The Fourth International Competition on Computational Models of Argumentation*

Solver Requirements

Jean-Marie Lagniez[1], Emmanuel Lonca[1], Jean-Guy Mailly[2], Julien Rossit[2]
[1]: CRIL, Université d'Artois & CNRS, France, {lagniez,lonca}@cril.fr
[2]: LIPADE, Université de Paris, France, {jean-guy.mailly, julien.rossit}@u-paris.fr

December 3, 2020

Abstract

This document contains requirements for solvers participating at ICCMA'21 [1]. In particular, this document provides some formal background on abstract and structured (assumption-based) argumentation, a list of computational problems 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.

1 Abstract Argumentation

An abstract argumentation framework (AF) [2] is a directed graph $F = \langle A, R\rangle$, where A is the set of arguments, and $R \subseteq A \times A$ is the attack relation. For $a, b, c \in A$, we say that a *attacks* b if $(a, b) \in R$. If in turn b attacks c, then a *defends* c against b. Similarly, a set $S \subseteq A$ attacks (respectively defends) an argument b if there is some $a \in S$ that attacks (respectively defends) b. For $S \subseteq A$ a set of arguments, S^+ is the set of arguments that are attacked by S, formally $S^+ = \{b \in A \mid \exists a \in S \text{ s.t. } (a, b) \in R\}$. The range of S is $S^\oplus = S \cup S^+$.

Different semantics have been defined for evaluating the acceptability of (sets of) arguments.

Definition 1. *Given an AF $F = \langle A, R\rangle$, a set of arguments $S \subseteq A$ is conflict-free iff $\forall a, b \in S$, $(a, b) \notin R$. A conflict-free set S is admissible iff $\forall a \in S$, S defends a against all its attackers. Conflict-free and admissible sets are respectively denoted by $\mathbf{CF}(F)$ and $\mathbf{ADM}(F)$.*

Now, we formally introduce the extension-based semantics. For $S \subseteq A$,

- *$S \in \mathbf{CO}(F)$ iff $S \in \mathbf{ADM}(F)$ and $\forall a \in A$ that is defended by S, $a \in S$;*
- *$S \in \mathbf{PR}(F)$ iff S is a $\subseteq$-maximal admissible set;*
- *$S \in \mathbf{ST}(F)$ iff S is a conflict-free set that attacks each $a \in A \setminus S$;*
- *$S \in \mathbf{SST}(F)$ iff $S \in \mathbf{CO}(F)$ and there is no $S_2 \in \mathbf{CO}(F)$ s.t. $S^\oplus \subset S_2^\oplus$;*
- *$S \in \mathbf{STG}(F)$ iff $S \in \mathbf{CF}(F)$ and there is no $S_2 \in \mathbf{CF}(F)$ s.t. $S^\oplus \subset S_2^\oplus$;*
- *$S \in \mathbf{ID}(F)$ iff $S \in \mathbf{ADM}(F)$, $S \subseteq \cap\mathbf{PR}(F)$, and there is no $S_2 \subseteq \cap\mathbf{PR}(F)$ such that $S_2 \in \mathbf{ADM}(F)$ and $S \subset S_2$.*

CO, **PR**, **ST**, **SST**, **STG** and **ID** stand (respectively) for the complete, preferred, stable [2], semi-stable [3], stage [4] and ideal [5] semantics. We refer the interested reader to [6] for more details about these semantics.

For $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{ID}\}$ a semantics, an argument $a \in A$ is credulously (respectively skeptically) accepted in $F = \langle A, R\rangle$ with respect to σ iff $a \in S$ for some (respectively each) $S \in \sigma(F)$.

Let us recall that, for any AF F, $|\mathbf{ID}(F)| = 1$, which means that an argument is credulously accepted if and only if it is skeptically accepted. For $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{SST}, \mathbf{STG}\}$, $|\sigma(F)| \geq 1$ for any AF F. Only in the case of the stable semantics, there are AFs F such that $\mathbf{ST}(F) = \emptyset$. In this situation, every argument is skeptically accepted (but none is credulously accepted).

*Adapted from last edition's version by Stefano Bistarelli, Lars Kotthoff, Theofrastos Mantadelis, Francesco Santini and Carlo Taticchi: `https://www.iccma2019.dmi.unipg.it/res/SolverRequirements.pdf`

2 Assumption-based Argumentation

Now, let us introduce a particular framework for structured argumentation, namely Assumption-based Argumentation (ABA) [7]. ABA is one of the most popular structured argumentation frameworks, with applications in various domains, like *e.g.* information seeking and inquiry dialogues [8], decision making in a medical context [9], or explanation of automated decisions [10]. An ABA framework is a tuple $F = \langle L, R, A, \overline{\ } \rangle$, where:

- L is a set of symbols called the language;
- R is a set of rules of the form $x_0 \leftarrow x_1, \ldots, x_n$, with $x_i \in L$ for $i \in \{0, \ldots, n\}$ and $n \geq 0$;
- $A \subseteq L$ is a non-empty set of particular symbols called assumptions;
- $\overline{\ } : A \to L$ is a total mapping that expresses a notion of contrariness.

In a rule $x_0 \leftarrow x_1, \ldots, x_n$, the left-hand part ($x_0$) is called the head of the rule, while the right-hand part ($x_1, \ldots, x_n$) is called the body. A rule $x_0 \leftarrow$ with no body can be interpreted as $x_0 \leftarrow \top$.

A deduction for $x \in L$ supported by $X_L \subseteq L$ and $X_R \subseteq R$ is a (finite) tree rooted in x, with nodes labeled by symbols in L or $\top$, such that each leaf is either a symbol in X_L or $\top$, and for each internal node with label x', its children are the elements $x_1, \ldots, x_n$ of the body of some rule $x' \leftarrow x_1, \ldots, x_n \in X_R$. An argument for the claim $x \in L$ supported by $X_A \subseteq A$ (denoted by $X_A \vdash x$) is a deduction for x supported by X_A (and some $X_R \subseteq R$). An argument $A_1 \vdash x_1$ attacks an argument $A_2 \vdash x_2$ iff x_1 is the contrary of some assumption in A_2.

Finally, we introduce the notion of *flat* ABA framework. This is a particular subclass of ABA frameworks $F = \langle L, R, A, \overline{\ } \rangle$, such that there is no rule $x_0 \leftarrow x_1, \ldots, x_n \in R$ with $x_0 \in A$, *i.e.* there is no assumption as the head of a rule.

Now, there are two equivalent ways of reasoning with an ABA framework: either the status of arguments is evaluated, or the status of assumptions. Here, we choose to consider the latter.

Definition 2. *Given $F = \langle L, R, A, \overline{\ } \rangle$ a flat ABA framework, a set of assumptions $A_1 \subseteq A$ attacks a set of assumptions $A_2 \subseteq A$ iff an argument supported by a subset of A_1 attacks an argument supported by a subset of A_2. A set of assumptions defends an assumption a if it attacks each set of assumptions that attacks a. Then, given a set of assumptions $X_A \subseteq A$,*

- $X_A \in \mathbf{CF}(F)$ *iff it does not attack itself;*
- $X_A \in \mathbf{ADM}(F)$ *iff* $X_A \in \mathbf{CF}(F)$ *and X_A defends all its elements;*
- $X_A \in \mathbf{CO}(F)$ *iff* $X_A \in \mathbf{ADM}(F)$ *and* $\forall a \in A$ *that is defended by X_A, $a \in X_A$;*
- $X_A \in \mathbf{PR}(F)$ *iff X_A is a $\subseteq$-maximal admissible set;*
- $S \in \mathbf{ST}(F)$ *iff* $X_A \in \mathbf{CF}(F)$ *and X_A attacks each $a \in A \setminus X_A$.*

3 Computational Problems

In the following Sections 3.1, 3.2 and 3.3, we describe (sub-)tracks where the solvers should provide exact result in all cases. On the contrary, Section 3.4 describes the introduction of the first track for approximate algorithms at ICCMA.

3.1 Static Abstract Argumentation

Let $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}\}$ be a semantics. For each one, we define a sub-track made of four computational problems:

- **CE**-σ: Given an AF $F = \langle A, R \rangle$, give the number of σ-extensions of F.
- **SE**-σ: Given an AF $F = \langle A, R \rangle$, give one σ-extension of F.
- **DC**-σ: Given an AF $F = \langle A, R \rangle$ and $a \in A$ an argument, is a credulously accepted in F?
- **DS**-σ: Given an AF $F = \langle A, R \rangle$ and $a \in A$ an argument, is a skeptically accepted in F?

These problems can be understood as: **SE**: "give **s**ome **e**xtension"; **CE**: "**c**ount the **e**xtensions"; **DC**: "**d**ecide **c**redulous acceptance"; **DS**: "**d**ecide **s**keptical acceptance".

The track also includes a sixth sub-track for the ideal semantics. We only consider **SE-ID** and **DS-ID**, since **CE-ID** is trivial (the answer is always 1), and **DC-ID** coincides with **DS-ID**.

3.2 Dynamic Abstract Argumentation

Similarly to the previous edition of the competition, ICCMA'21 includes a track dedicated to argumentation dynamics, *i.e.* we expect solvers that are able to solve, sequentially, a given task each time an AF is updated. However, we propose some changes to the track, that are described here. The first one is the reduction of the set of semantics, since (similarly to classical track), we do not consider the grounded semantics, that was part of the previous editions. So, in this track we have three sub-tracks, corresponding to $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$. For each of these sub-tracks, the reasoning tasks are, as previously, **CE**, **SE**, **DC**and **DS**. Now, we describe the two main differences:

- the possible kinds of updates were only only the addition/deletion of a single attack: we also consider the addition of an argument (with a set of incident attacks) and the deletion of an argument (as well as all the incident attacks);
- the solvers were reading the full set of updates in a text file, before any computation: now, the solvers will wait for the updates, that will be provided on the standard input.

"Complex" additions and deletions can be decomposed into simpler updates (for instance, adding an argument with incident attacks can be simulated with the addition of a disconnected argument, and then the sequential addition of the attacks), so any solvers that participated to the previous edition can participate again, with minor updates of the algorithms. However, we hope that the modification of the track will lead to the development of efficient algorithms for such complex updates, that are more relevant for applications of argumentation. Similarly, it seems more realistic *e.g.* in a debate that an agent learns the updates one at a time, rather than all of them at once before any calculation. This is why we replace the description of the updates in a text file by an online process. We give more details about the input of dynamic solvers in Section 4.4.

3.3 Assumption-based Argumentation

For the first time at ICCMA, we propose a track dedicated to structured argumentation. The track is divided into three sub-tracks, corresponding to the complete, preferred and stable semantics of flat ABA frameworks. Similarly to abstract argumentation, we consider the four computational problems introduced previously, *i.e.* for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$, the problems are **CE**-σ, **SE**-σ, **DC**-σ and **DS**-σ, corresponding respectively to counting the number of extensions, computing one extension, and determining the credulous or skeptical acceptance of an assumption.

3.4 Approximate Algorithms

Finally, for the first time at ICCMA, we separate exact algorithms from approximate algorithms. In each of the track described previously (Sections 3.1, 3.2 and 3.3), we expect exact algorithms, *i.e.* algorithms that always provide a correct answer (or do not provide an answer at all if the time required for computing the solution is longer than the runtime limit). This means that any mistake leads to the exclusion of a solver from the sub-track where it is not correct.

On the contrary, we will authorize the participation of approximate solvers in a dedicated track. The main interest of an approximate algorithm is the rapidity of the answer. Thus the approximate solvers will be given less time than exact algorithms. Also, wrong answers will not lead to the exclusion of the solver, but will only cause a decrease of the solver score. Since the issue of evaluating approximate algorithms is new to ICCMA, we have chosen to restrict this track to only two problems for each sub-track: **DS**-σ and **DC**-σ, where $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}\}$, and one problem **DS**-σ for $\sigma = \mathbf{ID}$.

4 Input File Formats

Each benchmark for abstract argumentation is provided in two different file formats: trivial graph format (`tgf`) and ASPARTIX format (`apx`), described respectively in Sections 4.1 and 4.2. For the ABA frameworks benchmarks, we propose the ABASPARTIX format (`abapx`), inspired by the ASPARTIX format, described in Section 4.3.

For the following examples, we use the AF $F = \langle A, R \rangle$ with $A = \{a_1, a_2, a_3\}$ and $R = \{(a_1, a_2), (a_2, a_3), (a_2, a_1)\}$, depicted at Figure 1.

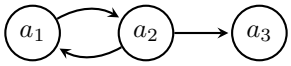


Figure 1: The AF F

4.1 Trivial Graph Format

The Trivial Graph Format describes a graph by giving a list of identified for nodes, then a list of edges, separated by a # symbol. See `https://en.wikipedia.org/wiki/Trivial_Graph_Format` for more details. We give below the content a `myFile.tgf`, that corresponds to the AF depicted at Figure 1.

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

4.2 ASPARTIX Format

The ASPARTIX format (named after the ASP-based argumentation solver ASPARTIX [11]) describes the argument names as rules `arg(name).`, and attacks as rules `att(name1,name2).` We give below, as an example, the content of `myFile.apx`, that corresponds to the AF depicted at Figure 1.

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

4.3 ABASPARTIX Format

Inspired by the ASPARTIX format for abstract AFs, we propose the ABASPARTIX format for ABA frameworks. For instance, we consider the ABA framework $F = \langle L, R, A, \overline{\ } \rangle$ with $L = \{a, b, c, p, q, r, s, t\}$, $R = \{(p \leftarrow q, a), (q \leftarrow), (r \leftarrow b, c)\}$, $A = \{a, b, c\}$ and $\overline{a} = r$, $\overline{b} = s$, $\overline{c} = t$. This example, borrowed from [7], can be represented as the following `myFile.abapx` file:

```
rule(p,q,a).
rule(q).
rule(r,b,c).
assum(a).
assum(b).
assum(c).
cont(a,r).
cont(b,s).
cont(c,t).
```

For each line `rule(...).` in the file, the first parameter corresponds to the head of the rule, and the other (optional) parameters correspond to the body. For instance, `rule(p,q,a).` represents the rule $p \leftarrow q, a$, and `rule(q).` corresponds to $q \leftarrow$. Assumptions and the contrariness mapping are represented, respectively, by `assum(...).` and `cont(...).` lines. Finally, the language does not need to be explicitly given, since it is simply the set of all the symbols that appear in the rules and assumptions.

4.4 Dynamic Track

For the dynamic track, the AF can be given as a `.tgf` or `.apx` file. The associated set of updates must then be given as a `.tgfm` or `.apxm` file. These formats have been defined at ICCMA'19 for updates that are only additions and deletions of attacks. Since we introduce new kinds of updates (addition of an argument with a set of attacks, and deletion of an argument with all the incident attacks), we extend the formats accordingly.

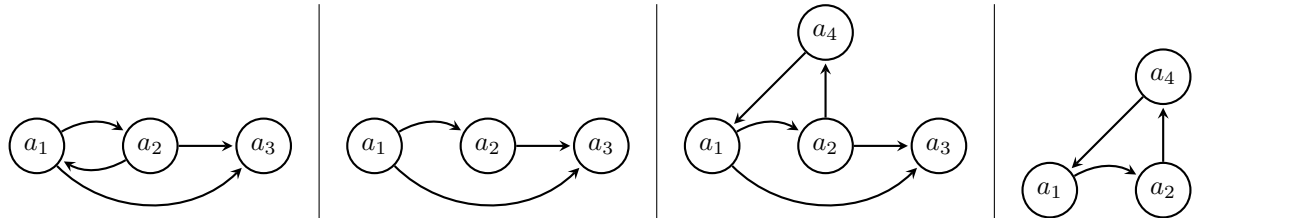
A `.tgfm` file is a succession of lines, each corresponding to one update:

- `+1 3` means that an attack from argument 1 to argument 3 must be added;
- `-2 1` means that the attack from argument 2 to argument 1 must be deleted;
- `+4:4 1:2 4` means that an argument 4 must be added, as well as the attacks from 4 to 1, and from 2 to 4;
- `-3` means that the argument 3 must be removed, as well as all the attacks where this argument appears.

Similarly, we extend the `.apxm` format. For instance:

- `+att(a1,a3).` for adding an attack from a_1 to a_3
- `-att(a2,a1).` for deleting an attack from a_2 to a_1
- `+arg(a4):att(a4,a1):att(a2,a4).` for adding an argument a_4 and attacks from a_4 to a_1 and from a_2 to a_4.
- `-arg(a3).` for removing argument a_3 (as well as all the indident attacks).

A `.tgfm` file or an `.apxm` containing these lines correspond to updates of the AF F (Figure 1) as follows:



(a) Adding an attack (a_1, a_3) (b) Deleting an attack (a_2, a_1) (c) Adding an argument a_4 (d) Deleting the argument a_3

Figure 2: Successive Updates of the AF F

Let us mention that the `.tgfm` or `.apxm` file will not be provided to the solvers. Indeed, this year, the updates will be provided one by one, on the solver standard input. This means, for instance, that the solver will first solve the problem for the AF F from Figure 1, and print the answer on the standard output. After this first computation, an update will be provided on the standard input, and the solver will have to print on the standard input the result for the AF given at Figure 2a. Only after that, the second update will be provided, and so on. After the last update, we will provide on the standard input an empty line, that indicates the end of the computations.

The solver wrapper we will use to implement this behavior is freely available at `https://github.com/crillab/iccma-dynamics-wrapper`. Please contact us if you experiment some issues with this software, and do not hesitate to "watch" it on github to be informed when updates are performed.

5 Output Format

The following subsections describe the format that the output need to follow, for both static tracks (*i.e.* static abstract argumentation, assumption-based argumentation, and approximate algorithms) in Section 5.1, and the dynamic track in Section 5.2.

5.1 Static Tracks

For all the tracks except the dynamic argumentation track, solvers must write the result to standard output exactly in the format described below.

- **DC**-σ and **DS**-σ, for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}, \mathbf{SST}, \mathbf{STG}, \mathbf{ID}\}$. The output must be either

  `YES`

  if the queried argument is (respectively) credulously or skeptically accepted in the given AF under σ, or if the queried assumption is (respectively) credulously or skeptically accepted in the given ABA framework under σ, or

  `NO`

  otherwise.

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

  `[a1,a2,a3]`

  meaning that $\{a_1, a_2, a_3\}$ is a σ-extension of the given AF or ABA framework. a_1, a_2 and a_3 are arguments in the former case, and assumptions in the latter case. If $\sigma = \mathbf{ST}$, there may be benchmarks that do not possess any extension. In that case, the output must be

  `NO`

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

  `k`

  where $k \in \mathbb{N}$ is the number of σ-extensions of the given AF or ABA framework.

5.2 Dynamic Track

For the dynamic track, we choose to simplify the output compared to the previous edition of the competition. The solvers must output the answer for the given problem on the standard output, then wait for the next update (provided on the standard input). Concretely, the first line on the standard output represents the answer for the initial AF, and then, for $i > 0$, the $(i+1)^{th}$ line corresponds to the answer for the AF obtained after sequentially applyling the first i updates. Each answer must follow the rules given in Section 5.1.

- **DS**-σ and **DC**-σ, for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$, the solver output must be a succession of lines containing either `YES` or `NO`.
- **CE**-σ, for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$, the solver output must be a succession of lines containing a number.
- **SE**-σ, for $\sigma \in \{\mathbf{CO}, \mathbf{PR}, \mathbf{ST}\}$, the solver output must be a succession of lines containing an extension.

6 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).

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

  ```
  user$ solver
  MySolver v1.0
  John Doe, john.doe@example.com
  user$ _
  ```

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

  `[supportedFormat1,...,supportedFormatN]`

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

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

- `solver --problems`
  Prints the supported computational problems in the form

  `[supportedProblem1,...,supportedProblemN]`

  The possible values are **DS-CO**, **DS-PR**, **DS-ST**, **DS-SST**, **DS-STG**, **DS-ID**, **DC-CO**, **DC-PR**, **DC-ST**, **DC-SST**, **DC-STG**, **SE-CO**, **SE-PR**, **SE-ST**, **SE-SST**, **SE-STG**, **SE-ID**, **CE-CO**, **CE-PR**, **CE-ST**, **CE-SST**, **CE-STG**, as well as their variants for the dynamic track **DS-CO-D**, **DS-PR-D**, **DS-ST-D**, **DC-CO-D**, **DC-PR-D**, **DC-ST-D**, **SE-CO-D**, **SE-PR-D**, **SE-ST-D**, **CE-CO-D**, **CE-PR-D**, **CE-ST-D**. Example:

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

- `solver -p <task> -f <file> -fo <fileformat> [-a <additional_parameter>]`
  Solves the given problem on the AF or ABA framework specified by the given file, represented in the given file format, and prints out the result. More specifically, the task can be any of the supported problems described previously, and the file format can be any of the supported format. If the file format is `tgf` or `apx`, then the file must describe an AF; if it is `abapx`, then the file must describe an ABA framework. Finally, the additional parameter is required for **DC** and **DS** problems: it represents the argument or assumption that is queried for acceptance. Here are some examples:

  - `solver -p DC-<semantics> -f <file> -fo <fileformat> -a <additional_parameter>`
    Solves credulous decision problem for the given AF or ABA framework, with respect to the argument or assumption given as additional parameter. 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 skeptical decision problem for the given AF or ABA framework, with respect to the argument or assumption given as additional parameter. Example:

    ```
    user$ solver -p DS-ST -f myFile.tgf -fo tfg -a a2
    NO
    user$ _
    ```

  - `solver -p CE-<semantics> -f <file> -fo <fileformat>`
    Solves counting problem for the given AF or ABA framework. Example:

    ```
    user$ solver -p CE-PR -f myFile.abapx -fo abapx
    4
    user$ _
    ```

  - `solver -p SE-<semantics> -f <file> -fo <fileformat>`
    Returns one extension of the given AF or ABA framework. Example:

```
user$ solver -p SE-ID -f myFile.tgf -fo tfg
[a1,a3,a4]
user$ _
```

Contrary to the previous edition, the dynamic version of the problems does not require an additional parameter to indicate the modification file. Indeed, the modifications will not be given together, in a text file, before any computation, but one at a time on the standard input. In the following examples, we suppose that there are three updates. This means that there must be four lines printed on the standard output: one for the initial AF, and one after each update. An empty line is sent on the solver standard input to indicate the end of the updates.

- `solver -p DC-<semantics>-D -f <file> -fo <fileformat> -a <additional_parameter>`
  Solves dynamically the credulous decision problem for the given AF, with respect to the argument additional parameter. Example:

  ```
  user$ solver -p DC-CO-D -f myFile.apx -fo apx -a a1
  YES
  NO
  NO
  YES
  user$ _
  ```

- `solver -p DS-<semantics>-D -f <file> -fo <fileformat> -a <additional_parameter>`
  Solves dynamically the skeptical decision problem for the given AF, with respect to the argument given as additional parameter. Example:

  ```
  user$ solver -p DS-ST-D -f myFile.tgf -fo tfg -a a2
  NO
  NO
  YES
  YES
  user$ _
  ```

- `solver -p CE-<semantics>-D -f <file> -fo <fileformat>`
  Solves dynamically the counting problem for the given AF. Example:

  ```
  user$ solver -p CE-PR-D -f myFile.abapx -fo abapx
  4
  6
  2
  7
  user$ _
  ```

- `solver -p SE-<semantics>-D -f <file> -fo <fileformat>`
  Returns dynamically one extension of the given AF. Example:

  ```
  user$ solver -p SE-CO-D -f myFile.tgf -fo tfg
  [a1,a3,a4]
  [a1,a2,a4]
  [a1,a3]
  [a3,a4]
  user$ _
  ```

Each solver has to support at least one file format, and one problem. It is ensure that each solver is only called with file formats and problems it supports. The behavior of a solver when called with unsupported parameters is undefined.

7 Submission format

Solvers must be submitted as a ZIP archive containing their source code and a `build.sh` shell script producing the solver binary from the source code (if needed). A `README` file must be present, explaining the material required

by the solver (e.g. libraries), how to launch it against a problem, and other information that may be useful to build/execute your solver.

The experiments are intended to be held on machines equipped with Intel Xeon E5-2637 v4 CPUs and 128GB of RAM. These machines are operated by 64bits CentOS Linux release 7.3.1611 (Linux kernel version 3.10).

References

[1] Lagniez JM, Lonca E, Mailly JG, Rossit J. Introducing the Fourth International Competition on Computational Models of Argumentation. In: Proc. of SAFA'20;. p. 80–85.

[2] Dung PM. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artif Intell. 1995;77(2):321–358.

[3] Caminada M, Carnielli WA, Dunne PE. Semi-stable semantics. J Log Comput. 2012;22(5):1207–1254.

[4] Verheij B. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In: Proc.of NAIC'96; 1996. p. 357–368.

[5] Dung PM, Mancarella P, Toni F. Computing ideal sceptical argumentation. Artif Intell. 2007;171(10-15):642–674.

[6] Baroni P, Caminada M, Giacomin M. Abstract Argumentation Frameworks and Their Semantics. In: Baroni P, Gabbay D, Giacomin M, van der Torre L, editors. Handbook of Formal Argumentation. College Publications; 2018. p. 159–236.

[7] Toni F. A tutorial on assumption-based argumentation. Argument & Computation. 2014;5(1):89–117.

[8] Fan X, Toni F. Agent Strategies for ABA-based Information-seeking and Inquiry Dialogues. In: Proc. of ECAI'12; 2012. p. 324–329.

[9] Fan X, Craven R, Singer R, Toni F, Williams M. Assumption-Based Argumentation for Decision-Making with Preferences: A Medical Case Study. In: Proc. of CLIMA XIV; 2013. p. 374–390.

[10] Zhong Q, Fan X, Toni F, Luo X. Explaining Best Decisions via Argumentation. In: Proc. of ECSI'14; 2014. p. 224–237.

[11] Egly U, Gaggl SA, Woltran S. Answer-set programming encodings for argumentation frameworks. Argument Comput. 2010;1(2):147–177.