An embedded domain specific language for Carneades

Bas van Gijzel

Supervisor:
Henrik Nilsson

University of Nottingham

April 20, 2012
Problem (1)

• Several argumentation models:
  • Dung (1995),
  • Carneades,
  • Assumption-based argumentation,
  • Besnard and Hunter’s approach,
  • ASPIC⁺,
  • etc.
Problem (1)

- Several argumentation models:
  - Dung (1995),
  - Carneades,
  - Assumption-based argumentation,
  - Besnard and Hunter’s approach,
  - ASPIC$^+$,
  - etc.

- Multiple implementations of these models.

- How do these models/implementations relate?
• Several (recent) translations between argumentation models:
  • Carneades $\rightarrow$ ASPIC$^+$ $\rightarrow$ Dung,
  • Assumption-based argumentation $\rightarrow$ Dung,
  • Besnard and Hunter’s approach $\rightarrow$ ASPIC$^+$,
  • etc...
Several (recent) translations between argumentation models:
- Carneades $\rightarrow$ ASPIC$^+$ $\rightarrow$ Dung,
- Assumption-based argumentation $\rightarrow$ Dung,
- Besnard and Hunter’s approach $\rightarrow$ ASPIC$^+$,
- etc...

However, a severe lack of concrete realisations.
Goal: provide a method/common language to:

1. implement argumentation models,
2. implement relations/translation between models.
Outline

1. Carneades

2. Carneades as an embedded domain specific language (EDSL)

3. Conclusions and future directions
1 Carneades

2 Carneades as an embedded domain specific language (EDSL)

3 Conclusions and future directions

- Structured argumentation (like ASPIC$^+$)
- Distinguishing feature: proof standards on a local level.
Pro and con arguments

Two types of arguments regarding a conclusion $c$:

- An argument with conclusion $c$ is called pro $c$,
- An argument for an opposite conclusion, $\overline{c}$, is called con $c$. 
Pro and con arguments

Two types of arguments regarding a conclusion $c$:

- An argument with conclusion $c$ is called pro $c$,
- An argument for an opposite conclusion, $\overline{c}$, is called con $c$.

Aggregation of pro and con is done through proof standards.
Arguments in Carneades consist of a two step inference:

- **Applicability** of an argument.
- **Acceptability** of the conclusion $c$. 
Applicability
Acceptability

$\text{intent}$ $\rightarrow$ $a_1$ $\rightarrow$ $0.8$ $\rightarrow$ $\text{murder}$

$\text{kill}$ $\rightarrow$ $a_1$

$\text{witness}$ $\rightarrow$ $a_2$ $\rightarrow$ $0.3$ $\rightarrow$ $\text{intent}$

$\text{unreliable}$ $\rightarrow$ $a_2$

$\text{witness2}$ $\rightarrow$ $a_3$ $\rightarrow$ $0.3$ $\rightarrow$ $\neg\text{intent}$

$\text{unreliable2}$
Proof standards

Five proof standards:
Proof standards

Five proof standards:

• Scintilla of evidence,
• Preponderance of the evidence,
• Clear and convincing evidence,
• Beyond reasonable doubt,
• Dialectical validity.
Beyond reasonable doubt

Given a CAES $C = \langle \text{arguments, audience, standard} \rangle$ and $p \in \mathcal{L}$.

\textit{beyond-reasonable-doubt}(p, arguments, audience) = true iff

1. There is an applicable $a \in \text{arguments}$ with weight $(a) > \alpha$,
2. weight $(a)$ exceeds the weight of the applicable contra arguments by $\beta$,
3. the weight of all applicable contra arguments is less than $\gamma$. 
Beyond reasonable doubt

Given a CAES $C = \langle \text{arguments}, \text{audience}, \text{standard} \rangle$ and $p \in \mathcal{L}$.

$\text{beyond}\text{-}\text{reasonable}\text{-}\text{doubt}(p, \text{arguments}, \text{audience}) = \text{true}$ iff

- There is an applicable $a \in \text{arguments}$ with $\text{weight}(a) > \alpha$, 
- $\text{weight}(a)$ exceeds the weight of the applicable counter-arguments by $\beta$, 
- The weight of all applicable counter-arguments is less than $\gamma$. 

Beyond reasonable doubt

Given a CAES $C = \langle \text{arguments}, \text{audience}, \text{standard} \rangle$ and $p \in \mathcal{L}$.

$\text{beyond-reasonable-doubt}(p, \text{arguments}, \text{audience}) = \text{true}$ iff

- There is an applicable $a \in \text{arguments}$ with $\text{weight}(a) > \alpha$,
- $\text{weight}(a)$ exceeds the weight of the applicable con arguments by $\beta$, 
Beyond reasonable doubt

Given a CAES $C = \langle arguments, audience, standard \rangle$ and $p \in \mathcal{L}$.

$\textit{beyond-reasonable-doubt}(p, arguments, audience) = true$ iff

- There is an applicable $a \in arguments$ with $\text{weight}(a) > \alpha$,
- $\text{weight}(a)$ exceeds the weight of the applicable con arguments by $\beta$,
- the weight of all applicable con arguments is less than $\gamma$. 
State of existing implementation

- Well-developed implementation of Carneades in Clojure\(^1\).
- A large part is focussed on user interaction, efficiency.
- Code relatively easy to read, but hard to see the formal relation.

\(^1\)http://carneades.github.com
State of existing implementation

- Well-developed implementation of Carneades in Clojure\(^1\).
- A large part is focused on user interaction, efficiency.
- Code relatively easy to read, but hard to see the formal relation.
  - Implementation is more general than the original model.
  - Function definitions behave differently than the original definition.

\(^1\)http://carneades.github.com
Outline

1. Carneades

2. Carneades as an embedded domain specific language (EDSL)

3. Conclusions and future directions
An embedded domain specific language

Why an **embedded** domain specific language?

- Allows **high-level code**, close to the mathematical definitions,
- An **embedding** does not need a separate parser, compiler and allows use of the embedding language.
An embedded domain specific language

Why an **embedded** domain specific language?

- Allows **high-level code**, close to the mathematical definitions,
- An **embedding** does not need a separate parser, compiler and allows use of the embedding language.

**Disadvantages:**

- Domain-specific, and therefore most suitable for users comfortable with the **domain**,
- Trade-off between **optimisation** and **complicatedness**.
An embedded domain specific language

Why an **embedded** domain specific language?

- Allows **high-level code**, close to the mathematical definitions,
- An **embedding** does not need a separate parser, compiler and allows use of the embedding language.

**Disadvantages:**

- Domain-specific, and therefore most suitable for users comfortable with the **domain**,
- Trade-off between **optimisation** and **complicatedness**.

Thus: mainly useful for **argumentation theorists** and to do **formal comparisons**.
Haskell data types

```haskell
data Bool = False | True
¬ :: Bool → Bool
¬ False = True
¬ True  = False
```
Arguments in Carneades in Haskell

A **propositional** language $\mathcal{L}$.
An argument $\langle P, E, c \rangle$ has 3 parts:
A propositional language $\mathcal{L}$. An argument $\langle P, E, c \rangle$ has 3 parts:

- premises, $P \subset \mathcal{L}$,
- exceptions, $E \subset \mathcal{L}$,
- conclusion, $c \in \mathcal{L}$.
A **propositional** language $\mathcal{L}$. An argument $\langle P, E, c \rangle$ has 3 parts:

- premises, $P \subset \mathcal{L}$,
- exceptions, $E \subset \mathcal{L}$,
- conclusion, $c \in \mathcal{L}$.

**type** Proposition = (Boolean, String)

**data** Argument = Arg [Proposition] [Proposition] Proposition
type ProofStandard = Proposition → CAES → Bool

type Audience = (Assumptions, Weight)

type Assumptions = [Proposition]

type Weight = Argument → Double
Acyclic argument set

\textbf{type} \textit{AGraph} = \ldots \quad \text{-- abstract}
Acyclic argument set

\[
\textbf{type} \ AGraph = \ldots \quad \text{-- abstract}
\]

One useful concrete instance:

\[
\textbf{type} \ AGraph = \text{Gr} (\text{Proposition}, \text{[Argument]}) ()
\]
Acyclic argument set

\textbf{type} \textit{AGraph} = \ldots \quad \text{-- abstract}

One useful concrete instance:

\textbf{type} \textit{AGraph} = \textit{Gr} (\textit{Proposition}, [\textit{Argument}]) ()

Graph with:
\begin{itemize}
  \item nodes: \textit{Propositions} paired with their \textit{pro} arguments.
  \item edges: unlabelled.
\end{itemize}
Acceptability of propositions

Given a CAES \( C = \langle \text{arguments, audience, standard} \rangle \).
A literal \( p \) is acceptable in \( C \) iff its proof standard returns \text{true}.

\[
\text{acceptable} :: \text{Proposition} \rightarrow \text{CAES} \rightarrow \text{Bool}
\]

\[
\text{acceptable} \ c \ \text{caes} @
(CAES (\text{args,}
      \text{audience} @ (\text{assumptions, weight,}
                        \text{standard}))
  = s \ c \ \text{caes}
\]

\text{where} \ s = \text{standard} \ c
Example revisited

\[
\begin{align*}
\text{intent} & \rightarrow 0.8 \rightarrow \text{murder} \\
\text{kill} & \rightarrow 0.3 \rightarrow \text{intent} \\
\text{witness} & \rightarrow 0.3 \rightarrow \text{unreliable} \\
\text{witness2} & \rightarrow 0.3 \rightarrow \text{unreliable2} \\
\end{align*}
\]
arg1, arg2, arg3 :: Argument
arg1 = mkArg["intent","kill"][] "murder"
arg2 = mkArg["witness"]["unreliable"] "intent"
arg3 = mkArg["witness2"]["unreliable2"] "-intent"
The EDSL in action (1)

arg1, arg2, arg3 :: Argument
arg1 = mkArg ["intent", "kill"] "murder"
arg2 = mkArg ["witness"] ["unreliable"] "intent"
arg3 = mkArg ["witness2"] ["unreliable2"] "-intent"

argGraph :: AGraph
argGraph = mkArgGraph [arg1, arg2, arg3]
weight :: Weight
weight arg | arg ≡ arg1 = 0.8
weight arg | arg ≡ arg2 = 0.3
weight arg | arg ≡ arg3 = 0.8
The EDSL in action (2)

weight :: Weight
weight arg | arg ≡ arg1 = 0.8
weight arg | arg ≡ arg2 = 0.3
weight arg | arg ≡ arg3 = 0.8

assumptions :: [Proposition]
assumptions = mkAssumptions
            ["kill","witness",
             "witness2","unreliable2"]
The EDSL in action (2)

weight :: Weight
weight arg | arg ≡ arg1 = 0.8
weight arg | arg ≡ arg2 = 0.3
weight arg | arg ≡ arg3 = 0.8

assumptions :: [Proposition]
assumptions = mkAssumptions
          ["kill","witness",
           "witness2","unreliable2"]

audience :: Audience
audience = (assumptions, weight)
standard :: AssignStandard
standard (_, "intent") = beyond_reasonable_doubt
standard _ = scintilla
The EDSL in action (3)

\[
\begin{align*}
\text{standard} &::= \text{AssignStandard} \\
\text{standard} (\_ , "intent") &:= \text{beyond\_reasonable\_doubt} \\
\text{standard} \_ &:= \text{scintilla} \\
\text{caes} &::= \text{CAES} \\
\text{caes} &= \text{CAES} (\text{argGraph}, \text{audience}, \text{standard})
\end{align*}
\]
The EDSL in action (4)

```haskell
acceptable (mkProp "murder") caes
> False
acceptable (mkProp "¬murder") caes
> False
```
Conclusions

Carneades as an EDSL in Haskell gave us:

- **High-level code close to the mathematical definitions:**
  - Allowing greater understanding of the implementation,
  - Easier realisation of existing/future translations,
  - Giving a DSL already familiar to argumentation theorists.

- **No separate parser, compiler, etc.**
Future directions

- Transfer the functional definitions to an interactive theorem prover, such as Agda.
  - Possible to prove properties of the implementation,
  - Possible to prove a translation keeps desired properties.
Future directions

• Transfer the functional definitions to an interactive theorem prover, such as Agda.
  • Possible to prove properties of the implementation,
  • Possible to prove a translation keeps desired properties.
• Strong connection between lambda calculus, category theory and argumentation:
  • See Logic of Argumentation (Krause et al. 1995).
  • Haskell/Agda as languages are both functional and strongly tied to category theory.