KR Project 2: Argumentation

KR 2023

VU Amsterdam

Abstract. This report explores the implementation of loaded argumentation frameworks in a game algorithm, using Python for simulating discussion games within an argumentation framework (AF). Key components include two classes: 'ArgumentationFramework' for argument storage and relation, and 'DiscussionGame' for game flow management. Challenges like enabling player attacks on proponent arguments and optimizing proponent moves in preferred extensions are addressed. The report also includes a program for evaluating credulous acceptance in AFs using preferred semantics, with effectiveness demonstrated through various AF tests. The conclusion showcases the algorithm's practical application in different AF scenarios.

1 Introduction

Knowledge representation plays an integral part in artificial intelligence(AI). Firstly, it serves as a substitute for actual entities, enabling reasoning about the world without direct action. Secondly, it establishes the terms through which we perceive the world. Thirdly, it encapsulates a theory of intelligent reasoning. Fourthly, it provides a computational environment facilitating efficient thinking and organizing information. Lastly, it acts as a language for human expression about the world [1]. In the realm of reasoning, and decision-making there is a theory on argumentation which is also what the loaded argumentation framework is based on.

Loaded argumentation frameworks serve as invaluable instruments for representing diverse perspectives, conflicting viewpoints, and uncertainties inherent in real-world scenarios. They allow for the modeling of intricate relationships between arguments, enabling the examination of how individual assertions support or challenge one another. Through the assignment of labels—such as accepted, rejected, defeated, or undetermined—to arguments based on their relationships within the framework, these models offer a nuanced portrayal of the dynamics of reasoning.

This report aims to go over the implementation of such a framework in a game type algorithm, what are the the challengers and their solutions, design choices and some examples by using different argumentation frameworks(AF).

2 Part I: Preferred Discussion Games

For our implementation of the game, we mainly used python to implement our design.

2.1 Overview

Initialization begins by loading an AF in the form of a JSON file and picking of the arguments from the provided AF as the claimed arguments which is set to IN. With this the computer takes on the roles of the proponent and the player the opponent that is trying to argue against the computer by attacking the claimed argument. This keeps looping where the user depending on the arguments that are set to IN, chooses from possible arguments that attack them and sets them to OUT. This flow is a menu type game which is running in a constant loop until an end is reached. The end is reached if one the 4 rules has been triggered:

1. If the opponent uses an argument previously used by the proponent, then the opponent wins (because he has shown that the proponent contradicts itself)
2. If the proponent uses an argument previously used by the opponent, then the opponent wins (for similar reasons as in the previous point).
3. If the proponent is unable to make a move, then the opponent wins.
4. If the opponent has no choices left, then the proponent wins.

With each new attack on the proponent by the user the arguments get checked against the rules which depending on them define the outcome of the game. For the development of the game an AF has been provided (Figure 3)

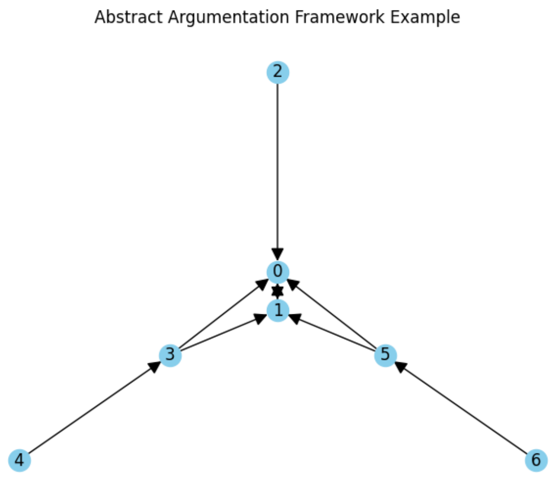


Fig. 1: Argumentation framework 1.

2.2 Implementation Details

In order to implement the game, two classes are defined.

The first is the `ArgumentationFramework` class, which serves to store arguments and attack relations. This class features a method `get_attackers` that facilitates the retrieval of all attackers for a given argument. Additionally, it includes a function `find_preferred_extensions` for obtaining the preferred extensions of the Argumentation Framework. The second class, `DiscussionGame`, is designated for storing intermediary data during the game-play and implements the functional flow of the game.

Initialization Phase Firstly, an instance of the `ArgumentationFramework` class is created, storing the AF (Argumentation Framework) obtained from the JSON file. Then an instance of the `DiscussionGame` class is initialized, incorporating the ArgumentationFramework instance and a claimed argument. During the initialization process of the `DiscussionGame`, three essential sets for the game process are established:

- The `IN` set, designated to store the arguments used by the proponent. Initially, this set contains the claimed argument.
- The `OUT` set, intended for the arguments used by the opponent, starts as an empty set.
- The `UNDEC` set, designed to hold arguments available for the opponent to use in attacking the claimed arguments. Initially, this includes all attackers of the claimed argument.

The `preferred_extensions` method of the `ArgumentationFramework` class is called to acquire the preferred extensions of the AF. For ease of subsequent processing, all arguments within the preferred extensions are consolidated into a single set.

Opponent's Turn The game starts with the opponent's turn. The first step is to determine if the `UNDEC` set is empty. If it is, the proponent wins directly (Rule 4). Otherwise, the opponent selects an argument from the available options in the `UNDEC` collection. This chosen argument is then removed from the `UNDEC` collection and added to the `OUT` collection, this way preventing the re-picking of the same assignment twice.

Next, it is assessed whether the opponent's chosen argument is present in the `IN` set. If it is, indicating that the proponent is self-contradictory, the opponent wins according to rule 1.

If neither of the previous rule checks have been triggered, the game then proceeds to the proponent's turn.

Proponent's Turn First, by searching through attack relations, all arguments that can attack the player's chosen argument are identified. If no such arguments are found, rule 3 is triggered, resulting in the opponent's victory.

If there are valid attacking arguments, the first step involves computing an intersection with the preferred extension set. This step prioritizes arguments

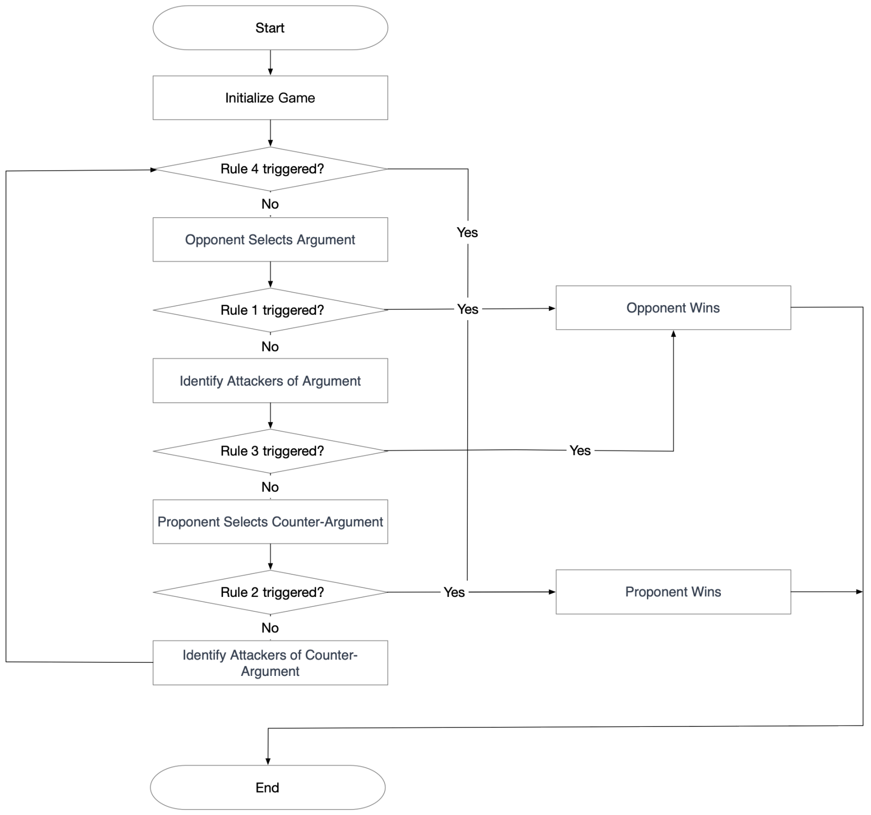


Fig. 2: Flowchart of Game process

within the preferred extension, ensuring that the proponent avoids unwise decisions that could lead to a loss in the presence of a preferred extension. In the event that the intersection is empty, which suggests that the claimed argument does not reside within the preferred extension, the proponent faces an unlikely chance of winning. Under these circumstances, the proponent may choose any argument from the attacking set at random.

Next, it is determined whether the proponent's chosen argument is in the `OUT` set, checking whether it has already been used by the opponent. If so, rule 2 is triggered, and the proponent wins. Otherwise, the chosen argument is added to the `IN` set, and all arguments attacking this chosen argument are then identified, with those already used by the opponent being excluded, and the remainder being added to the `UNDEC` set.

The game then proceeds to loop between these two states of the proponnent and oponent arguing against each other, until it concludes by triggering one of the rules.

2.3 Challenges and solution

Challenges in the Preliminary Prototype In the first iteration of our prototype, the following critical challenges were identified:

1. The absence of a mechanism allowing players to attack previously established arguments of the proponent, leading to game termination upon the conclusion of any branch.
2. The proponent was not programmed to guarantee a win in the presence of a preferred extension, potentially resulting in suboptimal moves.

Design Decisions in the Second Iteration To address the first issue, we introduced labelling semantics with IN, OUT, and UNDEC sets to store arguments used by the proponent, the opponent, and the remaining options available to the opponent, respectively. This allowed opponents to revisit previous nodes and continue the game beyond a branch's end.

Regarding the second issue, two solutions were deliberated:

1. Implementing a method within the `ArgumentationFramework` class to enumerate all preferred extensions under a given AF. This approach would allow the computer to avoid sub-optimal choices by intersecting the set of potential attackers with the preferred extensions at every decision point for the proponent.
2. Applying the labelling semantics algorithm to emulate the preferred extension through iterative attacks and counterarguments.

We adopted the second solution with optimism.

Assessment of the Second Iteration Post-implementation testing of version two revealed that the outcomes were not as expected. For instance, in the AF in Fig.3, when the claimed argument is 1 and the player opts for argument 0, the computer erroneously selects argument 5, consequently losing the game, instead of selecting argument 2, which would secure an immediate victory.

Reevaluation of Design Decisions Considering an alternative, we thought of calculating the complete extension of the IN set for each computer decision to prevent selecting arguments vulnerable to this extension. Yet, the complexity and potential exceptions to this logic, along with computational inefficiencies, led us back to the first proposed solution.

Implementation of Decision Rules Further testing indicated a flaw in our initial belief that the proponent would naturally avoid triggering Rule 2 by prioritizing arguments within the preferred extension. When a preferred extension was absent, Rule 2 did not trigger when it should have, unnecessarily prolonging the game. This was fixed by adding rule 1 and 2 explicitly into the code to ensure correct and efficient flow, addressing the challenges identified in the initial prototype.

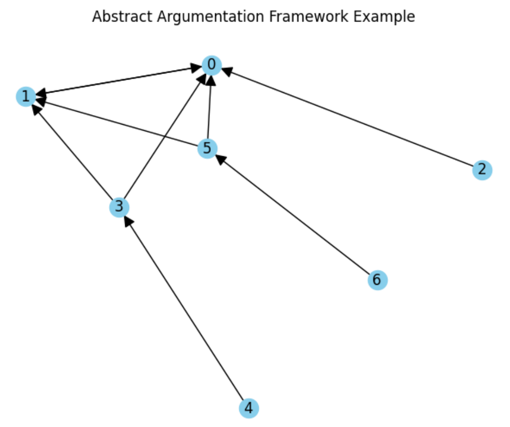


Fig. 3: Argumentation framework for Testing

2.4 Instructions of Discussion Games

To initiate the preferred discussion game, the following command must be executed in the command line window:

```
python discussion_game.py <"file_name"> <"claimed_argument_number">
```

Thereafter, the game starts, and players are required to choose optional attack arguments as prompted by the input box. The game continues until one side secures a victory.

3 Part II: Credulous decision problems in AFs

3.1 Objective

The task involves developing a program that accepts two inputs, the filename of an AF and a specific argument, then determines whether the given argument is credulously acceptable under preferred semantics within the provided AF. We chose preferred semantics due to its relevance in representing maximal admissible sets in AFs, which offers a comprehensive view of argument dynamics.
The program is structured as follows:

1. **Reading the AF**: The AF, provided in a JSON format, is parsed to extract arguments and their attack relations.
2. **Computing Preferred Extensions**:
   - *Conflict-Free Sets*: We first identify sets of arguments that do not attack each other.
   - *Admissible Sets*: From these, we find sets that defend themselves against attacks.
   - *Preferred Extensions*: We then determine preferred extensions as maximal admissible sets.
3. **Determining Credulous Acceptance**: The program checks if the specified argument is included in any of the preferred extensions.

3.2 Implementation

The implementation involves several Python functions, each corresponding to a key aspect of the preferred semantics:

- Functions to compute conflict-free and admissible sets:

```
def defends(attacks, subset, argument):
    return all(any((defender, attacker) in attacks for defender in subset) for attacker,
    ↪  attacked in attacks if attacked == argument)

def is_conflict_free(attacks, subset):
    return not any((a, b) in attacks for a in subset for b in subset)

def is_admissible(arguments, attacks, subset):
    if not is_conflict_free(attacks, subset):
        return False
    for arg in subset:
        if not defends(attacks, subset, arg):
            return False
    return True
```

- Identify all preferred extensions:

```
def find_preferred_extensions(arguments, attacks):
    all_subsets = sorted(powerset(arguments), key=lambda s: len(s), reverse=True)
    preferred_extensions = []
    for subset in all_subsets:
        if is_admissible(arguments, attacks, subset):
            if not any(subset < ext for ext in preferred_extensions):
                preferred_extensions.append(subset)
    return preferred_extensions
```

- Determine if an argument is credulously accepted:

```
def is_credulously_accepted(arguments, attacks, argument):
    preferred_extensions = find_preferred_extensions(arguments, attacks)
    return any(argument in ext for ext in preferred_extensions)
```

3.3 Instructions of the Credulous Decision Problems Tool

For the utilization of the tools associated with credulous decision problems, the following command needs to be inputted in the command line:

```
python preferred_check.py <"file_name"> <"argument_number">
```

Upon execution, the program will return the results of the check.

3.4 Results and Analysis

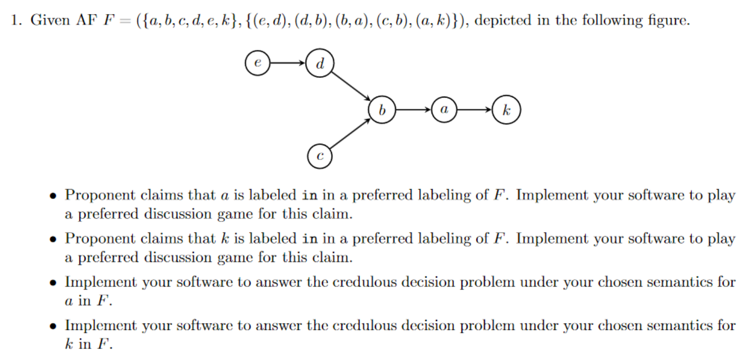
In testing, the custom AF, being more complex, demonstrated longer processing times compared to the example AF. This aligns with expectations, as more intricate argumentation frameworks with a greater number of arguments and attack relations naturally demand more computational resources. Such variations in running times reflect not only the correctness of the algorithm but also its capability to handle AFs of different complexities, underscoring the balance between computational depth and performance in argumentation analysis.

Argument	Accepted?	Time (ms)
A (Custom)	False	2.583
B (Custom)	True	2.563
C (Custom)	False	2.542
D (Custom)	True	2.627
E (Custom)	False	2.553
F (Custom)	True	2.646
G (Custom)	False	2.656
H (Custom)	False	2.558
I (Custom)	True	2.656
J (Custom)	True	2.525
0 (Example)	False	0.442
1 (Example)	True	0.437
2 (Example)	True	0.425
3 (Example)	False	0.420
4 (Example)	True	0.472
5 (Example)	False	0.441
6 (Example)	True	0.422

Table 1: Results and running time of Credulous Acceptance Tests

4 Practical Application and Case Studies

4.1 First argumentation framework



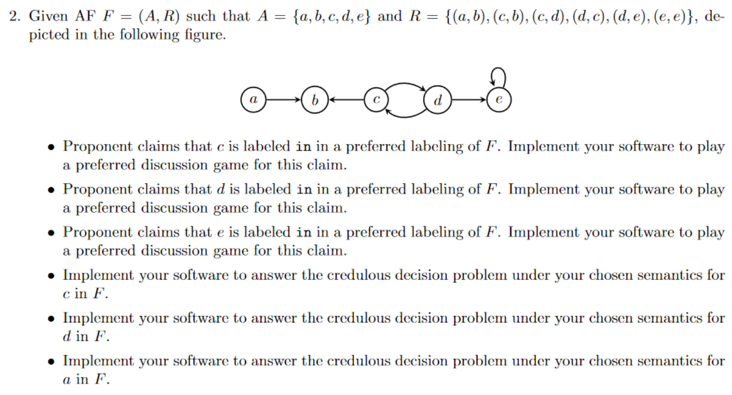
1. Given AF $F = (\{a, b, c, d, e, k\}, \{(e, d), (d, b), (b, a), (c, b), (a, k)\})$, depicted in the following figure.

- Proponent claims that a is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Proponent claims that k is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Implement your software to answer the credulous decision problem under your chosen semantics for a in F.
- Implement your software to answer the credulous decision problem under your chosen semantics for k in F.

Fig. 4: First AF

- P: IN(a) $\rightarrow$ O: OUT(b) $\rightarrow$ P: IN(c) = Proponent wins(rule 4)
- P: IN(k) $\rightarrow$ O: OUT(a) $\rightarrow$ P: IN(b) $\rightarrow$ O: OUT(c) = Opponent wins(rule 3)
- $Cred_{\delta}(a, F) = Yes, Executiontime = 0.004sec$
- $Cred_{\delta}(k, F) = No, Executiontime = 0.0006sec$

4.2 Second argumentation framework



2. Given AF $F = (A, R)$ such that $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$, depicted in the following figure.

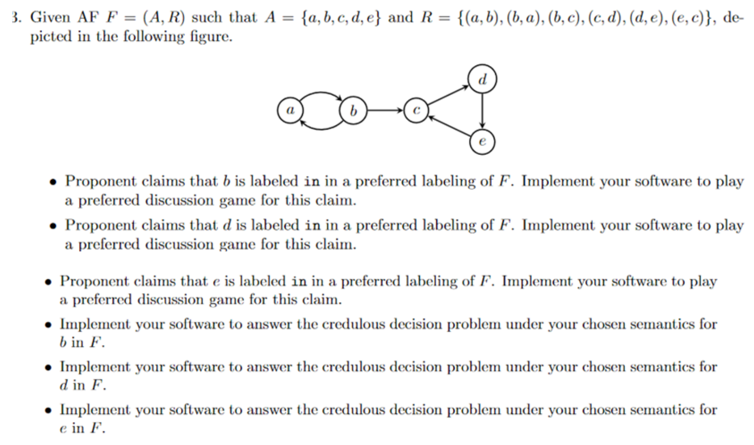
$$\begin{tikzcd} a \arrow[r] & b & c \arrow[l] \arrow[r, bend left] & d \arrow[l, bend left] \arrow[r] & e \arrow[loop above] \end{tikzcd}$$

- Proponent claims that c is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Proponent claims that d is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Proponent claims that e is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Implement your software to answer the credulous decision problem under your chosen semantics for c in F.
- Implement your software to answer the credulous decision problem under your chosen semantics for d in F.
- Implement your software to answer the credulous decision problem under your chosen semantics for a in F.

Fig. 5: Second AF

- P: IN(c) $\rightarrow$ O: OUT(d) $\rightarrow$ P: IN(c) = Proponent wins(rule 4)
- P: IN(d) $\rightarrow$ O: OUT(c) $\rightarrow$ P: IN(d) = Proponent wins(rule 4)
- P: IN(e) $\rightarrow$ O: OUT(e) = Opponent wins(rule 1)
- $Cred_{\delta}(c, F) = Yes,\ Executiontime = 0.006sec$
- $Cred_{\delta}(d, F) = Yes,\ Executiontime = 0.0004sec$
- $Cred_{\delta}(a, F) = Yes,\ Executiontime = 0.0005sec$

4.3 Third argumentation framework



3. Given AF $F = (A, R)$ such that $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}$, depicted in the following figure.

- Proponent claims that b is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Proponent claims that d is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Proponent claims that e is labeled `in` in a preferred labeling of F. Implement your software to play a preferred discussion game for this claim.
- Implement your software to answer the credulous decision problem under your chosen semantics for b in F.
- Implement your software to answer the credulous decision problem under your chosen semantics for d in F.
- Implement your software to answer the credulous decision problem under your chosen semantics for e in F.

Fig. 6: Third AF

- P: IN(b) $\rightarrow$ O: OUT(a) $\rightarrow$ P: IN(b) = Proponent wins(rule 4)
- P: IN(d) $\rightarrow$ O: OUT(c) $\rightarrow$ P: IN(b) $\rightarrow$ O: OUT(a) $\rightarrow$ P: IN(b) = Proponent wins(rule 4)
- P: IN(e) $\rightarrow$ O: OUT(d) $\rightarrow$ P: IN(c) $\rightarrow$ O: OUT(e) = Opponent wins(rule 1)
- $Cred_\delta(b, F) = Yes$, $Executiontime = 0.0005 sec$
- $Cred_\delta(d, F) = Yes$, $Executiontime = 0.0004 sec$
- $Cred_\delta(e, F) = No$, $Executiontime = 0.0005 sec$

References

1. Davis, R., Shrobe, H., Szolovits, P.: What is a knowledge representation? AI Magazine **14** (03 2002)