p EE-PR-D -f myFile.apx -m myFile_Modapx.apxm -fo
apx
[
[
[a1,a3]
[a2]
]
[
[a1]
[a3]
]
]
user$ _
```

The formats for the modification files is given in Section 5.3.

Each solver has to support at least one file format. It is ensured that each solver is only called with file formats and problems he supports (the behavior of a solver when called with unsupported parameters is undefined).

## References


G. Alfano, S. Greco, and F. Parisi. Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 49–55. ijcai.org, 2017.

P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

P. Baroni, M. Giacomin, and B. Liao. On topology-related properties of abstract argumentation semantics. A correction and extension to dynamics of argumentation systems: A division-based method. *Artif. Intell.*, 212:104–115, 2014.

R. Baumann and G. Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In *Computational Models of Argument (COMMA)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 75–86. IOS Press, 2010.

S. Bistarelli, F. Santini, and C. Taticchi. On looking for invariant operators in argumentation semantics. In *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS*, pages 537–540. AAAI Press, 2018.

G. Boella, S. Kaci, and L. W. N. van der Torre. Dynamics in argumentation with single extensions: Abstraction principles and the grounded extension. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty ECSQARU*, volume 5590 of *LNCS*, pages 107–118. Springer, 2009.

Martin Caminada, Walter Alexandre Carnielli, and Paul E. Dunne. Semi-stable semantics. *J. Log. Comput.*, 22(5):1207–1254, 2012.

C. Cayrol, F. de Saint-Cyr, and M.-C. Lagasquie-Schiex. Change in abstract argumentation frameworks: Adding an argument. *J. Artif. Intell. Res.*, 38:49–84, 2010.

Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(10–15):642–674, 2007.

Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.

N. Kökciyan, N. Yaglikci, and P. Yolum. Argumentation for resolving privacy disputes in online social networks: (extended abstract). In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1361–1362. ACM, 2016.

Bart Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In *Proceedings of the 8th Dutch Conference on Artificial Intelligence (NAIC'96)*, pages 357–368, 1996.

Y. Xu and C. Cayrol. The matrix approach for abstract argumentation frameworks. In *Theory and Applications of Formal Argumentation TAFA*, volume 9524 of *LNCS*, pages 243–259. Springer, 2015.