Supplementary Material: A Basic Framework for Explanations in Argumentation

Algorithm for the Computation of the Explanations

AnneMarie Borg and Floris Bex

This document contains a discussion on the computation and computational complexity of the basic explanations. In particular, we will provide a polynomial time algorithm that computes the explanations, once the extensions of the argumentation framework are known.

Some preliminaries

Since an (abstract) argumentation framework (AF) [1] can be seen as a directed graph, one can determine whether argument B is reachable from argument A and if so, what the distance between the two arguments is. If B is reachable from A, it can be determined, based on the distance, whether A (in)directly attacks or (in)directly defends B. These notions will be useful in calculating explanations.

Definition 1. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an AF and let $A, B \in \mathsf{Args}$. There is an *attack-path* from A to B if $(A, B) \in \mathsf{Att}$ or there are $C_1, \ldots, C_{n-1} \in \mathsf{Args}$, such that $(A, C_1), (C_1, C_2), \ldots, (C_{n-2}, C_{n-1}), (C_{n-1}, B) \in \mathsf{Att}$ and no attack appears twice in this sequence. It is said that this attack-path has *length n* and is along the attacks $(A, C_1), (C_1, C_2), \ldots, (C_{n-1}, B)$, if $(A, B) \in \mathsf{Att}$, the path has length 1 and if $A = B$ the attack-path has length 0.[1]

Intuitively, if the length of an attack-path between arguments A and B is odd [resp. even], A (in)directly attacks [resp. defends] B.

The algorithm

The main idea of the basic framework for explanations is that the explanations can be calculated from any argumentation framework, when it is know if an argument is accepted and how (i.e., whether an argument is skeptically or credulously (non-)accepted and under what semantics). In order to compute these explanations we introduce an algorithm that can be applied after the acceptance of the arguments in the framework is determined.

Algorithm 1 presents a method to calculate the length of all existing attack-paths in an argumentation framework (Dist) and for each argument A the set of arguments from which a path to A exists ($\mathsf{Reach}(A)$). The algorithm is based on Procedure ReReach (*Recursive Reach*), a depth-first search algorithm.

Note that the run time of the algorithm is finite when Args and Att are finite. This is the case since for each $A \in \mathsf{Args}$ at most all attacks in Att are considered $|\mathsf{Att}|$ times.

Complexity results

Algorithm 1 has some desirable properties. In particular, it is sound and complete (i.e., for the requested argument A it finds all the arguments from which A can be reached and the length of the attack-paths between those arguments) and it runs in polynomial time. The latter is useful since it shows that the computational complexity of computing the explanations for a certain semantics

[1]Note that an attack-path is known as a *trail* in graph theory.

Algorithm 1: Computing Reach and Dist.

```
Result: Reach and Dist are computed.
1 Given an AF $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$;
2 for $A \in \mathit{Args}$ do
3     $\mathsf{Reach}(A) = \{A\}$ and $\mathsf{Dist}(A, A) = \{0\}$;
4     for $B \in \mathit{Args} \setminus \{A\}$ do
5         $\mathsf{Dist}(A, B) = \emptyset$;
6 for $A \in \mathit{Args}$ do
7     ReReach$(A, A, 0, \emptyset)$;
```

Procedure ReReach(A, A', n, S)

```
1 $\mathsf{Visited}_0 = \mathsf{S}$;
2 for $A^\star \in \mathit{Args}$ s.t. $(A^\star, A') \in \mathit{Att}$ and $(A^\star, A') \notin \mathit{Visited}$ do
3     $\mathsf{Reach}(A) = \mathsf{Reach}(A) \cup \mathsf{Reach}(A^\star)$;
4     $\mathsf{Dist}(A^\star, A) = \mathsf{Dist}(A^\star, A) \cup \{n + 1\}$;
5     $\mathsf{Visited} = \mathsf{Visited} \cup \{(A^\star, A')\}$;
6     ReReach($A$, $A^\star$, $n + 1$, $\mathsf{Visited}$);
7     $\mathsf{Visited} = \mathsf{Visited}_0$;
```

is not more complex than computing the acceptance of an argument and/or the extensions under that semantics [2].

Theorem 1 (Soundness and completeness)**.** *Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an AF. Then:*

1. *there is an attack-path from A to B of length n iff $n \in \mathsf{Dist}(A, B)$;*
2. *$A \in \mathsf{Reach}(B)$ iff there is an attack-path from A to B;*

In order to show the above theorem, we need some lemmas and propositions. These lemmas and propositions will partially be shown by induction proofs, for which the following remark will be useful.

Remark 1. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an AF and let $A, B \in \mathsf{Args}$. It holds that $A = B$ iff there is an *attack-path* from A to B that has length 0. Similarly, $(A, B) \in \mathsf{Att}$ iff there is an attack-path from A to B of length 1.

Lemma 1. *If ReReach(A, A', n, S) is called, there is an attack-path from A' to A, of length n, along the attacks in S.*

Proof. Suppose that ReReach(A, A', n, S) is called, either at Line 7 of Algorithm 1 or at Line 6 of Procedure ReReach. We proceed by induction on n.

- **If $n = 0$:** then ReReach$(A, A, 0, \emptyset)$ is called at Line 7 of Algorithm 1. Since $A = A'$, by Remark 1, there is an attack-path of length 0 from A' to A, without any attacks.
- **If $n = 1$:** then ReReach(A, A', n, S) was called at the first iteration of Procedure ReReach. Hence $(A', A) \in \mathsf{Att}$ and $(A', A) \notin \emptyset$. By Remark 1, there is an attack-path of length 1 from A' to A.

Suppose now that the claim holds for n up to $k \geq 1$.

- **If $n = k + 1$:** then there is some $B \in \mathsf{Args}$ such that ReReach(A, A', n, S) is called at Line 6 of the call ReReach(A, B, k, S') where $\mathsf{S}' = \mathsf{S} \setminus \{(A', B)\})$. To see that $\mathsf{S}' = \mathsf{S} \setminus \{(A', B)\}$, note that $\mathsf{Visited}$ is updated at Line 5 with (A', B) before ReReach(A, A', n, S) is called and if $(A', B) \in \mathsf{S}'$, the call to ReReach(A, A', n, S) would not be reached.

By induction hypothesis, there is an attack-path from B to A of length k along the attacks in S'. Since $(A', B) \notin \mathsf{S}'$, it follows that the attack (A', B) was not used in the attack-path from B to A. Therefore, the path from A' to A along (A', B) and the attack-path B to A is an attack-path from A' to A of length $k+1$ along the attacks in S. □

The next proposition shows that Algorithm 1 is sound.

Proposition 1. *If $n \in \mathsf{Dist}(A, B)$ then there is an attack-path from A to B of length n.*

Proof. Suppose that $n \in \mathsf{Dist}(A, B)$, that there is an attack-path from A to B of length n is shown by induction on n:

- **If $n = 0$:** then $\mathsf{Dist}(A, B)$ was updated at Line 3 of the algorithm (since at any other place that $\mathsf{Dist}(A, B)$ might be updated, the addition is always more than 0). It follows immediately that $A = B$. Hence there is an attack path from A to B of length 0.
- **If $n = 1$:** then $\mathsf{Dist}(A, B)$ was updated at Line 4 of the procedure in the first iteration of the **for**-loop (since in any other iteration $n \neq 0$). It follows that $(A, B) \in \mathsf{Att}$. Hence the attack-path consists of one attack: (A, B). Thus there is an attack-path from A to B of length 1.

Suppose that the proposition holds for values of n up to k, where $k \geq 1$. Then:

- **If $n = k + 1$:** then $\mathsf{Dist}(A, B)$ was updated at Line 4 of Procedure ReReach. This is only the case if there is some $C \in \mathsf{Args}$ such that ReReach(B, C, k, S) was called and $(A, C) \in \mathsf{Att}$ such that $(A, C) \notin \mathsf{S}$. By induction hypothesis, there is an attack-path from C to B of length k and by Lemma 1, this attack-path is along the attacks in S. Since $(A, C) \notin \mathsf{S}$ by assumption, the path from A to B via the attack (A, C) and the attack-path from C to B is an attack-path, of length $k + 1$.

This shows that for any n, if $n \in \mathsf{Dist}(A, B)$, there is an attack-path of length n from A to B. □

In the next lemma the relation between the attacks in an attack-path and the attacks in $\mathsf{Visited}$ in the procedure is shown.

Lemma 2. *If there is an attack-path from A to B, along the attacks $(A, C_1), (C_1, C_2), \ldots, (C_{n-1}, B) \in \mathsf{Att}$, for $C_1, \ldots, C_{n-1} \in \mathsf{Args}$, then during the run of the algorithm ReReach$(B, A, n, \{(A, C_1), (C_1, C_2), \ldots, (C_{n-1}, B)\}$ will be called.*

Proof. Suppose that there is an attack-path from A to B, along the attacks $(A, C_1), (C_1, C_2), \ldots, (C_{n-1}, B) \in \mathsf{Att}$, for $C_1, \ldots, C_{n-1} \in \mathsf{Args}$. We proceed by induction on $n \geq 1$. Since $B \in \mathsf{Args}$, ReReach$(B, B, 0, \emptyset)$ will be called at Line 7 of the algorithm.

- **If $n = 1$:** then $(A, B) \in \mathsf{Att}$. At this point $\mathsf{Visited}$ is still empty, hence the **for**-loop at Line 2 of Procedure ReReach will be run for A. At Line 5, $\mathsf{Visited}$ becomes $\{(A, B)\}$ and at Line 6, ReReach$(B, A, 1, \{(A, B)\})$ is called.

Suppose the statement holds for values of n up to $k \geq 1$. Then:

- **If $n = k+1$:** then there are $C_1, \ldots, C_k$ such that there is an attack-path from A to B along the attacks $(A, C_1), \ldots, (C_k, B)$. And hence there is an attack-path from C_1 to B along the attacks $(C_1, C_2), \ldots, (C_k, B)$ of length k. By induction hypothesis, ReReach$(B, C_1, k, \{(C_1, C_2), \ldots, (C_k, B)\}$ is called during the run of the algorithm.

  Since $(A, C_1) \in \mathsf{Att}$, A is one of the arguments considered during this call. By assumption $(A, C_1), \ldots, (C_k, B)$ is an attack-path from A to B, therefore no attack appears twice. Hence $(A, C_1) \notin \{(C_1, C_2), \ldots, (C_k, B)\}$. Then $\mathsf{Visited}$ will be updated with (A, C_1) at Line 5 and at Line 6 ReReach$(B, A, k + 1, \{(A, C_1), (C_1, C_2), \ldots, (C_k, B)\})$ is called.

This shows that for any n, if there is an attack-path from A to B along the attacks in S, ReReach(B, A, n, S) will be called. □

The next proposition shows that Algorithm 1 is complete.

Proposition 2. *If there is an attack-path from A to B of length n then $n \in \mathsf{Dist}(A, B)$.*

Proof. Suppose that there is an attack-path from A to B of length n. We proceed again by induction on n.

- **If $n = 0$:** then by Remark 1 $A = B$. By Line 3 of Algorithm 1, $0 \in \mathsf{Dist}(A, A)$ and $0 \in \mathsf{Dist}(B, B)$.
- **If $n = 1$:** then by Remark 1 $(A, B) \in \mathsf{Att}$. By Line 4 of Procedure ReReach, $1 \in \mathsf{Dist}(A, B)$.

Suppose that the statement holds for n up to $k \geq 1$.

- **If $n = k+1$:** then there are $C_1, \dots, C_{k+1} \in \mathsf{Args}$, such that $A = C_1$, $B = C_{k+1}$, $(C_1, C_2), \dots, (C_k, C_{k+1}) \in \mathsf{Att}$ and there are no $1 \leq i, j \leq k$ such that $i \neq j$ and $(C_i, C_{i+1}) = (C_j, C_{j+1})$ (i.e., the attack-path does not follow an attack twice). Note that for any $2 \leq i, j \leq k+1$ such that $i \leq j$, the corresponding subset of attacks is an attack-path from C_i to C_j. In particular, $(C_2, C_3), \dots, (C_k, B)$ is an attack-path from C_2 to B of length k.

  By induction hypothesis, $k \in \mathsf{Dist}(C_2, B)$. Since $k \geq 1$, $\mathsf{Dist}(C_2, B)$ was updated at Line 4 of Procedure ReReach during the ReReach$(B, C_3, k-1, \mathsf{S}')$ call of the procedure. By Lemma 2 it follows that $\mathsf{S}' = \{(C_3, C_4), \dots, (C_k, B)\}$. Then, at Line 5 $\mathsf{Visited}$ is updated with (C_2, C_3) and, at Line 6, ReReach$(B, C_2, k, \mathsf{S}' \cup \{(C_2, C_3)\})$ is called. Since $(C_1, C_2) \in \mathsf{Att}$ and since there is an attack-path from C_1 to C_{k+1} along the attacks of $\mathsf{S}' \cup \{(C_1, C_2), (C_2, C_3)\}$, $\mathsf{Dist}(A, B)$ will be updated at Line 4 with $k + 1$. □

In our paper we are interested in the distance between two arguments (since this determines whether the relation is an attack (the distance is odd) or a defense (the distance is even)), but also in the arguments from which an argument is reachable. Both are computed by Algorithm 1 and the next lemma shows the relation between the two.

Lemma 3. $\mathsf{Dist}(A, B) \neq \emptyset$ *iff* $A \in \mathsf{Reach}(B)$.

Proof. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an AF and let $A, B \in \mathsf{Args}$. Assume that Algorithm 1 was run on $\mathcal{AF}$. Consider both directions separately.

⇒ Suppose that $\mathsf{Dist}(A, B) \neq \emptyset$. This direction is shown by induction on the minimal value n in $\mathsf{Dist}(A, B)$.

  - **If $n = 0$:** then $\mathsf{Dist}(A, B)$ was updated at Line 3 of the algorithm (since at Line 4 of the procedure the addition is always more than 0) and thus $A = B$. By Line 3 again it follows that $A \in \mathsf{Reach}(A)$.
  - **If $n = 1$:** then $\mathsf{Dist}(A, B)$ was updated at Line 4 of the procedure during the first iteration of the **for**-loop, in which case $(A, B) \in \mathsf{Att}$. By Line 3 it follows that $A \in \mathsf{Reach}(B)$.

  Now suppose that the statement holds for n up to a value of $k \geq 1$.

  - **If $n = k+1$:** then $\mathsf{Dist}(A, B)$ was updated at Line 4 of Procedure ReReach. Therefore, during this run of ReReach, at Line 3, $\mathsf{Reach}(B)$ is updated with $\mathsf{Reach}(A)$. Note that by Line 3 of Algorithm 1 $A \in \mathsf{Reach}(A)$ and hence $A \in \mathsf{Reach}(B)$.

⇐ Now assume that $A \in \mathsf{Reach}(B)$. We consider three cases:

  - $A = B$, then $\mathsf{Reach}(B)$ was updated at Line 3 of Algorithm 1, such that $B \in \mathsf{Reach}(B)$ and $0 \in \mathsf{Dist}(A, B)$.
  - $\mathsf{Reach}(B)$ was updated at Line 3 of Procedure ReReach, with $\mathsf{Reach}(A)$. Then at Line 4, $\mathsf{Dist}(A, B)$ is updated with $n + 1$.

– $\mathsf{Reach}(B)$ was updated at Line 3 of Procedure ReReach, with $\mathsf{Reach}(C)$ and $A \in \mathsf{Reach}(C)$. Hence, by Proposition 1, there is an attack-path from A to C and there is an attack-path from C to B. If no attack in the path from C to B is used in the attack-path from A to C, the procedure will call all arguments in the attack-path from A to C until it reaches A. At which point $\mathsf{Dist}(A, B)$ will be updated.

  Suppose now that there is some $(D_1, D_2) \in \mathsf{Att}$, such that (D_1, D_2) appears in both paths. Then there is an attack-path from A to D_1 (along the attacks $(A, E_k), \ldots, (E_1, D_1)$, where $E_k, \ldots, E_1 \in \mathsf{Args}$) and there is an attack-path from D_1 to B. Without loss of generality, suppose that (D_1, D_2) is such that there is no attack (D_1', D_2') in the attack-path from A to D_1 that also appears in the attack-path from C to B (otherwise the described procedure has to be repeated). Since there is an attack-path from D_1 to B (say of length l_d), by Lemma 2, ReReach$(B, D_1, l_d, \mathsf{S})$ is called. By assumption $\{(A, E_k), \ldots, (E_1, D_1)\} \cap \mathsf{S} = \emptyset$. Hence, for each $i \in \{1, \ldots, k\}$, during the call for ReReach$(B, D_1, l_d, \mathsf{S})$, ReReach$(B, E_i, l_d + i, \mathsf{S} \cup \{(E_i, E_{i-1}), \ldots, (E_1, D_1)\})$ is called. At ReReach$(B, E_k, l_d + k, \mathsf{S} \cup \{(E_k, E_{k-1}), \ldots, (E_1, D_1)\})$, note that $(A, E_k) \notin \mathsf{S} \cup \{(E_k, E_{k-1}), \ldots, (E_1, D_1)\}$. Hence $\mathsf{Reach}(B)$ is updated with $\mathsf{Reach}(A)$ and $\mathsf{Dist}(A, B)$ is updated with $l_d + k + 1$. Therefore $\mathsf{Dist}(A, B) \neq \emptyset$.

This shows that, in any situation, if $A \in \mathsf{Reach}(B)$ then $\mathsf{Dist}(A, B) \neq \emptyset$. □

With the above results we have the proof of Theorem 1:

Proof. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an argumentation framework, $A, B \in \mathsf{Args}$ and suppose that Algorithm 1 was run on $\mathcal{AF}$. Then:

1. Soundness and completeness of the algorithm follows immediately by Propositions 1 and 2.
2. By Lemma 3 we know that $A \in \mathsf{Reach}(B)$ iff $\mathsf{Dist}(A, B) \neq \emptyset$ and by the soundness and completeness of the algorithm (i.e., the first item) it is known that $n \in \mathsf{Dist}(A, B)$ iff there is an atack-path from A to B. □

We now turn to the computational complexity of the algorithm. Note that the algorithm does not determine whether an argument is accepted or not. It is therefore important that the extensions have been determined before running the algorithm.

Theorem 2 (Computational complexity)**.** *Algorithm 1 runs in polynomial time. In particular the time complexity is* $\mathcal{O}(|\mathsf{Args}| \cdot |\mathsf{Att}|^2)$.

Proof. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an argumentation framework and suppose that Algorithm 1 was run on this framework. Then:

- The first **for**-call of the algorithm takes $|\mathsf{Args}|$ time.
- Procedure ReReach runs in $|\mathsf{Att}|^2$: from each attack at most all other attacks are visited exactly once ($|\mathsf{Att}|$) and at most $|\mathsf{Att}|$ attacks end in a single argument ($|\mathsf{Att}|$).
- The procedure is called $|\mathsf{Args}|$ times from Algorithm 1.

This gives a total of $|\mathsf{Args}| + |\mathsf{Args}| \cdot |\mathsf{Att}|^2$, assuming that $\mathsf{Att} \neq \emptyset$ (this is safe to assume since argumentation could be considered interesting only when there are attacks), $\mathcal{O}(|\mathsf{Args}| \cdot |\mathsf{Att}|^2)$. □

From algorithm to explanation

Algorithm 1 determines for each argument the set of arguments from which it is reachable, as well as the distance between the arguments. From this we can define several notions that will be used in the explanations. We will denote by $\mathsf{Reach}_{\text{odd}}$ [resp. $\mathsf{Reach}_{\text{even}}$] the arguments with odd [resp. even] distance to the considered argument (i.e., $\mathsf{Reach}_{\text{odd}}(A) = \{B \in \mathsf{Reach}(A) \mid \exists n \in \mathsf{Dist}(B, A) \text{ s.t. } n \text{ is odd}\}$ [resp. $\mathsf{Reach}_{\text{even}}(A) = \{B \in \mathsf{Reach}(A) \mid \exists n \in \mathsf{Dist}(B, A) \text{ s.t. } n \text{ is even}\}$].

The next definition shows how DefBy and NotDef, used as a first suggestion for $\mathbb{D}$ in the definition of the explanations, can be defined in terms of the notions calculated by the algorithm.

Definition 2. Let $\mathcal{AF} = \langle \mathsf{Args}, \mathsf{Att} \rangle$ be an AF, $A \in \mathsf{Args}$ and $\mathcal{E} \in \mathsf{Ext}_{\mathsf{sem}}(\mathcal{AF})$ an extension for some semantics sem. Suppose that Algorithm 1 was run on $\mathcal{AF}$. Then:

- $\mathsf{DefBy}(A) = \{B \in \mathsf{Reach}_{\mathrm{even}}(A)\}$ denotes the set of arguments in Args that (in)directly defend A;
- $\mathsf{DefBy}(A, \mathcal{E}) = \mathsf{DefBy}(A) \cap \mathcal{E}$ denotes the set of arguments that (in)directly defend A in $\mathcal{E}$;
- $\mathsf{NotDef}(A, \mathcal{E}) = \{B \in \mathsf{Reach}_{\mathrm{odd}}(A) \mid \mathcal{E} \cap \mathsf{Reach}_{\mathrm{odd}}(B) = \emptyset\}$, denotes the set of all (in)direct attackers of A for which no defense exists from $\mathcal{E}$.

Example 1. In the running example from the paper, for the argumentation framework $\mathcal{AF}(\mathrm{AT}_2)$ we have that $\mathsf{Reach}(B_3) = \{A_2, A_3, B_1, B_3\}$ and $\mathsf{Dist}(A_2, B_3) = \{1, 3, 5\}$; and, where $\mathcal{E} = \{A_1, B_1, B_2, B_3\}$, we still have that $\mathsf{DefBy}(B_3, \mathcal{E}) = \{B_1, B_3\}$, while $\mathsf{NotDef}(A_2, \mathcal{E}) = \{B_1\}$.

References

[1] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[2] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In Pietro Baroni, Dov Gabay, Massimiliano Giacomin, and Leon van der Torre, editors, *Handbook of Formal Argumentation*, pages 631–688. College Publications, 2018.