A principled approach to the implementation of argumentation models

Bas van Gijzel and Henrik Nilsson
bmv@cs.nott.ac.uk

University of Nottingham

September 10, 2014
Outline

1. Motivation of the methodology
2. Overview of the specifications/implementations/formalisations
3. Examples of our approach
   Implementation and verification using FP Theorem proving
4. Conclusions
Outline

1 Motivation of the methodology

2 Overview of the specifications/implementations/formalisations

3 Examples of our approach
   Implementation and verification using FP
   Theorem proving

4 Conclusions
How to implement an argumentation model

Two main ways to implement an argumentation model:
How to implement an argumentation model

Two main ways to implement an argumentation model:

- Directly implement it into your favourite programming language;
How to implement an argumentation model

Two main ways to implement an argumentation model:

• **Directly implement** it into your favourite programming language;

• **Implement the translation**, given a formal relation to another (implemented) **simpler** model (e.g. to Dung’s AFs).
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

• A decent amount of well-documented and open source applications.
• Recent efforts to optimise the evaluation of AFs/ADFs using:
  • SAT-solvers
  • Answer-set programming
• A decent amount of other abstract models have been implemented through encodings into AFs.
  For instance ASPARTIX, DIAMOND and ArgSemSAT:
  See: http://www.dbai.tuwien.ac.at/research/project/argumentation/
  https://isysrv.informatik.uni-leipzig.de/diamond
  http://tiny.cc/argsemsat
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

• A decent amount of well-documented and open source applications.
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

• A decent amount of well-documented and open source applications.
• Recent efforts to optimise the evaluation of AFs/ADFs using:

  • SAT-solvers
  • Answer-set programming

Other abstract models have been implemented through encodings into AFs: For instance ASPARTIX, DIAMOND and ArgSemSAT:

See:
http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/
https://isysrv.informatik.uni-leipzig.de/diamond
http://tiny.cc/argsemsat
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

• A decent amount of well-documented and open source applications.
• Recent efforts to optimise the evaluation of AFs/ADFs using:
  • SAT-solvers
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

• A decent amount of well-documented and open source applications.
• Recent efforts to optimise the evaluation of AFs/ADFs using:
  • SAT-solvers
  • Answer-set programming

For instance ASPARTIX, DIAMOND and ArgSemSAT:
See: http://www.dbai.tuwien.ac.at/research/project/argumentation/
http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/
https://isysrv.informatik.uni-leipzig.de/diamond
http://tiny.cc/argsemsat
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

- A decent amount of well-documented and open source applications.
- Recent efforts to optimise the evaluation of AFs/ADFs using:
  - SAT-solvers
  - Answer-set programming
- A decent amount of other abstract models have been implemented through encodings into AFs.
Implementations of abstract models

Status of implementations for abstract models, e.g. Dung’s AFs:

- A decent amount of well-documented and open source applications.
- Recent efforts to optimise the evaluation of AFs/ADFs using:
  - SAT-solvers
  - Answer-set programming
- A decent amount of other abstract models have been implemented through encodings into AFs.

For instance ASPARTIX, DIAMOND and ArgSemSAT:

See:
http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/
https://isysrv.informatik.uni-leipzig.de/diamond
http://tiny.cc/argsemsat
Implementations of structured models

Status of implementations for structured models: ASPIC
Implementations of structured models

Status of implementations for structured models: ASPIC
Implementations of structured models

Status of implementations for structured models: ASPIC

The same holds for various other models/projects.
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...

See: http://carneades.github.io/carneades/Carneades/
http://www.arg.dundee.ac.uk/toast/
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  • **Effort** spent is practically **lost**,
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  • Effort spent is practically lost,
  • Implementation techniques are non-reproducible.
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  • Effort spent is practically lost,
  • Implementation techniques are non-reproducible.
• A few mature implementations (Carneades, TOAST, etc.), but:
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  • Effort spent is practically lost,
  • Implementation techniques are non-reproducible.

• A few mature implementations (Carneades, TOAST, etc.), but:
  • Again often closed source,
Implementations of structured models

- Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  - Effort spent is practically lost,
  - Implementation techniques are non-reproducible.
- A few mature implementations (Carneades, TOAST, etc.), but:
  - Again often closed source,
  - Often not directly related to the actual mathematical model.
Implementations of structured models

• Significant amount of implementations are unavailable and closed source: ASPIC (EU), ArguGrid, and many more...
  • Effort spent is practically lost,
  • Implementation techniques are non-reproducible.
• A few mature implementations (Carneades, TOAST, etc.), but:
  • Again often closed source,
  • Often not directly related to the actual mathematical model.

See:
http://carneades.github.io/carneades/Carneades/
http://www.arg.dundee.ac.uk/toast/
Implementations of translations

Implemented translations are even more rare:
Implementations of translations

Implemented translations are even more rare:

- Situation is improving for abstract argumentation (Sylwia Polberg and others);
Implementations of translations

Implemented translations are even more rare:

- Situation is improving for abstract argumentation (Sylwia Polberg and others);
- For existing translations from structured models to AFs, however again a lack of implementations;
Implementations of translations

Implementations of translations are even more rare:

- Situation is improving for abstract argumentation (Sylwia Polberg and others);
- For existing translations from structured models to AFs, however again a lack of implementations;
- Additionally:
Implementations of translations

Implemented translations are even more rare:

• Situation is improving for abstract argumentation (Sylwia Polberg and others);

• For existing translations from structured models to AFs, however again a lack of implementations;

• Additionally:
  • Translations are complex and relatively ad-hoc
Implementations of translations

Implementated translations are even more rare:

• Situation is improving for abstract argumentation (Sylwia Polberg and others);

• For existing translations from structured models to AFs, however again a lack of implementations;

• Additionally:
  • Translations are complex and relatively ad-hoc
  • Proofs of correctness are complex
Problem statement

We need:

• publicly available and reproducible implementations/implementation methods;
Problem statement

We need:

• **publicly available and reproducible implementations/implementation methods**;

• **further verification or even complete mechanical formalisation of translations/proofs**
A principled approach to solving this problem (1)

Abstract argumentation can be implemented using:

- **Logic programming**, formally related to Dung’s argumentation frameworks
- **Answer set programming**, a natural candidate for calculating semantics (extensions)
A principled approach to solving this problem (1)

Abstract argumentation can be implemented using:

- **Logic programming**, formally related to Dung’s argumentation frameworks
- **Answer set programming**, a natural candidate for calculating semantics (extensions)

Structured argumentation models need a similar language:
A principled approach to solving this problem (1)

Abstract argumentation can be implemented using:

- **Logic programming**, formally related to Dung’s argumentation frameworks
- **Answer set programming**, a natural candidate for calculating semantics (extensions)

Structured argumentation models need a similar language:

- Able to express more **general mathematics**
- **Data structures**
A principled approach to solving this problem (1)

Abstract argumentation can be implemented using:

- Logic programming, formally related to Dung’s argumentation frameworks
- Answer set programming, a natural candidate for calculating semantics (extensions)

Structured argumentation models need a similar language:

- Able to express more general mathematics
- Data structures
- Able to easily verify or prove properties
A principled approach to solving this problem (1)

Abstract argumentation can be implemented using:

- **Logic programming**, formally related to Dung’s argumentation frameworks
- **Answer set programming**, a natural candidate for calculating semantics (extensions)

Structured argumentation models need a similar language:

- Able to express more general mathematics
- Data structures
- Able to easily verify or prove properties

My suggestion: **functional programming**, in specific Haskell/Agda.
A principled approach to solving this problem (2)

• Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC⁺) in Haskell
A principled approach to solving this problem (2)

• Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC⁺) in Haskell
  • In a tutorial-like fashion;
A principled approach to solving this problem (2)

• Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC⁺) in Haskell
  • In a tutorial-like fashion;
  • Close to the actual mathematical definitions;
A principled approach to solving this problem (2)

- Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC+) in Haskell
  - In a tutorial-like fashion;
  - Close to the actual mathematical definitions;
  - With output usable by other existing efficient implementations.
A principled approach to solving this problem (2)

- Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC+) in Haskell
  - In a tutorial-like fashion;
  - Close to the actual mathematical definitions;
  - With output usable by other existing efficient implementations.

- In the same fashion: implement a translation
A principled approach to solving this problem (2)

• Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC+) in Haskell
  • In a tutorial-like fashion;
  • Close to the actual mathematical definitions;
  • With output usable by other existing efficient implementations.

• In the same fashion: implement a translation
• Provide quick verification by implementation of properties
A principled approach to solving this problem (2)

- Provide implementation of Dung’s AFs and some structured models (Carneades, ASPIC+) in Haskell
  - In a tutorial-like fashion;
  - Close to the actual mathematical definitions;
  - With output usable by other existing efficient implementations.

- In the same fashion: implement a translation
- Provide quick verification by implementation of properties

Result: a methodology that allows for quick and clean implementing and initial testing of properties.
A principled approach to solving this problem (3)

Additionally:

• Provide mechanical formalisation of implementations and translation, using the theorem prover, Agda;
A principled approach to solving this problem (3)

Additionally:

• Provide mechanical formalisation of implementations and translation, using the theorem prover, Agda;

• Using a theorem prover based on the Curry-Howard correspondence:
A principled approach to solving this problem (3)

Additionally:

• Provide mechanical formalisation of implementations and translation, using the theorem prover, Agda;
• Using a theorem prover based on the Curry-Howard correspondence:
  • Types with accompanying implementations (functions), correspond to theorems with accompanying proofs;
A principled approach to solving this problem (3)

Additionally:

• Provide mechanical formalisation of implementations and translation, using the theorem prover, Agda;

• Using a theorem prover based on the Curry-Howard correspondence:
  • Types with accompanying implementations (functions), correspond to theorems with accompanying proofs;
  • Meaning we get a mechanically verified formalisation and implementation in one.
Additionally:

- Provide **mechanical formalisation** of implementations and translation, using the **theorem prover, Agda**;
- Using a **theorem prover** based on the Curry-Howard correspondence:
  - **Types** with accompanying **implementations (functions)**, correspond to **theorems** with accompanying **proofs**;
  - Meaning we get a **mechanically verified formalisation** and **implementation** in one.

Result: a **verified** pipeline to **translate** models to an **efficiently implemented model**.
A principled approach to solving this problem (4)

Additionally:

\[1\text{See http://www.cs.nott.ac.uk/~bmv/COMMA/}\]
A principled approach to solving this problem (4)

Additionally:

- All Haskell code will or has been published on Hackage/GitHub under an open source license, with:

\[\text{See } \text{http://www.cs.nott.ac.uk/~bmv/COMMA/}\]
A principled approach to solving this problem (4)

Additionally:

• All Haskell code will or has been published on Hackage/GitHub under an open source license, with:
  • accompanying installation instructions;
  • elaborate examples;
  • and appropriate documentation.

---

¹See http://www.cs.nott.ac.uk/~bmv/COMMA/
A principled approach to solving this problem (4)

Additionally:

• All Haskell code will or has been published on Hackage/GitHub under an open source license, with:
  • accompanying installation instructions;
  • elaborate examples;
  • and appropriate documentation.

• Proofs in Agda are open source and publicly available

\[\textsuperscript{1}\text{See } \text{http://www.cs.nott.ac.uk/~bmv/COMMA/}\]
A principled approach to solving this problem (4)

Additionally:

- All Haskell code will or has been published on Hackage/GitHub under an open source license, with:
  - accompanying installation instructions;
  - elaborate examples;
  - and appropriate documentation.
- Proofs in Agda are open source and publicly available.

I hope this helps to tackle the problem of unavailable implementations and lost programming methodology.¹

¹See http://www.cs.nott.ac.uk/~bmv/COMMA/
Outline

1. Motivation of the methodology
2. Overview of the specifications/implementations/formalisations
3. Examples of our approach
   Implementation and verification using FP
   Theorem proving
4. Conclusions
Schematic overview of the work done

Carneades → translation → Dung’s AFs

- Formalised Carneades
- Formalised Dung’s AFs

Formalisation → translation → formalisation

15/26
Outline

1 Motivation of the methodology

2 Overview of the specifications/implementations/formalisations

3 Examples of our approach
   Implementation and verification using FP Theorem proving

4 Conclusions
Outline

1 Motivation of the methodology

2 Overview of the specifications/implementations/formalisations

3 Examples of our approach
   Implementation and verification using FP
   Theorem proving

4 Conclusions
Implementation of functions

Definition (Conflict-free)
Given $AF = \langle \text{Args}, \text{Atk} \rangle$. 

conflictFree :: Eq arg ⇒ DungAF arg → [arg] → Bool
conflictFree (AF atk) s = null [(a, b) | (a, b) ← atk, a ∈ s, b ∈ s]
Implementation of functions

Definition (Conflict-free)
Given $AF = \langle \text{Args}, \text{Atk} \rangle$.

A set $S \subseteq \text{Args}$ of arguments is called conflict-free iff
Definition (Conflict-free)

Given \( AF = \langle \text{Args}, \text{Atk} \rangle \).

A set \( S \subseteq \text{Args} \) of arguments is called conflict-free iff there is no \( A, B \in S \) such that \( (A, B) \in \text{Atk} \).
Definition (Conflict-free)

Given $AF = \langle \text{Args}, \text{Atk} \rangle$.

A set $S \subseteq \text{Args}$ of arguments is called conflict-free iff there is no $A, B \in S$ such that $(A, B) \in \text{Atk}$.

$$\text{conflictFree} :: \text{Eq arg} \Rightarrow \text{DungAF arg} \rightarrow [\text{arg}] \rightarrow \text{Bool}$$

$$\text{conflictFree} \ ((\text{AF - atk}) \ s) = \text{null } [(a, b) | (a, b) \leftarrow \text{atk}, a \in s, b \in s ]$$
Implementation of properties

Theorem (Correspondence of applicability)

Let $C$ be a carneades argument evaluation structure, $\langle \text{arguments, audience, standard} \rangle$, $\mathcal{L}_{\text{CAES}}$ the propositional language used and let the argumentation framework corresponding to $C$ be $\text{AF}$. 

$$
\text{corApp} :: \mathcal{L}_{\text{CAES}} \rightarrow \text{Bool}
$$
Implementation of properties

Theorem (Correspondence of applicability)

Let $C$ be a carneades argument evaluation structure, $\langle \text{arguments, audience, standard} \rangle$, $\mathcal{L}_{\text{CAES}}$ the propositional language used and let the argumentation framework corresponding to $C$ be $AF$. Then the following holds: An argument $a \in \text{arguments}$ is applicable in $C$ iff there is an argument contained in the complete extension of $AF$ with the corresponding conclusion $\text{arg}_a$ in an $AF$. 
Implementation of properties

Theorem (Correspondence of applicability)

Let $C$ be a carneades argument evaluation structure, 
$\langle \text{arguments, audience, standard} \rangle$, $\mathcal{L}_{\text{CAES}}$ the propositional language used and let the argumentation framework corresponding to $C$ be $\text{AF}$. Then the following holds: An argument $a \in \text{arguments}$ is applicable in $C$ iff there is an argument contained in the complete extension of $\text{AF}$ with the corresponding conclusion $\text{arg}_a$ in an $\text{AF}$.

$$\text{corApp} :: \text{CAES} \rightarrow \text{Bool}$$

$$\text{corApp caes@(CAES (argSet, _, _))} =$$

$$\text{let } \text{transCAES} = \text{translate caes}$$

$$\text{appArgs} = \text{filter ('applicable' 'caes)} (\text{getAllArgs argSet})$$

$$\text{transArgs} = \text{stripRight (groundedExt transCAES)}$$

$$\text{in fromList appArgs }\equiv\text{ fromList transArgs}$$
Outline

1 Motivation of the methodology

2 Overview of the specifications/implementations/formalisations

3 Examples of our approach
   Implementation and verification using FP
   Theorem proving

4 Conclusions
Grounded labelling takes three lists of arguments:
Grounded labelling takes three lists of arguments:

- Ins
- Outs
- Unlabelled arguments (initially all)
Grounded labelling takes three lists of arguments:

- Ins
- Outs
- Unlabelled arguments (initially all)

```
groundedList : \{ A : Set \} \to
    List A \to List A \to List A \to
    DungAF A \to List (A \times Status)
```
Code example of the formalisation of Dung’s AFs (2)

Grounded labelling of an AF:
Grounded labelling of an AF:

\[
grounded' : \{ A : \text{Set} \} \rightarrow \{ m, n, o : \mathbb{N} \} \rightarrow \\
\quad \cdots \rightarrow \\
\quad \text{Vec} \ A \ m \rightarrow \text{Vec} \ A \ n \rightarrow \text{Vec} \ A \ o \rightarrow \\
\quad \text{DungAF} \ A \rightarrow \text{Vec} \ (A \times \text{Status}) \\
\quad (m + n + o)
\]
Grounded labelling of an AF:

\[ \text{grounded}' : \{ A : \text{Set} \} \to \{ m n o : \mathbb{N} \} \to \]
\[ (\sum \mathbb{N} \lambda k \to k \equiv o) \to \]
\[ \text{Vec} A m \to \text{Vec} A n \to \text{Vec} A o \to \]
\[ \text{DungAF} A \to \text{Vec} (A \times \text{Status}) \]
\[ (m + n + o) \]
Outline

1 Motivation of the methodology

2 Overview of the specifications/Implementations/Formalisations

3 Examples of our approach
   Implementation and verification using FP Theorem proving

4 Conclusions
The functional programming approach has shown to work very well for:
Conclusions (1)

The functional programming approach has shown to work very well for:

- implementation of abstract argumentation;
- implementation of structured argumentation, and for DSLs;
- implementation of translations between models;
- quick verification of all three.
Conclusions (1)

The functional programming approach has shown to work very well for:

- implementation of abstract argumentation;
- implementation of structured argumentation, and for DSLs;
Conclusions (1)

The functional programming approach has shown to work very well for:

- implementation of abstract argumentation;
- implementation of structured argumentation, and for DSLs;
- implementation of translations between models;
Conclusions (1)

The functional programming approach has shown to work very well for:

- implementation of abstract argumentation;
- implementation of structured argumentation, and for DSLs;
- implementation of translations between models;
- quick verification of all three.
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:

- formalisation of Dung’s AFs up to grounded semantics is very manageable;

Part of this problem can be solved immediately, by using constructive mathematics/type theory for specifications and proofs of argumentation models.

The remaining research of my PhD will hopefully determine the merits of this approach more precisely.
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:

- formalisation of Dung’s AFs up to grounded semantics is very manageable;
- but further theorem proving is hard:
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:

- formalisation of Dung’s AFs up to grounded semantics is very manageable;
- but further theorem proving is hard:
  - existing proofs are often non-constructive, making formalisation a big effort;
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:

- **formalisation** of Dung’s AFs up to grounded semantics is very manageable;
- but further **theorem proving** is hard:
  - existing proofs are often non-constructive, making formalisation a big effort;
  - structure of translations and their proofs are relatively ad-hoc.

Part of this problem can be solved immediately, by using constructive mathematics/type theory for specifications and proofs of argumentation models.

The remaining research of my PhD will hopefully determine the merits of this approach more precisely.
Conclusions (2)

The theorem proving approach has large gains, but can be a significant effort:

- **formalisation** of Dung’s AFs up to grounded semantics is very manageable;
- but further **theorem proving** is hard:
  - existing proofs are often non-constructive, making formalisation a big effort;
  - structure of translations and their proofs are relatively ad-hoc.

Part of this problem can be solved immediately, by using constructive mathematics/type theory for specifications and proofs of argumentation models.
The theorem proving approach has large gains, but can be a significant effort:

- formalisation of Dung’s AFs up to grounded semantics is very manageable;
- but further theorem proving is hard:
  - existing proofs are often non-constructive, making formalisation a big effort;
  - structure of translations and their proofs are relatively ad-hoc.

Part of this problem can be solved immediately, by using constructive mathematics/type theory for specifications and proofs of argumentation models.

The remaining research of my PhD will hopefully determine the merits of this approach more precisely.