Towards a framework for the implementation and verification of translations between argumentation models

Bas van Gijzel and Henrik Nilsson

University of Nottingham

August 28, 2013
Outline

1. Argumentation theory: a perceived problem
2. Overview of the work done
Outline

1. Argumentation theory: a perceived problem
2. Overview of the work done
Argumentation theory

Interdisciplinary area with various applications:
Argumentation theory

Interdisciplinary area with various applications:

• **Law:**
  Systems *modelling* legal problems/cases,
Argumentation theory

Interdisciplinary area with various applications:

- **Law:** Systems *modelling* legal problems/cases,
- **Decision making:** *Organising* information and source of *efficiency* in decision theory,
Argumentation theory

Interdisciplinary area with various applications:

• **Law:**
  Systems *modelling* legal problems/cases,

• **Decision making:**
  *Organising* information and source of *efficiency* in decision theory,

• **Communication theory/linguistics:**
  *Making* argumentation in existing texts *precise.*
Argumentation theory

Interdisciplinary area with various applications:

- **Law:**
  Systems *modelling* legal problems/cases,

- **Decision making:**
  *Organising* information and source of *efficiency* in decision theory,

- **Communication theory/linguistics:**
  *Making* argumentation in existing texts *precise*.

All these topics can give rise to different notions of argument and therefore different argumentation models.
Abstract argumentation

Dung’s (abstract) argumentation frameworks are a gold standard of argumentation.
Abstract argumentation

Dung’s (abstract) argumentation frameworks are a gold standard of argumentation.

- Abstracts from the concrete structure of argument and the reasons of conflict between arguments
Abstract argumentation

Dung’s (abstract) argumentation frameworks are a gold standard of argumentation.

- Abstracts from the concrete structure of argument and the reasons of conflict between arguments
- A significant amount of models are instances of Dung’s model (are translatable to)
Dung’s (abstract) argumentation frameworks are a gold standard of argumentation.

- Abstracts from the concrete structure of argument and the reasons of conflict between arguments
- A significant amount of models are instances of Dung’s model (are translatable to)
- Relatively simple data structures/algorithms (complexity still NP or higher for most problems)
Abstract argumentation

Dung’s (abstract) argumentation frameworks are a gold standard of argumentation.

• Abstracts from the concrete structure of argument and the reasons of conflict between arguments
• A significant amount of models are instances of Dung’s model (are translatable to)
• Relatively simple data structures/algorithms (complexity still NP or higher for most problems)
• Some recent efforts to optimise the evaluation of AFs (and Answer Set Programming)
A perceived problem

• Lack of (documented) implementations of more complex argumentation models
A perceived problem

- Lack of (documented) implementations of more complex argumentation models
- Existing translations from complex models to Dung, however again a lack of implementations
A perceived problem

- Lack of (documented) implementations of more complex argumentation models
- Existing translations from complex models to Dung, however again a lack of implementations
  - Translations are complex
  - Proofs of correctness are complex (page long proofs)
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 proposed solution (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:
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 proposed solution (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**

Our suggestion: **functional programming**, in specific **Haskell**.
A proposed solution (2)

- Provide implementation of Dung and some other models (Carneades, ASPIC+) in Haskell
A proposed solution (2)

- Provide implementation of Dung and some other models (Carneades, ASPIC+) in Haskell
  - In a tutorial-like fashion,
  - Close to the actual mathematical definitions
A proposed solution (2)

• Provide implementation of Dung and some other models (Carneades, ASPIC⁺) in Haskell
  • In a tutorial-like fashion,
  • Close to the actual mathematical definitions
• In the same fashion: implement a translation
A proposed solution (2)

• Provide implementation of Dung and some other models (Carneades, ASPIC\(^+\)) in Haskell
  • In a tutorial-like fashion,
  • Close to the actual mathematical definitions
• In the same fashion: implement a translation
• Provide a formalisation of implementations and translation (in Agda)
A proposed solution (2)

• Provide implementation of Dung and some other models (Carneades, ASPIC\(^+\)) in Haskell
  • In a tutorial-like fashion,
  • Close to the actual mathematical definitions
• In the same fashion: implement a translation
• Provide a formalisation of implementations and translation (in Agda)

Result: a verified way to translate models to an efficiently implemented model.
Outline

1 Argumentation theory: a perceived problem

2 Overview of the work done
Definition of AFs

An abstract argumentation framework (AF) is a tuple
$AF = \langle \text{Args}, \text{Def} \rangle$ such that:

- $\text{Args}$ is a set of (abstract) arguments,
- $\text{Def} \subseteq \text{Args} \times \text{Args}$.

In other words a directed graph.
An abstract argumentation framework (AF) is a tuple $AF = \langle \text{Args}, \text{Def} \rangle$ such that:

- $\text{Args}$ is a set of (abstract) arguments,
- $\text{Def} \subseteq \text{Args} \times \text{Args}$. 

Definition of AFs
An abstract argumentation framework (AF) is a tuple \( AF = \langle \text{Args}, \text{Def} \rangle \) such that:

- \text{Args} is a set of (abstract) arguments,
- \( \text{Def} \subseteq \text{Args} \times \text{Args} \).

In other words a directed graph.
Definition of AFs

An **abstract argumentation framework** (AF) is a tuple $AF = \langle \text{Args}, \text{Def} \rangle$ such that:

- $\text{Args}$ is a set of (abstract) arguments,
- $\text{Def} \subseteq \text{Args} \times \text{Args}$.

In other words a **directed graph**.

\[ A \rightarrow B \rightarrow C \]
Given $AF = \langle \text{Args}, \text{Def} \rangle$
AFs in Haskell

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

```haskell
data DungAF arg = AF [arg] [(arg, arg)]
deriving (Show)
```
AFs in Haskell

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

```haskell
data DungAF arg = AF [arg] [(arg, arg)]
deriving (Show)
```

Considering arguments as Strings:
AFs in Haskell

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

```haskell
data DungAF arg = AF [arg] [(arg, arg)]
  deriving (Show)
```

Considering arguments as Strings:

```haskell
type AbsArg = String
```

AFs in Haskell

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

\begin{verbatim}
data DungAF arg = AF [arg] [(arg, arg)]
deriving (Show)
\end{verbatim}

Considering arguments as Strings:

\begin{verbatim}
type AbsArg = String
\end{verbatim}

$\text{A} \rightarrow \text{B} \rightarrow \text{C}$
AFs in Haskell

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

```
data DungAF arg = AF [arg] [(arg, arg)]
deriving (Show)
```

Considering arguments as Strings:

```
type AbsArg = String
```

And in Haskell:

```
a, b, c :: AbsArg
a = "A"
b = "B"
c = "C"

AF₁ :: DungAF AbsArg
AF₁ = AF [a, b, c] [(a, b), (b, c)]
```
Conflict-freeness

Given $AF = \langle \text{Args}, \text{Def} \rangle$. 
Conflict-freeness

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

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

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

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

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

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

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

$$
\text{conflictFree} (AF \_ \text{ def}) s
= \text{null } [(a, b) \mid (a, b) \leftarrow \text{def}, a \in s, b \in s]
$$
Overview of work done (1)

- Large parts of Dung’s definitions have been implemented in Haskell,
Overview of work done (1)

• Large parts of Dung’s definitions have been implemented in Haskell,
• Most of these definitions have been formalised in Agda,
Overview of work done (1)

• Large parts of Dung’s definitions have been implemented in Haskell,
• Most of these definitions have been formalised in Agda,
• We implemented Carneades in Haskell,
Overview of work done (1)

• Large parts of Dung’s definitions have been implemented in Haskell,
• Most of these definitions have been formalised in Agda,
• We implemented Carneades in Haskell,
• Provided an implementation of a translation from Carneades into Dung in Haskell and showed which properties one would want to prove.
Overview of work done (2)

• All code is or will be available as literate Haskell/Agda,
Overview of work done (2)

• All code is or will be available as literate Haskell/Agda,
  • Paper and even the slides can be loaded into the compiler

This has caused some people to pick this up (used as a course
in Edinburgh by Alan Smaill).

Formalisation in Agda, the work on the translation and all
Haskell code is either discussed or linked to in the paper.
Overview of work done (2)

• All code is or will be available as literate Haskell/Agda,
  • Paper and even the slides can be loaded into the compiler
• Cabalised and uploaded the Dung implementation to Hackage,
Overview of work done (2)

• All code is or will be available as literate Haskell/Agda,
  • Paper and even the slides can be loaded into the compiler
• Cabalised and uploaded the Dung implementation to Hackage,
• Cabalised and uploaded the Carneades implementation to Hackage,
Overview of work done (2)

- All code is or will be available as literate Haskell/Agda,
  - Paper and even the slides can be loaded into the compiler
- Cabalised and uploaded the Dung implementation to Hackage,
- Cabalised and uploaded the Carneades implementation to Hackage,
- Almost Cabalised and uploaded the translation from Carneades into Dung,
Overview of work done (2)

- All code is or will be available as literate Haskell/Agda,
  - Paper and even the slides can be loaded into the compiler
- Cabalised and uploaded the Dung implementation to Hackage,
- Cabalised and uploaded the Carneades implementation to Hackage,
- Almost Cabalised and uploaded the translation from Carneades into Dung,
- Installation instructions (hopefully) usable for non-experienced programmers.
Overview of work done (2)

• All code is or will be available as literate Haskell/Agda,
  • Paper and even the slides can be loaded into the compiler
• Cabalised and uploaded the Dung implementation to Hackage,
• Cabalised and uploaded the Carneades implementation to Hackage,
• Almost Cabalised and uploaded the translation from Carneades into Dung,
• Installation instructions (hopefully) usable for non-experienced programmers.

This has caused some people to pick this up (used as a course in Edinburgh by Alan Smaill).
Overview of work done (2)

- All code is or will be available as literate Haskell/Agda,
  - Paper and even the slides can be loaded into the compiler
- Cabalised and uploaded the Dung implementation to Hackage,
- Cabalised and uploaded the Carneades implementation to Hackage,
- Almost Cabalised and uploaded the translation from Carneades into Dung,
- Installation instructions (hopefully) usable for non-experienced programmers.

This has caused some people to pick this up (used as a course in Edinburgh by Alan Smaill).

Formalisation in Agda, the work on the translation and all Haskell code is either discussed or linked to in the paper.
Contributions

• High-level Haskell code close to the mathematical definitions:
Contributions

• High-level Haskell code close to the mathematical definitions:
  • Allowing greater understanding of the implementation,
Contributions

• High-level Haskell code close to the mathematical definitions:
  • Allowing greater understanding of the implementation,
  • Easier realisation of existing/future translations,
Contributions

• High-level Haskell code close to the mathematical definitions:
  • Allowing greater understanding of the implementation,
  • Easier realisation of existing/future translations,
  • Written in a notation closely related to the actual use.
Contributions

• High-level Haskell code close to the **mathematical definitions**:
  • Allowing greater understanding of the implementation,
  • Easier realisation of existing/future translations,
  • Written in a notation closely related to the actual use.

• Agda **formalisation** of the Dung implementation:
Contributions

• High-level Haskell code close to the mathematical definitions:
  • Allowing greater understanding of the implementation,
  • Easier realisation of existing/future translations,
  • Written in a notation closely related to the actual use.

• Agda formalisation of the Dung implementation:
  • The first formalisation (to our knowledge) of an argumentation model,
Contributions

- High-level Haskell code close to the mathematical definitions:
  - Allowing greater understanding of the implementation,
  - Easier realisation of existing/future translations,
  - Written in a notation closely related to the actual use.

- Agda formalisation of the Dung implementation:
  - The first formalisation (to our knowledge) of an argumentation model,
  - Easier formalisation of existing/future translations,
Contributions

• High-level Haskell code close to the mathematical definitions:
  • Allowing greater understanding of the implementation,
  • Easier realisation of existing/future translations,
  • Written in a notation closely related to the actual use.

• Agda formalisation of the Dung implementation:
  • The first formalisation (to our knowledge) of an argumentation model,
  • Easier formalisation of existing/future translations,
  • A better understanding of the meaning of some of the complexer argumentation models.
Future work

- **Formalisation** of Carneades’ definitions
Future work

- Formalisation of Carneades’ definitions
- Further formalisation of Dung’s definitions and theorems:
Future work

- **Formalisation** of Carneades’ definitions
- **Further formalisation** of Dung’s definitions and theorems:
  - Formalisation of fixpoints in Agda is a lot of work!
Future work

- **Formalisation** of Carneades’ definitions
- **Further formalisation** of Dung’s definitions and theorems:
  - Formalisation of fixpoints in Agda is a lot of work!
- **Further documentation and a formalisation** of the translation from Carneades to Dung.
Future work

• **Formalisation** of Carneades’ definitions
• **Further formalisation** of Dung’s definitions and theorems:
  • Formalisation of fixpoints in Agda is a lot of work!
• **Further documentation and a formalisation** of the translation from Carneades to Dung.
• **Connect the implementation** of Dung’s AFs to an **optimised implementation** using ASP or SAT