Imperial College
London

NOTES

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Machine Arguing

Author:
W (CID: your college-id number)

Date: November 24, 2019

1 Arguing

Argumentation involves dealing with inconsistencies among the beliefs of multiple agents. Argumentation provides principles techniques for handling inconsistency, and in particular, for extracting rationally justifiable positions from the inconsistent pool of arguments.

2 Abstract Argumentation

In Abstract Argumentation, the argumentation framework is defined as pair

$$AF = \langle AR, attacks \rangle$$

AR is a set of **arguments**, and $attacks \subseteq AR \times AR$ is a set of binary relation of AR. The binary tuple (a, b) means a **attacks** b. The system can be represented using a directed graph where vertices are arguments and edges indicate attacks.

Given these basic definitions, we can define various notions of when an argument is rationally justifiable. Each notion has their own advantages and disadvantages.

2.1 Extensions

A **position** $S \subseteq AR$ is simply a set of arguments.

A position S is **conflict free** if no member of S attacks any other members of S.

If an argument a is attacked by some other argument b, we say that a is **defended** by c if c attacks b. A position S is **mutually defensive** if every element in S that is attacked is defended by some element of S.

A position S is **admissible** if it is conflict free and mutually defensive.

Preferred extension

A preferred extension is the maximal admissible set in AF.

Maximal does not mean cardinality maximal. The set is maximal if it is impossible to add another argument into the set while still being admissible.

Stable extension

A conflict-free set S is a stable extension if and only if S attacks every argument which does not belong to S.

Complete extension

A admissible set S is a complete extension if and only if all arguments are acceptable with respect to S, belongs to S.

Grounded extension

A set S is a grounded extension if it is minimally complete

Ideal extension

A admissible set S is an deal extension if it is contained in every preferred set.

Semi Stable extension

Given a set of arguments S, we define S^+ as

$$S^+ = \{\alpha \in Args \mid S \; attacks \; \alpha\}$$

A set S is semi-stable iff S is complete and $S \cup S^+$ is maximal among all complete extensions.

Related admissible extension

Argument α **r-defends** argument β iff

- $\alpha = \beta$, or
- α attacks an argument γ that attacks β, or
- α r-defends an argument γ that r-defends β

A set S is related admissible is there is an argument $\alpha \in S$ such that all arguments in S r-defends α and S is admissible.

2.2 Argument Labelling

Argument labelling is a function that assigns each argument a label IN, OUT, or $UNDEC$ (undecided).

Labelled arguments is represented by a tuple

$$LabArg = \langle IN(Args), \; OUT(Args), \; UNDEC(Args) \rangle$$

To convert labelled arguments to an extension, we simple consider all arguments labelled IN.

To convert an extension S to labelled arguments,

$$LabArg = \langle S, \; S^+, \; Args \setminus (S \cup S^+) \rangle$$

Conflict free labelling

For every argument $\alpha \in Args$,

- if α is labelled IN then arguments attacking α cannot be labelled IN.

Admissible labelling

For every argument $\alpha \in Args$,

- if α is labelled IN then arguments attacking α are labelled OUT
- If α is labelled OUT then there must be an argument attacking α labelled IN

Complete labelling

For every argument $\alpha \in Args$,

- if α is not attacked it must be labelled IN
- α is labelled OUT iff there is an argument attacking α labelled IN
- α is labelled IN iff **all** arguments attacking α are labelled OUT

Preferred labelling

A complete labelling with maximal $IN(Args)$ or maximal $OUT(Args)$.

Maximal does not mean cardinality maximal. The set is maximal if it is impossible to add another argument into the set while still begin complete.

Grounded labelling

A complete labelling with minimal $IN(Args)$ or maximal $OUT(Args)$.

To construct a grounded labelling

1. All arguments not attacked are labelled IN
2. An argument is labelled OUT if it is attacked by an argument labelled IN
3. An argument is labelled IN iff all arguments attacking it are labelled OUT
4. Repeat (2) and (3) until nothing changed
5. Everything else are left $UNDEC$

Stable labelling

A complete labelling with $UNDEC(Args) = \emptyset$

Semi Stable labelling

A complete labelling with minimal $UNDEC(Args)$ or maximal $IN(Args) \cup OUT(Args)$

Ideal labelling

A maximal admissible labelling such that for **all** preferred labelling $PLab$:

- $IN(Args) \subseteq IN(PLab)$
- $OUT(Args) \subseteq OUT(PLab)$

2.3 Dispute Trees

Dispute tree is a way of generating a winning strategy for an argument. More formally, a dispute tree $\mathcal{T}$ for an argument a is defined as

- Every node in $\mathcal{T}$ is an argument, and assigned a label **proponent** or **opponent**.
- The root of $\mathcal{T}$ is the **proponent** node a
- Every **proponent** node p must include **all opponent** arguments that attacks p as children.
- Every **opponent** node o have **at most one proponent** argument that attacks o as child.

The set arguments labelled as **proponent** in $\mathcal{T}$ is the **defence set** of $\mathcal{T}$.

Admissible dispute tree

A dispute tree is admissible if every opponent node has exactly one child and no arguments are labelled both proponent and opponent.

If $\mathcal{T}$ is an admissible dispute tree then the defence set of $\mathcal{T}$ is also admissible.

Grounded dispute tree

A admissible tree is grounded if every opponent node has exactly one child and the tree is finite.

Ideal dispute tree

An admissible tree is **ideal** if no opponent node o in $\mathcal{T}$ can create an admissible tree with root o.

If $\mathcal{T}$ is an ideal dispute tree then the defence set of $\mathcal{T}$ is also ideal.

2.4 Answer Set Programming

We can map the Abstract Argument Framework to a logic programme such that the stable extension corresponds to the answer set.

2.4.1 Herbrand Model

Herbrand base of a logic programme P is the set of all atoms that appeared in P.

A Herbrand model is a subset of the Herbrand base that satisfies all rules in P. The Least Herbrand model is the smallest subset of Herbrand base that satisfies all rules in P.

2.4.2 Answer Set

Answer sets also called stable models provide a simple semantics for normal logic programs. Given a logic programme P and a set of atoms X, the reduct P^X is a set of clauses obtained from P as follow:

1. Delete any clause in P that has $not\ A$ in its body where $A \in X$
2. Delete every condition in the form $not\ A$ in the bodies of the remaining clauses

X is an answer set if the Least Herbrand Model of P^X coincides with X.

2.4.3 Answer Sets for Abstract Argument

Given an Abstract Argumentation Framework $\langle Args,\ attacks\rangle$, we can construct a logical programme P as follow:

1. For all $\alpha \in Args$, create rule $args(\alpha) \leftarrow$
2. For all $(\alpha, \beta) \in Attacks$, create rule $att(\alpha, \beta) \leftarrow$
3. For all instances of $X, Y \in Args$, create the rules:
   - $\bot \leftarrow in(X),\ in(Y),\ att(X, Y)$
   - $in(X) \leftarrow not\ out(X)$
   - $out(X) \leftarrow not\ in(X)$
   - $\bot \leftarrow out(X),\ not\ defeated(X)$
   - $defeated(X) \leftarrow in(Y),\ att(Y, X)$

We can solve for the answer sets of P. If a set S is an answer set, then the set of arguments $\{\alpha \in Args \mid in(\alpha) \in S\}$ is a stable extension.

2.5 Mining AA From Logic Programs

Given a logic program P, we can create an AA Framework, $\langle AR,\ attacks\rangle$. The framework is defined as follow:

- Each argument in AR is in the form: $P \cup \Delta \vdash \alpha$.
  - Δ can be an empty set or a set of Negation As Failure literals ($not\ \beta$)
  - α is a single literal in the form $not\ \beta$ or β
- An argument $P \cup \Delta_1 \vdash x$ attacks an argument $P \cup \Delta_2 \vdash y$ if the literal $not\ x \in \Delta_2$.

Given a logic program P and the AA framework mined from P, AA_p.

- If $\{x \mid P \cup \_ \vdash x \in S\}$ is an answer set of P, then S is a stable extension of AA_p
- If $\{P \cup \Delta \vdash x \mid \Delta \subseteq \Delta_M\}$ is a stable extension of AA_p, where $\Delta_M = \{not\ x \mid x \notin M\}$, then M is an answer set of P.

3 Bipolar Abstract Argumentation

Bipolar Abstract Argumentation builds on top of Abstract Argumentation by adding a support relation between arguments. The bipolar abstract argumentation framework is defined as

$$AF = \langle AR,\ attacks,\ supports \rangle$$

Where AR and $attacks$ is exactly the same as AA framework. $supports \subseteq AR \times AR$ is a set of binary relation of AR. The binary tuple (a, b) means a **supports** b. In the directed graph, we use double line edges to indicate supports.

A set of arguments $A \subseteq AR$ supports an argument β is there is one argument $\alpha \in A$ that **directly or indirectly** supports β.

3.1 Mapping BAA to AA Framework

We can map BAA to AA by removing support relations and adding two types of attack relations: supported attacks and super-mediated attacks.

Supported attacks means that supporting arguments attack everything their supported arguments can attack. An argument $\alpha\ attacks_{sup}\ \beta$ if α directly or indirectly supports γ and γ attacks β.

Super-mediated attacks means that an argument α attack all supporting arguments of β if α attacks or support attacks β. An argument $\alpha\ attacks_{s-med}\ \beta$ if β directly or indirectly supports γ, and α attacks or support attacks γ.

We can map BAA to AA by extending the set of attacks with supported attacks and super-mediated attacks.

$$\langle AR,\ attacks,\ supports \rangle \rightarrow \langle AR,\ attacks \cup attacks_{sup} \cup attacks_{s-med} \rangle$$

The extensions of BAA framework is d-admissible/d-preferred/d-stable if the transformed framework is admissible/preferred/stable.

3.2 Quantitative Argumentation Debate (QuAD)

In previous sections, there is no way to define the strength of each attacking and supporting arguments. QuAD is a special kind of BAA that assigns a score for each argument. The QuAD framework is defined as

$$\langle \mathcal{A},\ \mathcal{C},\ \mathcal{P},\ \mathcal{R},\ \mathcal{BS} \rangle$$

Where $\mathcal{A}$ is a set of answer arguments. $\mathcal{C}$ is a set of con arguments. $\mathcal{P}$ is a set of pro arguments. $\mathcal{R} \subseteq (\mathcal{C} \cup \mathcal{P}) \times (\mathcal{A} \cup \mathcal{C} \cup \mathcal{P})$ are **acyclic** binary relations between arguments. $\mathcal{BS} : (\mathcal{A} \cup \mathcal{C} \cup \mathcal{P}) \rightarrow [0, 1]$ is a function that assigns each argument a base score.

Each of the arguments α starts with base score. The final score is calculated as follow:

1. Calculate the aggregate score of all attackers that directly attacks α. The set of attackers is $\{\beta \in \mathcal{C} \mid (\beta, \alpha) \in \mathcal{R}\}$.

2. Calculate the aggregate score of all supporters that directly supports α. The set of supporters is $\{\beta \in \mathcal{P} \mid (\beta, \alpha) \in \mathcal{R}\}$

3. Combine the two aggregated values to find the new score for α

Given that the relation $\mathcal{R}$ is acyclic, we can form a tree from the arguments. Starting from the bottom, we recursively calculate the score for each argument until we reach the root node.

3.2.1 Aggregate Function

This function aggregates the value of arguments in a set $\mathcal{S} = \{v_1, v_2, \ldots, v_n\}$and returns a final score $\mathcal{F}(\mathcal{S})$.

$$
\begin{aligned}
&\mathcal{F}(\{\}) = 0 \\
&\mathcal{F}(\{v_1\}) = v_1 \\
&\mathcal{F}(\{v_1, v_2\}) = v_1 + v_2 - v_1 v_2 \\
&\mathcal{F}(\{v_1, \ldots, v_n\}) = \mathcal{F}(v_1, v_2, \ldots, v_{n-1}) + v_n - \mathcal{F}(v_1, v_2, \ldots, v_{n-1}) v_n
\end{aligned}
$$

3.2.2 Combination Function

The combination function takes the base score of an argument v_0, the aggregated values of the attackers v_a and the aggregated values of the supporters v_s and return the new score for the argument.

$$
c(v_0, v_a, v_s) = \begin{cases} v_0 - v_0 |v_s - v_a|, & v_a \geq v_s \\ v_0 + (1 - v_0)|v_s - v_a|, & otherwise \end{cases}
$$

4 Assumptions Based Argumentation

An Assumption Based Argumentation framework is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{} \rangle$. Where $\mathcal{L}$ is the language of the framework and $\mathcal{R}$ is the set of rules. $\mathcal{A} \subseteq \mathcal{L}$ is a non-empty set of assumptions. $\overline{}$ is a total mapping that maps each assumption $a \in \mathcal{A}$ to its contrary $\bar{a} \in \mathcal{L}$.

In this section we consider **flat** framework meaning assumptions cannot appear at the head of each rule.

4.1 Arguments

Arguments in ABA framework are deductions of the form

$$A \vdash^{R} \sigma$$

Where $A \subseteq \mathcal{A}$ is a set of assumptions (can be empty set). $R \subseteq \mathcal{R}$ is a set of rules (can be empty set). $\sigma \in \mathcal{L}$ is a claim that can be deduced from the assumptions A and rules R.

All assumptions in A must be relevant in deducing σ for it to be a valid argument. Also, all assumptions deduce itself.

$$\forall a \in \mathcal{A}.\ \{a\} \vdash a$$

An argument $A \vdash^{R} \sigma$ is

- **assumption subset minimal** if there is no other $A' \vdash^{R} \sigma$ where $A' \subset A$
- **assumption cardinality minimal** if there is no other $A' \vdash^{R} \sigma$ where $|A'| < |A|$
- **rule subset minimal** if there is no other $A \vdash^{R'} \sigma$ where $R' \subset R$
- **rule cardinality minimal** if there is no other $A \vdash^{R'} \sigma$ where $|R'| < |R|$

4.2 Attacks

4.2.1 Attack Between Arguments

In ABA, an argument $A_1 \vdash \sigma_1$ attacks argument $A_2 \vdash \sigma_2$ if the contrary of σ_1, $\overline{\sigma_1} \in A_2$.

4.2.2 Attack Between Set of Assumptions

A set of assumptions A attacks a set of assumptions A' iff an argument supported by A attacks an argument supported by A'.

If an argument α attacks an argument α', then the set of assumptions supporting α attacks the set of assumptions supporting α'.

4.3 ABA Semantics

A extension semantics (admissible, complete, etc.) for ABA arguments is exactly the same as AA framework at the argument level. We can also talk about extension semantics from an assumption level. A set of assumption is

- **admissible** if it attacks all sets of assumptions that attacks it
- **complete** if it is admissible and contains all assumptions it defends.
- **grounded** if it is minimally complete

- **stable** if it attacks everything not in the set

A set of assumptions A attacks assumption a if A attacks $\{a\}$.

If a set of argument A is admissible/complete/grounded etc, then the union of all set of assumptions supporting the arguments in A is admissible/complete/grounded etc.

If a set of assumptions A is admissible/complete/grounded etc, then the union of all arguments supported by any subset of A is admissible/complete/grounded etc.

4.4 Mapping AA to ABA

Given a AA framework $\langle AR, attacks \rangle$ corresponds to the ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ where

$$\begin{aligned}
\mathcal{L} &= AR \cup \{\alpha^c \mid \alpha \in AR\} \\
\mathcal{R} &= \{\beta^c \leftarrow \alpha \mid (\alpha, \beta) \in attacks\} \\
\mathcal{A} &= AR \\
\overline{\alpha} &= \alpha^c, \ \forall \alpha \in \mathcal{A}
\end{aligned}$$

4.5 Dispute Derivations

Instead of mapping ABA into AA then compute semantics in AA framework, the computation of ABA can be abstracted away as disputes between a **proponent** and an **opponent**. These disputes are defined as a sequence of tuples called **dispute derivations**

4.5.1 Data Structures

The dispute derivations include the following components:

- Proponent to do list $\mathcal{P}$
- Opponent to do list $\mathcal{O}$
- Defense set constructed by the proponent $\mathcal{D}$
- Culprit set constructed by the opponent $\mathcal{C}$
- The set of arguments constructed and dealt with $Args$
- The attacks between $Args$, Att

Once successfully terminated, the $Args$ and Att components will amount to a dialectical tree

Elements of $\mathcal{P}$, $\mathcal{O}$ and $Args$ may be **potential arguments** rather than actual arguments. Potential arguments are intermediate steps in the construction of actual arguments (or failed attempts). Potential arguments have the form

$$\mathcal{A} \vdash_S \sigma$$

Where $\alpha \in \mathcal{A}$ are marked assumptions and $\alpha \in S$ are unmarked sentences/assumptions.

4.5.2 Computation Steps

Given a flat ABA framework and we want to determine whether a sentence σ is admissible/grounded. We start with the dispute derivation

$$\mathcal{P} = \{\{\} \vdash_{\{\sigma\}} \sigma\} \qquad \mathcal{O} = \{\} \qquad \mathcal{D} = \{\sigma\} \cap \mathcal{A} \qquad \mathcal{C} = \{\}$$

While $\mathcal{P}$ and $\mathcal{O}$ are not empty, we pick one of the to do list to work on. The aim of the proponent is to attack assumptions in the culprit set $\mathcal{C}$ while the aim of the opponent is to attack assumptions in the defense set $\mathcal{D}$.

The opponent has to find **all** rules attacking assumptions in $\mathcal{D}$, but the proponent only has to find one rule that attacks each assumption in $\mathcal{C}$.

The dispute derivations are successful if they are finite and $\mathcal{P}$ and $\mathcal{O}$ becomes empty.

4.5.3 Filtering

Dispute derivations incorporate various kinds of filtering, some common to all types of dispute derivations, some different for computing different semantics. Filtering common to all types of dispute derivations (grounded and admissible) include:

- Defenses by Culprits: The proponent should avoid choosing rules that uses assumptions already in $\mathcal{C}$
- Culprits by Defenses: The opponent should avoid choosing rules that uses assumptions already in $\mathcal{D}$

The following filtering only applies the admissible dispute derivations:

- Culprits by Culprits: Avoid re-computation of assumptions from opponent already in $\mathcal{C}$
- Defenses by Defenses: Avoid re-computation of assumptions from proponent already in $\mathcal{D}$

4.6 Non-Flat ABA

In non-flat ABA, an assumption a can appear at the head of a rule. In non-flat ABA, the assumption set A in each arguments $A \vdash^R \sigma$ must be **closed**, meaning it must include all assumptions deducible from the set.

4.6.1 Mapping BAA to Non-Flat ABA

Given a BAA framework $\langle AR, attacks, supports\rangle$ corresponds to the ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ }\rangle$ where

$$\mathcal{L} = AR \cup \{\alpha^c \mid \alpha \in AR\}$$
$$\mathcal{R} = \{\beta^c \leftarrow \alpha \mid (\alpha, \beta) \in attacks\} \cup \{\beta \leftarrow \alpha \mid (\alpha, \beta) \in supports\}$$
$$\mathcal{A} = AR$$
$$\overline{\alpha} = \alpha^c,\ \forall \alpha \in \mathcal{A}$$

An extension E is d-preferred in the BAA framework if E is a preferred extension in the ABA framework.

An extension E is d-stable in the BAA framework if E is a set stable extension in the ABA framework, meaning E is closed, conflict free and attacks the closure of every assumption not in the extension.

5 Argumentation Preferences

5.1 Abstract Argumentation with Preferences

An AA framework with preference is a framework $\langle Args, Attacks, <\rangle$, where $<$ is a transitive binary relation on $Args$. Given two arguments $\alpha, \beta \in Args$, $\alpha < \beta$ means β is preferred over α.

5.1.1 Deleting Attacks

One way to handle preferences is to remove attacks from α to β if $\alpha < \beta$. More formally, an AAP framework yields an AA framework $\langle Args, defeat\rangle$ where $(\alpha, \beta) \in defeat$ if $(\alpha, \beta) \in attacks$ and $\alpha \not< \beta$.

5.1.2 Reversing Attacks

Removing attacks can be problematic because extensions may no longer be conflict free. One way to preserve the conflict between arguments is to reverse attacks from α to β if $(\alpha, \beta) \in attacks$ and $\alpha < \beta$. More formally, an AAP frameworks yields an AA framework $\langle Args, attacks'\rangle$ where

1. $(\alpha, \beta) \in attacks'$ if $(\alpha, \beta) \in attacks$ and $\alpha \not< \beta$
2. $(\alpha, \beta) \in attacks'$ if $(\beta, \alpha) \in attacks$ and $\beta < \alpha$

5.1.3 Value Based Argumentation Framework

The Value Based Argumentation Framework $\langle Args, attacks, \mathcal{V}, <\rangle$ defines a set of values $\mathcal{V}$ and a function that assigns each argument $\alpha \in Args$ to a value $v \in \mathcal{V}$. $<$ is a transitive preference relation on $\mathcal{V}$. Therefore when comparing preferences of arguments, we compare their values.

5.2 Selecting Among Extensions

Alternatively we can define a preference relation on extensions, $\prec$. And then choose the most preferred extension after we compute all the (grounded, stable, etc...) extensions.

6 Case-Based Reasoning

Case-Based Reasoning is used for making decisions on new queries by analysing similar past instances. This example in this chapter decides whether a vehicle is allowed in the park given the following past cases.

- A bicycle is allowed in the park $(\{b\}, +)$
- A motorised bike is not allowed in the park $(\{b, m\}, -)$
- An ambulance is allowed in the park when there is health emergency $(\{a, h, m\}, +)$
- A vehicle is allowed in the park where there is health emergency $(\{h\}, +)$

Using the past cases, we want to determine is whether a motorised bike is allowed in the park when there is a health emergency. $(\{b, m, h\}, ?)$

To solve the problem, we first define a default outcome $(\{\},+)$ or $(\{\}, -)$. In this case we choose $(\{\}, -)$ as the default.

The arguments are all the past cases and the default case.

An argument α attacks another argument β if it is more specific $\beta \subset \alpha$ (there are no other arguments γ, such that $\beta \subset \gamma \subset \alpha$), and has a different outcome. From the above examples, the attacks are

$$(\{b\}, +) \rightarrow (\{\}, -)$$
$$(\{h\}, +) \rightarrow (\{\}, -)$$
$$(\{b, m\}, -) \rightarrow (\{b\}, +)$$

We then determine the new case by adding attacks from the new case to past cases. The new case attacks an argument α if α contains an atom not in the new case.

Finally, we find the grounded extension. If the default case is in the grounded extension, then the outcome is the default outcome. Otherwise, it is the opposite of default outcome.