

Preferred Discussion Game and Credulous Decision Problem under Admissible Semantics

December 14, 2023

Abstract

This report introduces and implements Preferred Discussion Games and Admissible Semantics for Argumentation Frameworks (AFs). The motivation for the Preferred Discussion Game is to bridge formal argumentation theories with practical applications, making complex concepts more tangible. Admissible Semantics, on the other hand, serves as a vital tool for assessing argument credibility and acceptability in structured reasoning systems. The report provides an overview of the implementation details and testing conducted for the Preferred Discussion Game and Admissible Semantics.

1 Introduction

In this section, we outline the background and motivations behind the creation of preferred discussion games and admissible semantics of argumentation frameworks (AFs).

1.1 Motivation for Preferred Discussion Game

The Preferred Discussion Game acts as a link between the complex theories of formal argumentation and their practical application. Formal argumentation, a facet of non-monotonic reasoning rooted in the work of influential figures such as Pollock, Vreeswijk, and Simari, utilizes the concept of constructing and assessing arguments, which differ from rigid proofs due to their defeasibility—their susceptibility to counterarguments (Caminada et al., 2014).

Dung’s abstract argumentation theory, a cornerstone of contemporary research in this domain, presents AFs using directed graphs (Caminada et al., 2014). These frameworks serve as representations where various semantics—grounded, complete, stable, and preferred—dictate the acceptance of argument sets.

The creation of the Preferred Discussion Game stemmed from the necessity to make these theoretical concepts tangible, comprehensible, and overall more accessible. By simulating dynamic dialogues between proponents and opponents within a structured gameplay environment, this tool immerses users in the intricacies of argumentative strategies. It allows participants to actively engage in the defense or critique of specific claims within AFs, fostering a deeper

understanding of the practical implications of different semantic approaches and loop-handling strategies.

1.2 Motivation for Credulous Decision Problem under Admissible Semantics

In the realm of AFs, the concept of admissible semantics serves as a crucial tool for assessing argument credibility and acceptability within structured reasoning systems. This significance stems from the diverse interpretations of acceptability and credibility of arguments based on different semantic perspectives (Xu & Cayrol, 2018). The incorporation of admissible semantics allows users to delve into the nuanced evaluation of arguments, enabling a deeper exploration of their credibility under defined criteria. By implementing various semantic interpretations, this tool facilitates the understanding of whether specific arguments can be credulously accepted within a given framework. This empowers informed decision-making within complex frameworks.

Recent AI-focused research builds upon Dung’s work, investigating different argumentation semantics. These model arguments as nodes in a directed graph, with edges representing attacks. The introduction of extension-based semantics, such as initial sets semantics, represents a significant advancement in the study of AFs. This extension explores the structural features of existing extensions, offering a broader understanding of argument frameworks. Initial arguments, fundamental in describing grounded extensions, serve as starting points from which attacks are suppressed iteratively until no new initial argument emerges (Xu & Cayrol, 2018). This iterative process culminates in the identification of the grounded extension, representing the least complete extension within the framework.

Recent attention has been given to AFs with collective attacks (SETAFs), where the use of sets instead of singular attackers has sometimes proven advantageous (Dimopoulos et al., 2023). However, SETAFs are limited as they allow sets of arguments to attack only a single argument. Due to this, considering attacks between sets of arguments has received preliminary attention in the research community. Various notions, such as universal defeat, indeterministic defeat, and collective defeat, have emerged to capture different motivations and models of attack between sets of arguments (Dimopoulos et al., 2023). These notions aim to represent phenomena like incomplete information in an agent’s knowledge base or to generalize attack and defense notions applicable to sets of arguments, elucidating emerging properties within structured argumentation.

The exploration of these varied semantic interpretations and extensions within AFs contributes significantly to advancing understanding and applications within AI and structured reasoning systems.

2 Implementation

In this section, we present the implementation details of three key components: argumentation framework, a preferred discussion game and admissible semantics for AFs. The entire implementation is carried out using the Python programming language.

2.1 Argumentation Framework

The initial phase of our implementation involves the development of the *ArgumentationFramework* class. This class is instantiated with the path to a JSON file representing an argumentation framework. The expected format of the JSON file is as follows:

Listing 1: Example JSON File

```
{
  "Arguments": {
    "0": "We should go to the cinema.",
    "1": "We should go to the gym.",
    ...
  },
  "Attack Relations": [
    ["0", "1"], ["1", "0"], ...
  ]
}
```

Alternatively, arguments can be presented as a label-free list, like ["a", "b", "c"], instead of the dictionary format that includes both arguments and labels.

The JSON file is loaded into the program using the *open_json* method, which returns its content. In case the file is not found, the user is prompted to input the correct file path in the console. The extracted arguments and attack relations are then stored as instance variables.

Within the *ArgumentationFramework* class, a method named *get_attackers* is implemented. This method plays a crucial role in identifying all attackers for a given list of arguments, a step essential for subsequent processes.

The implementation of the *ArgumentationFramework* class is located within the *argumentation_framework.py* file.

2.2 Preferred Discussion Game

In this subsection, we present an implementation of a preferred discussion game, engaging both the computer and the user. The computer acts as a proponent, taking on the role of defending a specified argument. On the other hand, the user assumes the role of the opponent, attempting to challenge the provided argument.

2.2.1 Technical Part

The implementation of the Preferred Discussion Game involves creating a class *PreferredDiscussionGame* that extends the *ArgumentationFramework* class. The primary functionality includes methods for obtaining interpretations of sides' choices, choosing arguments by an opponent, and playing the game.

The *get_interpretation* method provides meaningful interpretations for selected arguments. The interpretation depends on the format of arguments in the given framework. If arguments are in a dictionary with labels, the labels are used explicitly. If arguments are in a list without, the interpretation of arguments in this game involves the use of Socratic Discussion. Details on interpretations are provided in sections 2.2.2 and 2.2.3.

The *choose_argument_opponent* method lets the opponent select arguments attacking those presented by the proponent in previous rounds. It prints interpretations (obtained with previously described *get_interpretation* method) of options available for the opponent to choose from, and each option has its own number. The opponent selects an option by typing its number in the console. If the input is invalid, then the opponent should provide the correct number of the option through the console again.

The *play* method initiates the gameplay, where the proponent defends and the opponent attacks. To find options for moves for each side, the previously described method *get_attackers* is used. All arguments used by the proponent are stored in a separate list to find attackers of the proponent's used arguments, that will serve as the opponent's options, and to check if the opponent used an argument previously used by the proponent. Similarly, all arguments used by the opponent are stored in a separate list to check if the proponent used an argument previously used by the opponent, and to guarantee that the opponent won't use the same argument twice. In this implementation, the proponent does not follow any strategy to choose the argument, so it is chosen randomly from available options.

The implementation of the *PreferredDiscussionGame* class is located within the *preferred_discussion_game.py* file.

2.2.2 Design Decisions

At the game start, the proponent's initial claimed argument is printed, followed by the options available for the opponent. Opponent is expected to type a chosen argument using the console. In each round, when one side chooses an argument, its interpretation is printed.

If arguments are provided as a dictionary, which assumes that labels for arguments are provided, options are printed in a format *argument. label*. However, if arguments are provided in a list without labels, the interpretation of arguments in this game involves the use of Socratic Discussion. Players engage in a dialogue where the labeling of arguments as 'in' or 'out' prompts an exchange of questions and answers, resembling the principles of Socratic Discussion. This approach enhances the understanding of the labeled arguments through a collaborative

exploration of their justifications (Caminada et al., 2014). Thus, we have the following interpretations depending on the case for arbitrary chosen argument A and attacked argument B :

- Labels are given:
 - interpretation: given label of argument A
- Labels are not given:
 - Proponent is choosing:
 - * It is the first move:
 - interpretation: "I have an admissible labelling in which A is labelled 'in'."
 - * It is not first move:
 - interpretation: " B is labelled 'out' because A is labelled 'in'."
 - Opponent is choosing:
 - interpretation: "But then in your labelling it must also be the case that B 's attacker A is labelled 'out'. Why?"

If one of the conditions is met that leads to the end of the game, the condition, winner and conclusion of the game are printed. If the opponent wins, the conclusion of the game is, "That is, there is no preferred labelling in which 0 is labelled 'in'."; otherwise, "That is, there is preferred labelling in which 0 is labelled 'in'."

An example of the game looks as follows:

Listing 2: Game example

P: We should go to the cinema.

Consider the following options:

1. We should go to the gym.
2. The gym is better for the health than the cinema.
3. We have no time for evening activities, since there is an exam coming up.
5. We have no money for cinema or gym.

Type argument number that you want to choose: 2

O: The gym is better for the health than the cinema.

Proponent is unable to make a move.

—— Winner: opponent ——

That is, there is no preferred labelling in which 0 is labelled 'in'.

2.2.3 Challenges and solutions

One of the challenges occurred while implementing the *choose_argument_opponent* method was related to a necessity of handling cases when one argument attacks several different proponent's arguments. In situations when arguments are provided with labels, sentences are expected to be written in human language, for example "We should go to the cinema.". Here, knowing which arguments are attacked is not necessary for a human to make a choice. However, if there are no labels for given arguments and the user is playing in a game involving Socratic Discussion, labeling arguments 'out', the user may want to know which proponent's argument is attacked by the opponent's argument, and to distinguish cases when one argument attacks several different proponent's arguments.

From a technical point of view it means that when looking for attackers using *get_attackers* method, we need to know which specific arguments are attacked by attackers. So, *get_attackers* methods return a dictionary with attackers and attacked arguments in a format "argument": "attackers", where *argument* is a string, and *attackers* is a set. Thus, *choose_argument_opponent* method returns a tuple of chosen argument and argument attacked by the chosen argument. These values are used for proper interpretation and distinguishability between different opponent's options.

Such cases lead to the situation when there are different options for the same argument. Thus, if arguments' labels are not provided, to distinguish between these options, they are marked by ordinal number instead of argument name while providing options to the opponent. It is assumed that if labels are provided, then arguments' names are presented as digits.

2.3 Credulous Decision Problem under Admissible Semantics

In this subsection, the implementation of the credulous decision problem under admissible semantics is described.

2.3.1 Technical Part

The implementation of Admissible Semantics involves creating a class *AdmissibleSemantics* that also extends the *ArgumentationFramework* class. It includes methods for , which together determine the admissible semantics of the argumentation framework.

The *conflict_free* method generates a list of all possible subsets of arguments and identifies those that do not contain conflicting relations. Each conflict-free subset includes n arguments, where $n \in [0, n_arguments]$, where $n_arguments$ is the number of arguments in the framework. First, the method defines a list of all possible subsets of size n . Then, for each subset, all possible relations are determined. For example, for a set with $n=2$, $\{a, b\}$, a list of possible relations will include (a, a) , (a, b) , (b, a) , (b, b) . Each possible relation is checked for its presence in the attack relations of a given framework. If all the possible

relations for a subset are not in attack relations, the subset is stored in a list of conflict-free subsets.

The *s_defends_argument* method checks if a given set of arguments defends a specific argument. First, all attackers for the given argument are determined. Then, for each attacker, the method defines a set of defenders, the arguments that defend the argument from its attackers or, in other words, attack argument's attackers. If the sets of defenders and the given set are disjoint, then the argument is not defended from at least one attacker, and the method returns *False*.

The *admissible* method identifies sets of arguments that are admissible, for which all elements in these sets are defended by this set. To check if the set defends each of its elements, *s_defends_argument* method is used.

The *credulous_admissible* method checks if a given argument is credulously acceptable. It explicitly checks if the given argument belongs to at least one admissible set obtained using the *admissible* method.

The implementation of the *AdmissibleSemantics* class is located within the *admissible_semantics.py* file.

2.3.2 Design Decisions

For a given argumentation framework and argument, lists of conflict-free sets and admissible sets, given argument, running time and the result are printed. Result for an argument *A* is outputted in the format: "Yes (No), the given argument *A* is (not) credulously acceptable under admissible semantics in a given AF." Example output is provided below:

Listing 3: Admissible semantics output example

```
cf: [set(), {'a'}, {'b'}, {'c'}, {'d'}, {'c', 'a'},
{'a', 'd'}, {'b', 'd'}]
adm: [set(), {'a'}, {'c'}, {'d'}, {'c', 'a'}, {'a', 'd'}]
argument: d
— Running time: 1.0269 milliseconds —
```

Yes, the given argument d is credulously acceptable under admissible semantics in a given AF.

3 Usage

In this section, the instructions on how to run the implemented preferred discussion game and admissible semantics are provided. For both parts of the implementation, the JSON file must follow the format described in section 2.1., and if the given file does not exist or the given argument is not in the argumentation framework, correct file path and argument should be provided using the console.

3.1 Preferred Discussion Game

The game is designed to be invoked from the command line using the format: `python preferred_discussion_game.py json_file_path.json argument`, for example: `python preferred_discussion_game.py arg_frameworks/example-argumentation-framework.json 0`. By default, the path for the json file with the argument framework is `arg_frameworks/example-argumentation-framework.json`, and the argument is `0`.

3.2 Credulous Decision Problem under Admissible Semantics

Analogously to the preferred discussion game, the admissible semantics is designed to be invoked from the command line using the format: `python admissible_semantics.py json_file_path.json argument`, for example: `python admissible_semantics.py arg_frameworks/af_instance_2.json a`. By default, the path for the json file with the argument framework is `arg_frameworks/af_instance_2.json`, and the argument is `a`.

4 Testing Implementation

In this section, the tests conducted using the implemented preferred discussion game and admissible semantics are described. Plots that are utilized in this section were generated using graph visualization software *Graphviz*. The code for generating plots is located within the `plot_af.py` file.

4.1 Given Examples

Figure 1 shows the results of testing of preferred discussion games for given Argumentation Frameworks (AFs) and claims (arguments). For each AF, the plot is identified, along with the claimed argument. The result section details the end-game condition, winner (either proponent or opponent), and the concluding statement regarding the preferred labeling of the chosen argument. In Example (c), the claimed argument *d* demonstrates that an implementation without specifying the winning strategy for the proponent can lead to an incorrect conclusion if the proponent chooses a path that results in a loss, even when a winning path exists.

Similarly, Figure 2 summarizes the results of testing the Credulous Decision Problem under Admissible Semantics for various AFs and arguments. The result section includes the sets of admissible arguments and the conclusion regarding the credulous acceptability of the chosen argument under admissible semantics in the given AF. Additionally, the running time (in milliseconds) for each scenario is provided.

AF	Plot	Game	
		Argument	Result (condition, winner, conclusion)
(a)		a	Opponent is unable to make a move. (Winner: proponent) That is, there is preferred labelling in which a is labelled 'in'.
		k	(regardless opponent's choice) Proponent is unable to make a move. (Winner: opponent) That is, there is no preferred labelling in which k is labelled 'in'.
(b)		c	Opponent is unable to make a move. (Winner: proponent) That is, there is preferred labelling in which c is labelled 'in'.
		d	Opponent is unable to make a move. (Winner: proponent) That is, there is preferred labelling in which d is labelled 'in'.
		e	(regardless opponent's choice) Argument chosen by opponent was used by proponent. (Winner: opponent) That is, there is no preferred labelling in which e is labelled 'in'.
(c)		b	Opponent is unable to make a move. (Winner: proponent) That is, there is preferred labelling in which b is labelled 'in'.
		d	Argument chosen by opponent was used by proponent. (Winner: opponent) That is, there is no preferred labelling in which d is labelled 'in'.
		e	(regardless opponent's choice) Argument chosen by opponent was used by proponent. (Winner: opponent) That is, there is no preferred labelling in which e is labelled 'in'.

Figure 1: Results of preferred discussion games for given AFs and claims.

AF	Plot	Semantics		
		Argument	Result (admissible sets, conclusion)	Running time (milliseconds)
(a)		a	adm: {set(), {c}, {e}, {a, 'c'}, {e, 'c'}, {a, 'a', 'c'}} Yes, the given argument a is credulously acceptable under admissible semantics in a given AF.	1.7228
		k	adm: {set(), {c}, {e}, {c, 'a'}, {c, 'e'}, {c, 'a', 'e'}} No, the given argument k is not credulously acceptable under admissible semantics in a given AF.	0.6421
(b)		c	adm: {set(), {a}, {c}, {d}, {a, 'c'}, {a, 'd'}} Yes, the given argument c is credulously acceptable under admissible semantics in a given AF.	0.524
		d	adm: {set(), {a}, {c}, {d}, {c, 'a'}, {d, 'a'}} Yes, the given argument d is credulously acceptable under admissible semantics in a given AF.	0.545
		a	adm: {set(), {a}, {c}, {d}, {c, 'a'}, {a, 'd'}} Yes, the given argument a is credulously acceptable under admissible semantics in a given AF.	0.3657
		e	adm: {set(), {a}, {c}, {d}, {a, 'c'}, {a, 'd'}} No, the given argument e is not credulously acceptable under admissible semantics in a given AF.	0.3262
(c)		b	adm: {set(), {a}, {b}, {d, 'b'}} Yes, the given argument b is credulously acceptable under admissible semantics in a given AF.	4.499
		d	adm: {set(), {a}, {b}, {d, 'b'}} Yes, the given argument d is credulously acceptable under admissible semantics in a given AF.	0.325
		e	adm: {set(), {a}, {b}, {d, 'b'}} No, the given argument e is not credulously acceptable under admissible semantics in a given AF.	0.32

Figure 2: Answers to the credulous decision problem under admissible semantics for given AFs and arguments.

4.2 Additional Examples

To examine an end-game condition when the argument chosen by the proponent was used by the opponent and address cases described in section 2.2.3., additional tests for the Preferred Discussion Game were conducted. The results are shown in Figure 3.

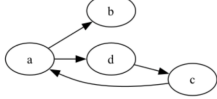
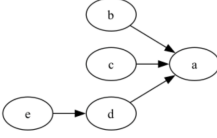
AF	Plot	Game	
		Argument	Result (condition, winner, conclusion)
(d)		b	Argument chosen by proponent was used by opponent. (Winner: opponent) That is, there is no preferred labelling in which b is labelled 'in'.
(e)		a	Opponent's options: 1. But then in your labelling it must also be the case that a's attacker b is labelled 'out'. Why? 2. But then in your labelling it must also be the case that a's attacker c is labelled 'out'. Why? 3. But then in your labelling it must also be the case that a's attacker d is labelled 'out'. Why? Proponent is unable to make a move. (Winner: opponent) That is, there is no preferred labelling in which a is labelled 'in'.

Figure 3: Additional results of preferred discussion games for given AFs and claims.

5 Conclusion

In summary, the report introduces and implements a Preferred Discussion Game, offering a practical link between formal argumentation theories and real-world applications. This tool enables users to actively engage in defending or critiquing claims within argumentation frameworks, fostering a deeper understanding of argumentative strategies. Additionally, the report presents a Credulous Decision Problem under Admissible Semantics implementation for assessing argument credibility and acceptability within structured reasoning systems. In the end, the report outlines the testing procedures undertaken for both aspects.

References

- Caminada, M. W. A., Dvořák, W., & Vesic, S. (2014). Preferred semantics as socratic discussion. *Journal of Logic and Computation*, 26, 1257–1292. <https://doi.org/10.1093/logcom/exu005>
- Dimopoulos, Y., Dvořák, W., König, M., Rapberger, A., Ulbricht, M., & Woltran, S. (2023). Sets attacking sets in abstract argumentation. Retrieved December 14, 2023, from <https://ceur-ws.org/Vol-3464/paper3.pdf>
- Xu, Y., & Cayrol, C. (2018). Initial sets in abstract argumentation frameworks. *Journal of Applied Non-Classical Logics*, 28, 260–279. <https://doi.org/10.1080/11663081.2018.1457252>