UNIVERSITY “ALEXANDRU IOAN CUZA” IAȘI
FACULTY OF COMPUTER SCIENCE

DISERTATION PAPER

Argumentation Framework with constrains

Proposed by

Drăgoi Ștefan

Session: July 2023

Scientific coordinator

Lect. Dr. Frăsinaru Cristian

Table of Contents

Introduction

In the last ten years, Argumentation has become an important approach in handling Artificial Intelligence problems. The main subject for research from this field is based on the *abstract argumentation* theory defined by Dung (1995). The central concept of this paper is that an argumentation framework is viewed as a directed graph in which arguments are represented as graph nodes and the attacks between arguments are represented as graph arcs. Given a graph described as above, the main tasks are to determine properties of framework's arguments (e.g. an argument is acceptable) and several properties of the framework's subsets (e.g. conflict – free, admissible, stable). Following Dung (1995) definition of abstract Argumentation framework there have been defined a couple of derived notions and extensions. In this paper, intially, we will focus on the theoretically aspects of this topic, providing a relevant and complex of an directed graph representing an Argumentation framework. Afterwards, we will define the most important properties of such a system and then, based on the visual representation, we will explain how and why the graph meets the properties defined. This will be done in order to have a clear and precise understanding of the subject, to be able later to focus on the programatically aspects of the problem. Having the topic defined and well understood, we will focus in the following chapters to address the subject from a more practical point of view. The core of the work is represented by an web application which has as the main backend component an algorithm based on constraints implemented in Java using the library „Choco – solver". The mentioned algorithm is designed to be able to receive as input a set of arguments (nodes) and a set of pairs representing attacks between system arguments and based on the defined constraints, it will be able to provide the properties that the user defined argumentation framework satisfies. The web application allows the user to draw a directed graph and its afferent arcs representing attacks between arguments and obtain the properties from the algorithm implemented in the backend.

1 Practical example of Argumentation

In this century argumentation has become a more relevant subject, most of the companies or schools organizing sessions where its participants are to express their opinion on a certain subject, in this way arguing why they think their perspective is the right one. In this way we can understand that argumentation is a major component of human intelligence. In order to illustrate better the principle of argumentation, we will present an example, a fictional situation where there is an argument between two people A and B, whose countries are at war, about "who is responsible for blocking negotiation in their region" (Dung, 1995). The example has been extracted from Dung article in 1995 referenced at (Dung, 1995).

"My government cannot negotiate with your government because your government doesn't even recognize my government.

A: Your government doesn't recognize my government either. The explicit content of B's utterance is that the failure of B's government to recognize B's government blocks the negotiation. This establishes the responsibility of B's government for blocking the negotiation by an implicit appeal to the following commonsense interpretation rule:

Responsibility attribution: If an actor performs an action which causes some state of affairs, then the actor is responsible for that state of affairs unless its action was justified.

A uses the same kind of reasoning to counterargue that **B**'s government is also responsible for blocking the negotiation as **B**'s government doesn't recognize **A**'s government either.

At this point, neither arguer can claim "victory" without hurting his own position. Consider the following continuation of the above arguments:

B: But your government is a terrorist government.

This utterance justifies the failure of **B**'s government to recognize **A**'s government. Thus, the responsibility attribution rule cannot be applied to make **B**'s government responsible for blocking the negotiation. So, this represents an attack on **A**'s argument. If the exchange stops here, then **B** clearly has the "last word", which means that he has successfully argued that **A**'s government is responsible for blocking the negotiation."

2 Abstract Argumentation framework

In 1995, Phan Minh Dung enunciated the theory of *Abstract Argumentation framework* as "a pair of a set of arguments, and a binary relation representing the attack relationship between arguments. An argument is an abstract entity whose role is solely determined by its relations to other arguments" (Dung, 1995). The framework has been defined as abstract because in the article there has not been paid attention to the internal structure of the arguments, but on the relationships and the properties determined by these.

An argumentation framework is defined as a pair:

AF = <S, attacks>,

where S is a set of arguments and *attacks* is a binary relation on S (attacks ⊆ S x S).

In this paper, we will represent an attack between two arguments A and B as (A, B), which means that A attacks B.

In the following subchapters we are going to define and describe each property an argumentation framework can satisfy. In order to have a clear and correct perspective over the concept of these, I have decided to create a complex and relevant example with multiple arguments and several relationships between them (attacks).

The example of this framework we propose consists of a set S of thirteen arguments:

- S = {A, B, C, D, E, F, G, H, I, J, K, L, M}

and a set R of eighteen attacks between arguments (in Figure 1 attacks are represented as arcs between nodes)

- R = {(A, B), (B, D), (D, C), (C, A), (D, E), (E, F), (G, F), (J, G), (D, L), (L, J), (H, G), (I, G), (H, I), (I, H), (K, H), (K, I), (M, K), (K, M)}.

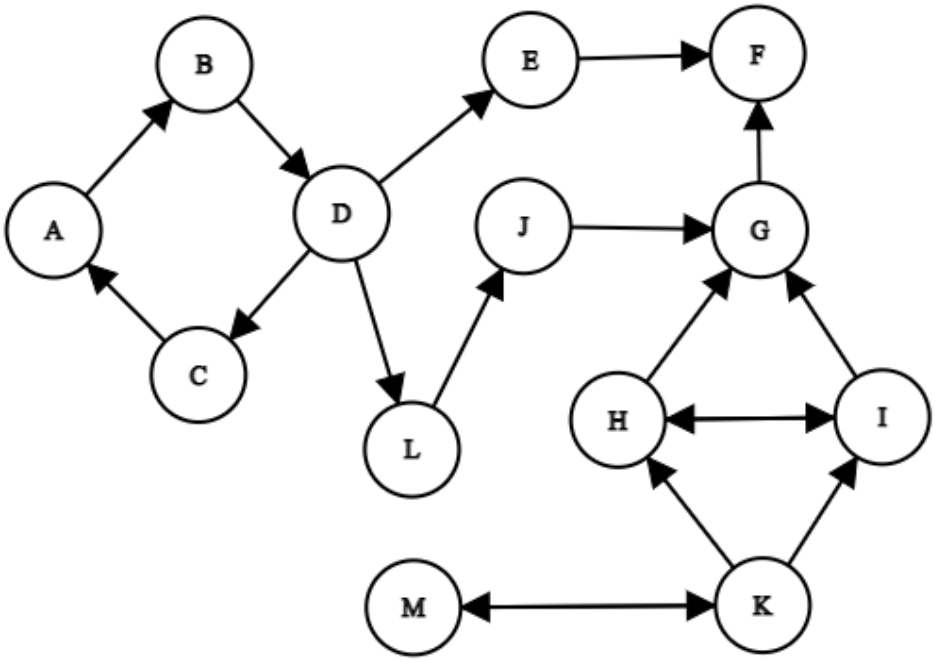

Figure 1 – Example 1 of argumentation framework graph

2.1 Attacks

As defined above, in abstract argumentation framework is a binary relation on S (the set of arguments) which in the example provided in Figure 1 is represented as an arc between two nodes. In our example, there exists an attack from A to B written in this paper as **(A, B)**. The attacks can be mutual, and in the example, there is an attack from node M to K and another attack from K to M. It is worth to be mentioned that, even if it is not illustrated in the graph, in an argumentation framework an argument can attack itself.

2.2 Attackers and defends

In an abstract argumentation framework, two properties of an argument are defined: to be an attacker and to defend.

- A node A ∈ S is said to be an **attacker** if ∃ B ∈ S such that (A, B).

In Figure 1 all arguments are attackers. For our example, argument B is an attacker because it attacks D: (B, D) and also argument D is an attacker because (D, C), (D, E) and (D, L). The notion of *attack* can be applied for an argument and a set of arguments. An argument A attacks a set S' if and only if ∃ B ∈ S' such that (A, B). Similarly, a set S' attacks an argument A if and only ∃ B ∈ S' such that (B, A).

- A node A ∈ S is said to be that **defends** a node C ∈ S, if ∀ B ∈ S such that (B, C), then (A, B) (i.e. A defends C if A attacks all arguments that attack C).

In the provided example, argument A defends D because A attacks all the arguments that attack D (i.e. B). In this case, the "defending" is mutual because D defends A due to the attack (D, C), C being an attacker for A. It is important to mention that we can say that a set of arguments *defend* an argument or another set of arguments. In the example, {L, K} defends argument G because: (K, H), (K, I) and (L, J), H, I and J being attackers for G. At the same time, argument B defends the set of arguments {C, E, L} because (D, C), (D, E), (D, L) and B attacks D (B,D), so it is defends these three arguments.

2.3 Conflict – free set

- A set S' of arguments is said to be **conflict – free** if for ∀ A, B ∈ S', then there are no attacks between A, B.

A relevant mention in this subchapter is that S' can contain only one node. As mentioned before, an argument can attack itself and then the set made up of that argument is not conflict – free. In the example presented, there are couple of this kind of sets: {A, D, J, F, H, M}, {B, C, E, L, G, K}, {K, J, F, D}.

2.4 Acceptable arguments

In abstract argumentation framework:

- An argument A is defined as **acceptable** with respect to a set of arguments S', if ∀ B such that (B, A), then B is attacked by S'.

This means that the argument is acceptable if there exists a set of arguments which defends it from all its attackers. For our example from Figure 1 – Example 1 of argumentation framework graphFigure 1, all arguments are acceptable. In that case it is simple to obtain this property since there is no argument in the framework which is not attacked by another argument. Let us give

some concrete examples: F is an acceptable argument because it is attacked by E and G. Both two are attacked by another two arguments as follows: (D, E) and (J, G). In the same way, the argument M is acceptable because it is attacked by K and M is defended by itself by attacking node K (we called it a mutual attack)

2.5 Admissible set

The property of admissibility is one of the most relevant when talking about abstract argumentation framework. This one defines a strong relationship between members of a set as follows:

- A set S' of arguments is said to be **admissible** if and only if:
  - S' is conflict – free;
  - For ∀ A ∈ S', if B attacks A (B, A), then S' attacks B;

The definition can be stated more informally: a set is admissible if there are no attacks between arguments of that set (conflict – free) and for any argument from the set is attacked by an outsider, then another argument from the set attacks the attacker. Another explanation for the second condition of the property is that the arguments of the admissible set defend themselves. As mentioned for previous properties, also admissibility can be applied to a set with only one argument. To have a clear and better understanding, we will explain the property using the example provided in Figure 1. The shortest and simple admissible sets are {M} and {K} because M does not attack itself and is attacked by K being able be defended by attacking back node K, thus being defended. Due to the fact that M is defended by an argument of set {M}, this makes the set admissible. The same logic applies to set {K}. Another small admissible set is {A, D}. There are no attacks between them, so satisfy the first rule. The second rule is also satisfied with the following explanation: A is attacked by argument C, which is attacked in turn by an argument of the set {A, D} (i.e. D with (D, C)). D is attacked by B, which is attacked by an argument of the set (i.e. A with (A, B)). Other admissible sets are: {B, C, E, L}, {M, H, F, D}, {D, A, J, F}, {A, D, J, M, H}, {A, D, J, F, M, H}, {B, C, E, L, G, K}.

2.6 Preferred extension

As Dung (1995):

- A **preferred** extension is "a maximal (with respect to set inclusion) admissible set of AF", where AF is Argumentation Framework.

Defining above what an admissible set means, we can select the maximal sets in order to obtain the preferred ones: {A, D, J, F, M, H} and {B, C, E, L, G, K}. Having these two definitions, we can extract an important theorem of an argumentation framework related to admissibility and preference: for ∀ admissible sets, there ∃ at least one preferred set.

2.7 Characteristic function

In abstract argumentation framework is defined a function named the **characteristic function** which has the following definition:

- $F : 2^S \rightarrow 2^S$
- F(S') = {A | A is acceptable with respect to S'}

The definition on characteristic function can be expressed with other words as follows: F(S') = the set of arguments which are defended by the arguments of S'. Referring the graph from the Figure 1 we can calculate the characteristic function of some arguments: F({A}) = {D} because A defends only one argument in the graph which is D; F({D}) = {A, F, J}, by attacking K, E and L, D is defending A, F and J. Function F can be calculated for a set with more than one argument: F({L, K}) = {G, K, I, H}, F({A, L, I}) = {D, G, I, F}, F({B, C, D, H}) = {A, B, C, D, E, L, J, F, H, G}.

2.8 Complete extension

As the property of a set to be admissible, to be a complete extension is also one of the most important one in an abstract argumentation framework. It defines also a strong relationship between members of a set:

- A **complete** extension is an admissible set of arguments if and only if for ∀ A ∈ S', A acceptable with respect to S', A belongs to S'.

With other words, a set S' is complete if all the arguments that set S' defends are included in S'. The fact that we have defined the characteristic function before, allows us to give another definition of complete extension: a set S' is complete if F(S') = S'.

In the argumentation framework provided as example in the chapter 1, there is a complete extension: S' = {A, D, F, J}. We can proof this by calculating the characteristic function for each argument of the set S' and using the definition of completion F(S') = S'. To avoid any confusion related to the letter F whether is an argument or the characteristic function, for this proof, we will note **argument** F as F", so the complete function S' now becomes {A, D, F", J}.

- F(A) = {D};
- F(D) = {A, F", J};
- F(F") = ∅;
- F(J) = {F"}.

Having he characteristic function calculated for each argument of set S', we can calculate now F(A, D, F", J) = F(A) ∪ F(D) ∪ F(F") ∪ F(J) = {A, D, F", J} = S'. So, the relation for a complete extension is satisfied: F(S') = S'.

2.9 Grounded extension

In subchapter 2.6 we defined the property of a set to be preferred as being the maximal admissible set in an argumentation framework. Also defining complete extension and the characteristic function, we can define the notion of grounded extension which is the opposite of a preferred extension:

- A **grounded** extension is the least fixed point of F (characteristic function).

In the example presented in Figure 1 a grounded extension is the empty set {}.

2.10 Stable extension

Another important property of a set in an argumentation framework is to be stable. We will provide de definition of this one followed then by some examples to have a good understanding of it:

- A set of arguments S' is defined as **stable** extension if

- S’ is conflict – free
- $\nexists$ A $\in$ S \ S’ such that S’ does not attack A, where S is the set of all arguments in an argumentation framework and S’ is a subset of it.

With other words, a set S’ is a stable extension if and only if its arguments attack all the arguments of the framework that do not belong to S’. In our example, some of the stable extensions are: {B, C, E, L, H, M}, {B, C, E, L, G, K}.

2.11 Threat and defender

In an abstract argumentation framework there are defined the notions of an argument which is a threat for a set and an argument which is a defender for a set:

- An argument A is said to be a threat for a set of arguments S’ if: for $\forall$ B $\in$ S’, then (A, B) and $\nexists$ (B, A).

More formally, an argument is a threat for a set if it attacks all the arguments of the set and none of the set arguments attacks A. In the example, argument D is considered a threat for the set {C, E, L} because (D, C), (D, E), (D, L) and none of these three arguments attack D. In the same manner K is a threat for {H, I}. On the other hand, K is not a threat for M because M attacks K back, so the property of stabilization does not stand.

- An argument A is defined as a defender for a set S’ if it attacks the threat of S’.

In the example from the Figure 1, a defender for the set {C, E, L} is B because node B attacks the threat of the set, argument D. Another example of defender is J for the set {F} because there is an attack (J, G) and G is a threat for {F}.

3 Algorithm

As stated in the previous chapters, the main purpose of this dissertation work is to provide a web application in which the users are able to add arguments as graph nodes in a canvas, add attacks between them (directed edges / arcs) and get the argumentation framework properties. Besides the user interface, there is an algorithm based on constraints implemented in Java. In order to determine the properties I used a java library called Choco-solver which provides an important amount of methods through which the constrains are defined and a solver which retrieve the problem solution based on the defined constrains.

3.1 Constraint Programming

"Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics" (Francesca Rossi, 2006). A Constraint Satisfaction Problem (CSP) is defined by a set of variables, each one with a domain of values, and a set of constraints defining the feasible combinations of values. Solving a CSP consists of assigning each variable a value in its domain such that all constraints are satisfied. It is then possible to add one or many optimization criteria. The term Constraint Programming (CP) comes from the fact that the problem is described in a declarative way through its constraints. It is a very nice programming paradigm where, instead of giving a procedure about how to solve the problem, the user simply specifies the list of properties that must hold in a solution: "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." (Freuder, 1997). This clear separation between the technical aspects and the functional aspects goes beyond programming. It is a different way of thinking that is highly valuable when talking to business representatives for industrial projects.

Once the CP model has been established, it can be solved using a generic CP solver, such as Choco Solver.

3.2 Choco-solver

Choco Solver is an Open Source Java library for Constraint Programming that has been used for more than 20 years by many companies and universities. Choco is among the most efficient solvers available and has a high code quality. Being a Java library, it can be integrated very easily to develop innovative web-services for decision making. Choco-solver comes with:

- various type of variables (*integer, boolean, set, graph* and *real*);
- various state-of-the-art constraints (*alldifferent, count, nvalues,* etc.);
- various search strategies, from basic ones (*first_fail, smallest,* etc.) to most complex (impact-based and activity-based search);
- explanation-based engine, that enables conflict-based back jumping, dynamic backtracking and path repair.

3.3 Argumentation framework algorithm

As stated in the chapter 2, an Argumentation Framework could have a set of properties. In many articles, especially (Dung, 1995), these properties are enounced as *Acceptability semantics* as follows: Let F = (A, R) be an AF and B ⊆ A.

- B is an admissible set iff it is conflict-free and defends its elements.
- B is a preferred extension iff it is a maximal (for set ⊆) admissible set.
- B is a stable extension iff it is a preferred extension that attacks any argument in A\B.
- B is a complete extension iff it is conflict-free and it contains all the arguments it defends.

Example 2. Let us consider an Argumentation Framework: $F_1 = (A_1, R_1)$, where A_1 = {A, B, C, D} and R_1 = {(A, B), (B, A), (A, C), (C, D)}. In order to have a better understanding, we will present the directed graph associated with F_1:

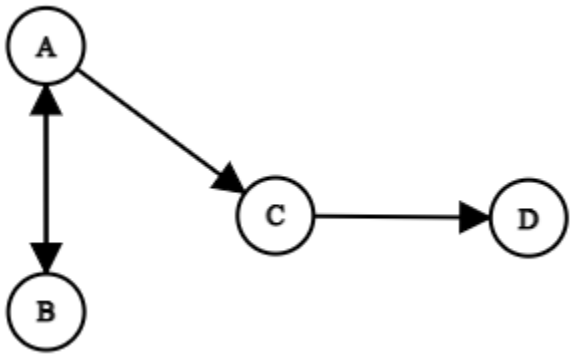

Figure 2 - Example 2 of Argumentation Framework graph

In the following sections we will present five mapping from Dung's Abstract Argumentation properties into CSP. The approach is the following: having as example the argumentation framework designed in Figure 2 - Example 2 of Argumentation Framework graph Figure 2, we will define a CSP instance which will be the representation of a property as a Choco-solver constraint. The solutions of that constraint are the extensions (sets of arguments) satisfying the desired properties. For each property expressed as a constrained, the arguments are declared as variables. Each variable can take two values: 0 or 1 having the following meaning: the arguments is or is not included in the extension, so the type of these variables is bool. The associated type in Choco-solver is *BoolVar*.

3.3.1 Conflict – free as CSP

The first and most relaxed constrained in an Abstract Argumentation Framework is conflict – free. As mentioned and described in the chapter Conflict – free set2.3, a conflict – free extension is a set of arguments that do not attack each other. In this case, having Figure 2 as example, there is an attack between arguments A and B (expressed as (A, B) ∈ R), this will be transformed in a constraint that states that the variables A and B cannot take the value 1 (or true) at the same time in the solution. Respecting the structure of a constraint, this one has the following form: ((A, B, C, D), (1, 0, 0, 1), (0, 1, 1, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 0, 0, 0)). It is important to mention that the equivalent formula for the above constraint is A ⇒ ¬B, for (A, B) ∈ R.

Therefore, we can define the propositional formula for a conflict – free extension in an Argumentation Framework:

Definition 1: Let F(A, R) be an Argumentation Framework. A free – conflict constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { a ⇒ ¬b | (a, b) ∈ R }.

Applying the definition written above on example from Figure 2, the set C = { A ⇒ ¬B, B ⇒ ¬A, A ⇒ ¬C, C ⇒ ¬D }.

In Choco-solver there are defined methods to express propositional logic, so in the algorithm, we iterate over the attacks in the graph and *post* the constraint to the model.

3.3.2 Stable as CSP

As mentioned in in the subchapter Conflict – free set 2.10, a stable extension is a set S' of arguments with the property that the arguments of the set attack all the other arguments in the argumentation framework that do not belong to S'. For simplicity reasons, we will continue with the same approach as described in the previous sub subchapter to define the constraint, the domain for the variables and other specifications to express the transformation from abstract argumentation framework property to propositional logic.

Definition 2: Let F(A, R) be an Argumentation Framework. A stable constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { a ⇔ $\bigwedge_{b:(b,a)\in R} \neg b$ | a ∈ A }.

It is worth to mention that the Definition 2 has been inspired from (Creignou, 1995).

Illustrating the definition stated above about stable constraint on the Argumentation Framework example from Figure 2 we obtain the following set of constraints *C* from CSP definition:

C = { A ⇔ ¬B, B ⇔ ¬A, C ⇔ ¬A, D ⇔ ¬C }.

Solving the problem with the respective constraint from set C, the solution containing stable extensions is S = ((A, B, C, D), (1, 0, 0, 1), (0, 1, 1, 0)). Thus, the two stable extensions are {A, D} and {B, C}.

3.3.3 Admissible as CSP

In the subchapter 2.5 we described what an admissible extension means: a set is said to be an admissible extension if and only if there are no attacks between arguments of the set (is a conflict – free set) and for any argument from the set is attacked by an outsider, then another argument from the set or the attacked argument itself attacks the attacker. To be able to apply these, we need to transform it to propositional logic as stated in the following definition:

Definition 3: Let F(A, R) be an Argumentation Framework. An admissible constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { (a ⇒ $\bigwedge_{b:(b,a)\in R} \neg b$) ∧ (a ⇒ $\bigwedge_{b:(b,a)\in R}(\bigvee_{c:(c,b)\in R} c)$) | a ∈ A }.

It is important to mention that the Definition 3 has been inspired from (Leila Amgoud, 2011).

Applying this definition to the example designed in Figure 1Figure 2, we will obtain the following set of constraints C:

C = { B ⇒ ¬A , A ⇒ ¬B, (C⇒ ¬A) ∧ (C ⇒ B), (D ⇒ ¬C) ∧ (D ⇒ A) }.

Given the fact that arguments can take only values from set (0, 1), the solution of admissible constraint over the example presented in Figure 2 is: ((A, B, C, D), (0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (1, 0, 0, 1), (0, 1, 1, 0)). Transforming the binary solution into argumentation framework arguments the resulted admissible sets are: {{}, {A}, {B}, {A, D}, {B, C}}.

3.3.4 Complete as CSP

The notion of a complete extension has been defined in the subchapter 2.8 as a set that does contain arguments with the property that it is conflict – free, it is admissible and in addition to these, all the arguments that are defended by any argument of this set belongs to respective set. The last restriction can be translated in other words as: an argument of a complete extension does not defend another argument that does not belong to the complete set. The propositional logic for the complete extension is written in the below definition:

Definition 4: Let F(A, R) be an Argumentation Framework. An complete constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall\ x_i \in S$, D_i = {0, 1}, C = { $(a \Rightarrow \bigwedge_{b:(b,a)\in R} \neg b) \wedge (a \Leftrightarrow \bigwedge_{b:(b,a)\in R}(\bigvee_{c:(c,b)\in R} c))$ | a ∈ A }.

It is worth to mention that the Definition 4 has been inspired from (Leila Amgoud, 2011).

As we can observe, the constraints for admissible extension and complete extension are similar, the admissible constraint being more relaxed than the complete one.

C = { B ⇒ ¬A , A ⇒ ¬B, (C⇒ ¬A) ∧ (C ⇒ B), (D ⇒ ¬C) ∧ (D ⇒ A) }.

This expression could have been applied also as a propositional logic in the implemented algorithm, but due to the fact that one of the main target of the tool that we want to develop in within this dissertation work is performance, we do not solved this Argumentation Framework property as a CSP because it would have cost us extra memory and time, so the complexity of the algorithm would have increased a lot, besides that, we made use of the other definition of a complete extension also described and exemplified in detail in the subchapter 2.8. To be able to present the other definition, we need to remind what is the characteristic function in an Abstract Argumentation Framework. This concept has been detailed in the subchapter 2.7. Summarily, the characteristic function of a set of arguments is equal with the set of arguments that are defended by the first set. The definition of a complete extension that we used in the implementation is the following: a complete set S' is an admissible set with F(S') = S'. Therefore, to have a faster algorithm, we used this definition in our implementation. Having the admissible from the previous method in the algorithm, the next step is to calculate the characteristic function for each admissible resulted set and to determine those which are equal with the set for whose characteristic function is calculated.

Determining the characteristic function of the obtained admissible sets for the example provided in Figure 2, the results are the following:

F({B}) = {B, C}, admissible extension {B} is not complete;

F({A}) = {A, D}, admissible extension {A} is not complete;

F({A, D}) = {A, D}, admissible extension {A, D} is complete;

F({B, C}) = {B, C}, admissible extension {B, C} is complete.

3.3.5 Preferred extension

The definition of a preferred extension has been provided in the subchapter 2.6 as "a maximal (with respect to set inclusion) admissible set of AF" (Dung, 1995). For the same reason

as the previous approach with the complete extension, we did not use a constraint to determine the preferred extensions in the provided Argumentation Framework. Instead, we implemented this functionality using two maps:

- One having as key an argument and as value a list of arguments that the key argument;
- One having as key an argument and as value a list of arguments which attacks the key argument.

It is worth to mention that we used the same maps in the previous step when we calculate the characteristic function for a given set of arguments.

Bibliography

Dung, P. M. (1995). *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.* Bangkok 10501, Thailand: Division of Computer Science, Asian Institute of Technology, GPO Box 2754.

Creignou, N. (1995). The class of problems that are linearly equivalent to Satisfiability or a uniform method for proving NP-completeness. *Theoretical Computer Science*.

Francesca Rossi, P. v. (2006). *Handbook of Constraint Programming.*

Freuder, E. C. (1997). *Constraints.*

Leila Amgoud, C. D. (2011). Argumentation Frameworks as Constraint. *International Conference on Scalable Uncertainty Management*.

Choco-solver. (n.d.). Retrieved from https://choco-solver.org/