MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

Université de Paris Cité
UFR Mathématiques et Informatique

KRR: Abstract Argumentation Solver

Master 1 : VMI/IAD

Bertrand Noureddine – Mbodji Papa Abdou – Thiam Aziz Abdou

Année universitaire 2023-2024

1) Goal of the project

The project involves implementing a software solution to address problems related to Argumentation Frameworks (AF). The specific tasks are denoted as :

- VE-CO
- DC-CO
- DS-CO
- VE-ST
- DC-ST
- DS-ST

Where VE-σ means "Verify an Extension with respect to the semantics ", DC-σ means "Decide the Credulous acceptability of the argument with respect to ," and DS- means "Decide the Skeptical acceptability of the argument with respect to ."

The Argumentation Framework (AF) is read from a file, and arguments and attacks are defined using specific lines. The program will be implemented in Python.

Given a specific command line format, the solver will be able to response YES or NO to the given task.

2) Code analysis

Import

We used "os" and "sys" which are standard Python modules for interacting with the operating system and parsing data from the command line. We also used "itertools" and "permutations" for creating iterators which allow efficient looping and easy computation of permutations.

Global Variable

We define two global variable :

- arguments : A set to store argument names.
- attacks : A set to store attack relationships between arguments.

We decided to use Set's over other collection (ie. table or tuples) for various reason.

The most obvious one is the **uniqueness** of element within the set. This ensures that each argument and attack is unique within its respective set. This prevents redundancy and simplifies data management. The second advantage is the **efficiency** in testing if an element is in a given set. Checking if an element is in a set has an average time complexity of O(1). This is crucial for the functions like "is_attacked(arg, attacks)" which iterate over the list of argument and check if the given argument is attacked.

Functions

The following is an non exhaustive list of key function that we used throughout the project.

- is_valid_argument(arg) : Checks if an argument name is valid (alphanumeric and not "arg" or "att").
- process_line(line, line_number) : Processes each line in the input file, extracting arguments and attacks.
- is_attacked(arg, attacks) : Checks if an argument is attacked by any other argument.
- get_arg_attackers(arg, attacks) : Returns a set of attackers for a given argument.
- get_attacked_args(set_of_args, attacks) : Returns a set of arguments attacked by any argument in the given set.
- powerset(iterable) : Computes the power set of an iterable.
- compute_acceptability(arg, E, relations) : Computes the acceptability of an argument in a given set of arguments and relations.
- checkArgumentsInRelations(arguments, relations) : Checks if arguments are defined in relations.
- decide(elem, arg, set1) : Checks if an element is in a set and prints "YES" or "NO" accordingly.
- verify_complete(set1, arg, bigset) : Verifies if a set is a complete extension and prints "YES" or "NO".
- verify_stable(set1, arg, bigset) : Verifies if a set is a stable extension and prints "YES" or "NO".
- process_data(input_data) : Processes input data, checking for commas and returning a tuple for multiple arguments or a single alphanumeric word for one arguments.

Classes

Since the project is quite small we only used two classes : We could have done a better code segmentation but it was not a preoccupation for us since we knew that the project would not scale.

- Extensions : Represents extensions, has methods to get sceptically and credulously accepted arguments.
- Dung : Represents Dung Argumentation Framework, has methods to compute conflict-free sets and admissible sets. Contains a nested 'Semantics' class with methods to compute stable and complete extensions.

3) Algorithm

Conflict free

Our method compute_cfs uses the powerset function to generate all possible subsets of the set of arguments. It then iterates through the attack relations, removing subsets that contain pairs of arguments where an attack relation exists.

The resulting set of subsets represents the conflict-free sets of arguments in the AF. Each subset is conflict-free since, by construction, no pair of arguments within the same subset attacks each other.

Acceptability

The algorithm checks whether the set E defends the argument arg against its attackers. If, for every attacker of arg, there exists at least one argument in E that attacks the attacker, then arg is considered acceptable with respect to E. The function returns a boolean value reflecting the acceptability status of the argument.

Admissible

An admissible set is a conflict-free set in which each argument is acceptable with respect to a given set.

As we did before we ensure that the set of arguments and attack relations are valid.

Then, we compute all conflict-free subsets.

Finaly, we iterate through each conflict-free subset obtained in the previous step. For each conflict-free subset (S), we check if each argument in S is acceptable with respect to S. This is done by ensuring that for every argument A in S, there exists an argument B in S such that (B, A) is in the attack relations. If the condition holds for all arguments in the subset, the subset is considered admissible.

Stable extension

A stable extension is a conflict-free set of arguments that attacks every argument not belonging to the extension. Here is a step-by-step explanation of our algorithm :

First we ensure that the set of arguments and attack relations are valid(ie. the argmuments exists and there are in relations). Then we compute all conflict free subsets(cfr Conflict free). This step involves finding all subsets of arguments where no pair of arguments attacks each other. After we iterate through each conflict-free subset obtained. For each conflict-free subset (S), we check if the union of S and the set of arguments attacked by S is equal to the entire set of arguments. This ensures that the subset attacks every argument not belonging to the subset. If the condition holds, the subset is considered as stable extension and we return the identified stable extensions as a set of sets.

Complete extension

A complete extension is an admissible set of arguments in which each argument, acceptable with respect to the set, belongs to the set.

We make sure that the set of arguments and attack relations are valid like we have done for the stable extension.

Then we compute all admissible sets (S) of the given set of arguments using the compute_admissibility method (cfr Admissible). This step involves finding sets of arguments in which each argument is acceptable with respect to the conflict free set.

Finally, for each admissible set (S), we check if the set of arguments acceptable with respect to S is equal to S. This ensures that each argument, which is acceptable with respect to S, belongs to S. If the condition holds, the set is considered a complete extension.

Skeptical acceptability of arguments

Skeptical acceptance of an argument means that the argument is accepted only if it is accepted in every possible extension.

First, we compute all extensions (stable or complete) using the corresponding methods.

Then we iterate through each argument in the set of arguments. For each argument (A), we check if it is present in every extension. If so, mark it as skeptically accepted.

Credulous acceptability of arguments

Credulous acceptance of an argument means that the argument is accepted if it is accepted in at least one possible extension.

As for Skeptical acceptability, we compute all extensions (stable or complete) using the corresponding methods.

Then we Iterate through each argument in the set of arguments and for each argument (A), we check if it is present in at least one extension. If so, mark it as credulously accepted.