Haskell Gets Argumentative

Bas van Gijzel and Henrik Nilsson

Functional Programming Laboratory,
School of Computer Science,
University of Nottingham,
United Kingdom
{bmv,nhn}@cs.nott.ac.uk

Abstract. Argumentation theory is an interdisciplinary field studying how conclusions can be reached through logical reasoning. The notion of argument is completely general, including for example legal arguments, scientific arguments, and political arguments. *Computational* argumentation theory is studied in the context of artificial intelligence, and a number of computational argumentation frameworks have been put forward to date. However, there is a lack of concrete, high level realisations of these frameworks, which hampers research and applications at a number of levels. We hypothesise that the lack of suitable domain-specific languages in which to formalise argumentation frameworks is a contributing factor. In this paper, we present a formalisation of a particular computational argumentation framework, Carneades, as a case study with a view to investigate the extent to which functional languages are useful as a means to realising computational argumentation frameworks and reason about them.

Keywords: computational argumentation theory, domain-specific languages, functional programming

1 Introduction

Argumentation theory is an interdisciplinary field studying how conclusions can be reached through logical reasoning. Argumentation should here be understood in a general sense, including for example political debates along with more rigorous settings such as a legal or a scientific argument. A central aspect is that there is usually not going to be any clear-cut truth, but rather arguments and counter-arguments, possibly carrying different weights, and possibly relative to to the understanding of a specific audience and what is known at a specific point in time. The question, then, is what it means to systematically evaluate such a set of arguments to reach a rational conclusion, which in turn leads to the notion of *proof standards*, such as "beyond reasonable doubt". Fields that intersect with argumentation theory thus include philosophy (notably epistemology and the philosophy of science and mathematics), logic, rhetoric, and psychology.

Argumentation theory has also been studied from a *computational* perspective in the field of artificial intelligence, with the dual aim of studying argumentation theory as such [7, 24] and of more direct applications such as verification of arguments [27] or for programming autonomous agents capable of argumentation [25]. Dung's *abstract* argumentation framework [8] has been particularly influential, as it attempts to capture only the essence of arguments, thus making it generally applicable across different argumentation domains.

Since Dung's seminal work, a number of other computational argumentation frameworks have been proposed, and the study of their relative merits and exact, mathematical relationships is now an active sub-field in its own right [3, 2, 13, 18, 23]. However, a problem here is the lack of concrete realisations of many of these frameworks, in particular realisations that are sufficiently close to the mathematical definitions to serve as specifications in their own right. This hampers communication between argumentation theorists, impedes formal verification of frameworks and their relationships as well as investigation of their computational complexity, and raises the barrier of entry for people interested in developing practical applications of computational argumentation.

We believe that a contributing factor to this state of affairs is the lack of a language for expressing such frameworks that on the one hand is sufficiently high-level to be attractive to argumentation theorists, and on the other is rigorous and facilitates formal (preferably machine-checked) reasoning. We further hypothesise that a functional, domain-specific language (DSL) would be a good way to address this problem, in particular if realised in close connection with a proof assistant.

The work presented in this paper is a first step towards such a language. In order to learn how to best capture argumentation theory in a functional setting, we have undertaken a case study of casting a particular computational argumentation framework, Carneades [17, 16], into Haskell[1]. We ultimately hope to generalise this into an embedded DSL for argumentation theory, possibly within the dependently typed language Agda with a view to facilitate machine checking of proofs about arguments and the relationships between argumentation frameworks. While we are still a long way away from this goal, the initial experience from our case study has been positive: our formalisation in Haskell was deemed to be intuitive and readable as a specification on its own by Tom Gordon, an argumentation theorist and one of the authors of the Carneades argumentation framework [15]. Furthermore, our case study is a contribution in its own right in that it:

- already constitutes a helpful tool for argumentation theorists;
- demonstrates the usefulness of a language like Haskell itself as a tool for argumentation theorists, albeit assuming a certain proficiency in functional programming;
- is a novel application of Haskell that should be of interest for example to researchers interested in using Haskell for AI research and applications.

This is not to say that there are no implementations of *specific* argumentation theory frameworks around; see Sect. 4 for an overview. However, the goals and structure of those systems are rather different from what we are concerned

[1] Cabal package on Hackage: `http://hackage.haskell.org/package/CarneadesDSL`.

with in our case study. In particular, a close and manifest connection between argumentation theory and its realisation in software appears not to be a main objective of existing work. For us, on the other hand, maintaining such a connection is central, as this is the key to our ultimate goal of a successful *generic* DSL suitable for realising *any* argumentation framework.

The rest of this paper is structured as follows. In Sect. 2, we give an intuitive introduction to Carneades, both to provide a concrete and easy to grasp example of what argumentation frameworks are and how they work, and to provide a grounding for the technical account of Carneades and our implementation of it that follows. We then continue in Sect. 3 by giving the formal definition of the central parts of Carneades juxtapositioned with our realisation in Haskell. The section covers central notions such as the argumentation graph that captures the relationships between arguments and counter arguments, the exact characterisation of proof standards (including "beyond reasonable doubt"), and the notion of an audience with respect to which arguments are assigned weights. Related work is discussed in Sect. 4, and we conclude in Sect. 5 with a discussion of what we have learnt from this case study, its relevance to argumentation theorists, and various avenues for future work.

2 Background: The Carneades Argumentation Model

The main of purpose of the Carneades argumentation model is to formalise argumentation problems in a legal context. Carneades contains mathematical structures to represent arguments placed in favour of or against atomic propositions; i.e., an argument in Carneades is a *single* inference step from a set of *premises* and *exceptions* to a *conclusion*, where all propositions in the premises, exceptions and conclusion are literals in the language of propositional logic. For example, Fig. 1 gives an argument in favour of the proposition *murder* mimicking an argument that might be put forward in a court case.

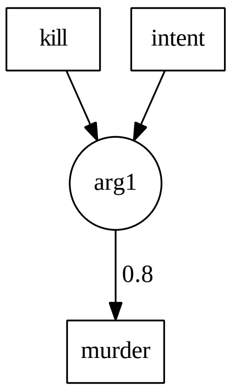
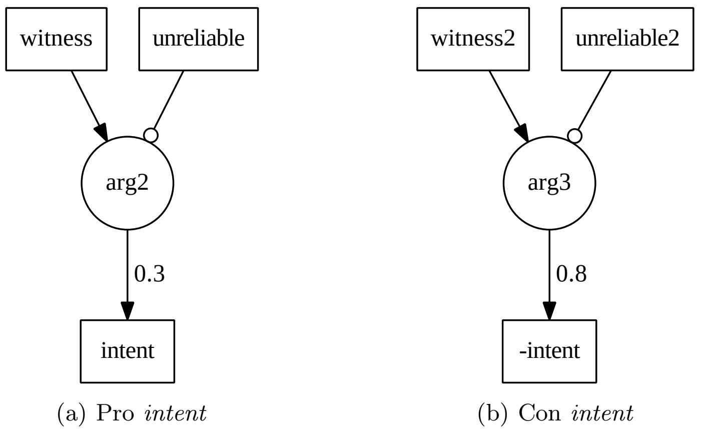


Fig. 1. Carneades argument for murder

For ease of reference, we name the argument (*arg1*). However, arguments are not formally named in Carneades, but instead identified by their logical content. An argument is only to be taken into account if it is *applicable* in a technical sense defined in Carneades. In this case, *arg1* is applicable given that its two premises *kill* and *intent* are *acceptable*, also in a technical sense defined in Carneades. (We will come back to exceptions below.) In other words, we are able to derive that there was a murder, given that we know (with sufficient certainty) that someone was killed and that this was done with intent.

In Carneades, a set of arguments is evaluated relative to a specific *audience* (jury). The audience determines two things: a set of *assumptions*, and the *weight* of each argument, ranging from 0 to 1. The assumptions are the premises and exceptions that are taken for granted by the audience, while the weights reflect the subjective merit of the arguments. In our example, the weight of *arg1* is 0.8, and it is applicable if *kill* and *intent* are either assumptions of the audience, or have been derived by some other arguments, relative to the *same* audience.

Things get more interesting when there are arguments both for and against the same proposition. The conclusion of an argument against an atomic proposition is the propositional negation of that proposition, while an argument against a negated atomic proposition is just the (positive) proposition itself. Depending on the type of proposition, and even the type of case (criminal or civil), there are certain requirements the arguments should fulfil to tip the balance in either direction. These requirements are called *proof standards*. Carneades specifies a range of proof standards, and to model opposing arguments we need to assign a specific proof standard, such as *clear and convincing evidence*, to a proposition.



(a) Pro *intent* (b) Con *intent*

Fig. 2. Arguments pro and con *intent*

Consider the two arguments in Fig. 2, where the arrows with circular heads indicate exceptions. Figure 2(a) represents an argument in favour of *intent*. It is applicable given that the premise *witness* is acceptable and the exception *unreliable* does not hold. Figure 2(b) represents an argument against *intent*. It involves a second witness, *witness2*, who claims the opposite of the first witness. Let us assume that the required proof standard for *intent* indeed is *clear and convincing evidence*, which Carneades formally defines as follows:

Definition 1 (Clear and convincing evidence). *Given two globally predefined positive constants α and β; clear and convincing evidence holds for a specific proposition p iff*

- *There is at least one applicable argument for proposition p that has at least a weight of α.*
- *The maximal weight of the applicable arguments in favour of p are at least β stronger than the maximal weight of the applicable arguments against p.*

Taking $\alpha = 0.2$, $\beta = 0.3$, and given an audience that determines the argument weights to be as per the figure and that assumes $\{\mathit{witness}, \mathit{witness2}\}$, we have that $-\mathit{intent}$ is acceptable, because *arg3* and *arg2* are applicable, $\text{weight}(\mathit{arg3}) > \alpha$, and $\text{weight}(\mathit{arg3}) > \text{weight}(\mathit{arg2}) + \beta$.

For another example, had *unreliable2* been assumed as well, or found to be acceptable through other (applicable) arguments, that would have made *arg3* inapplicable. That in turn would make *intent* acceptable, as the weight 0.3 of *arg2* satisfies the conditions for clear and convincing evidence given that there now are no applicable counter arguments, and we could then proceed to establish *murder* by *arg1* had it been established that someone indeed was killed.

3 Towards a DSL for Carneades in Haskell

3.1 Arguments

As our ultimate goal is a DSL for argumentation theory, we strive for a realisation in Haskell that mirrors the mathematical model of Carneades argumentation framework as closely as possible. Ideally, there would be little more to a realisation than a transliteration. We will thus proceed by stating the central definitions of Carneades along with our realisation of them in Haskell.

Definition 2 (Arguments). *Let $\mathcal{L}$ be a propositional language. An* argument *is a tuple $\langle P, E, c\rangle$ where $P \subset \mathcal{L}$ are its* premises*, $E \subset \mathcal{L}$ with $P \cap E = \emptyset$ are its* exceptions *and $c \in \mathcal{L}$ is its* conclusion*. For simplicity, all members of $\mathcal{L}$ must be literals, i.e. either an atomic proposition or a negated atomic proposition. An argument is said to be* pro *its conclusion c (which may be a negative atomic proposition) and* con *the negation of c.*

In Carneades all logical formulae are literals in propositional logic; i.e., all propositions are either positive or negative atoms. Taking atoms to be strings suffice in the following, and propositional literals can then be formed by pairing this atom with a Boolean to denote whether it is negated or not:

```
type PropLiteral = (Bool, String)
```

We write $\overline{p}$ for the negation of a literal p. The realisation is immediate:

```
negate :: PropLiteral → PropLiteral
negate (b, x) = (¬ b, x)
```

We chose to realise an *argument* as a newtype (to allow a manual Eq instance) containing a tuple of two lists of propositions, its *premises* and its *exceptions*, and a proposition that denotes the *conclusion*:

```
newtype Argument = Arg ([PropLiteral], [PropLiteral], PropLiteral)
```

Arguments are considered equal if their premises, exceptions and conclusion are equal; thus arguments are identified by their logical content. The equality instance for *Argument* (omitted for brevity) takes this into account by comparing the lists as sets.

A set of arguments determines how propositions depend on each other. Carneades requires that there are no cycles among these dependencies. Following Brewka and Gordon [3], we use a dependency graph to determine acyclicity of a set of arguments.

Definition 3 (Acyclic set of arguments). *A set of* arguments *is* acyclic *iff its corresponding dependency graph is acyclic. The corresponding dependency graph has a node for every literal appearing in the set of arguments. A node p has a link to node q whenever p depends on q in the sense that there is an argument pro or con p that has q or $\overline{q}$ in its set of premises or exceptions.*

Our realisation of a set of arguments is considered abstract for DSL purposes, only providing a check for acyclicity and a function to retrieve arguments pro a proposition. We use FGL [9] to implement the dependency graph, forming nodes for propositions and edges for the dependencies. For simplicity, we opt to keep the graph also as the representation of a set of arguments.

```
type ArgSet = ...
getArgs    :: PropLiteral → ArgSet → [Argument]
checkCycle :: ArgSet → Bool
```

3.2 Carneades Argument Evaluation Structure

The main structure of the argumentation model is called a Carneades Argument Evaluation Structure (CAES):

Definition 4 (Carneades Argument Evaluation Structure (CAES)). *A* Carneades Argument Evaluation Structure *(CAES) is a triple*

$$\langle arguments, audience, standard \rangle$$

where arguments is an acyclic set of arguments, audience is an audience as defined below (Def. 5), and standard *is a total function mapping each proposition to to its specific proof standard.*

Note that propositions may be associated with *different* proof standards. This is considered a particular strength of the Carneades framework. The transliteration into Haskell is almost immediate[2]:

newtype *CAES* = *CAES* (*ArgSet*, *Audience*, *PropStandard*)

Definition 5 (Audience). *Let* $\mathcal{L}$ *be a propositional language. An* audience *is a tuple* ⟨assumptions, weight⟩*, where assumptions* $\subset \mathcal{L}$ *is a propositionally consistent set of literals (i.e., not containing both a literal and its negation) assumed to be acceptable by the audience and* weight *is a function mapping arguments to a real-valued weight in the range* $[0, 1]$.

This definition is captured by the following Haskell definitions:

type *Audience* = (*Assumptions*, *ArgWeight*)
type *Assumptions* = [*PropLiteral*]
type *ArgWeight* = *Argument* → *Weight*
type *Weight* = *Double*

Further, as each proposition is associated with a specific proof standard, we need a mapping from propositions to proof standards:

type *PropStandard* = *PropLiteral* → *ProofStandard*

A proof standard is a function that given a proposition p, aggregates arguments pro and con p and decides whether it is acceptable or not:

type *ProofStandard* = *PropLiteral* → *CAES* → *Bool*

This aggregation process will be defined in detail in the next section, but note that it is done relative to a specific CAES, and note the cyclic dependence at the type level between *CAES* and *ProofStandard*.

The above definition of proof standard also demonstrates that implementation in a typed language such as Haskell is a useful way of verifying definitions from argumentation theoretic models. Our implementation effort revealed that the original definition as given in [17] could not be realised as stated, because proof standards in general not only depend on a set of arguments and the audience, but may need the whole CAES.

3.3 Evaluation

Two concepts central to the evaluation of a CAES are *applicability of arguments*, which arguments should be taken into account, and *acceptability of propositions*, which conclusions can be reached under the relevant proof standards, given the beliefs of a specific audience.

[2] Note that we use a newtype to prevent a cycle in the type definitions.

Definition 6 (Applicability of arguments). *Given a set of arguments and a set of assumptions (in an audience) in a CAES C, then an argument $a = \langle P, E, c\rangle$ is* applicable *iff*

- $p \in P$ *implies* p *is an assumption or [*$\overline{p}$ *is not an assumption and* p *is acceptable in* C *] and*
- $e \in E$ *implies* e *is not an assumption and [*$\overline{e}$ *is an assumption or* e *is not acceptable in* C *].*

Definition 7 (Acceptability of propositions). *Given a CAES* C*, a proposition* p *is* acceptable *in* C *iff* $(s\ p\ C)$ *is true, where* s *is the proof standard for* p*.*

Note that these two definitions in general are mutually dependent because acceptability depends on proof standards, and most sensible proof standards depend on the applicability of arguments. This is the reason that Carneades restricts the set of arguments to be acyclic. (Specific proof standards are considered in the next section.) The realisation of applicability and acceptability in Haskell is straightforward:

```
applicable :: Argument → CAES → Bool
applicable (Arg (prems, excns, _)) caes@(CAES (_, (assumptions, _), _))
   = and $ [(p ∈ assumptions) ∨ (p `acceptable` caes) | p ← prems]
             ++
            [(e ∈ assumptions) ↓ (e `acceptable` caes) | e ← excns ]
     where
       x ↓ y = ¬ (x ∨ y)
acceptable :: PropLiteral → CAES → Bool
acceptable c caes@(CAES (_, _, standard))
   = c `s` caes
   where s = standard c
```

3.4 Proof standards

Carneades predefines five proof standards, originating from the work of Freeman and Farley [12, 11]: *scintilla of evidence*, *preponderance of the evidence*, *clear and convincing evidence*, *beyond reasonable doubt* and *dialectical validity*. Some proof standards depend on constants such as α, β, γ; these are assumed to be defined once and globally. This time, we proceed to give the definitions directly in Haskell, as they really only are translitarations of the original definitions.

For a proposition p to satisfy the weakest proof standard, scintilla of evidence, there should be at least one applicable argument pro p in the CAES:

```
scintilla :: ProofStandard
scintilla p caes@(CAES (g, _, _))
  = any (`applicable`caes) (getArgs p g)
```

Preponderance of the evidence additionally requires the maximum weight of the applicable arguments pro p to be greater than the maximum weight of the applicable arguments con p. The weight of zero arguments is taken to be 0. As the maximal weight of applicable arguments pro and con is a recurring theme in the definitions of several of the proof standards, we start by defining those notions:

```
maxWeightApplicable :: [Argument] → CAES → Weight
maxWeightApplicable as caes@(CAES (_, (_, argWeight), _))
 = foldl max 0 [argWeight a | a ← as, a ‘applicable‘ caes]
maxWeightPro :: PropLiteral → CAES → Weight
maxWeightPro p caes@(CAES (g, _, _))
 = maxWeightApplicable (getArgs p g) caes
maxWeightCon :: PropLiteral → CAES → Weight
maxWeightCon p caes@(CAES (g, _, _))
 = maxWeightApplicable (getArgs (negate p) g) caes
```

We can then define the proof standard preponderance:

```
preponderance :: ProofStandard
preponderance p caes = maxWeightPro p caes > maxWeightCon p caes
```

Clear and convincing evidence strengthen the preponderance constraints by insisting that the difference between the maximal weights of the pro and con arguments must be greater than a given positive constant β, and there should furthermore be at least one applicable argument pro p that is stronger than a given positive constant α:

```
clear_and_convincing :: ProofStandard
clear_and_convincing p caes
 =  (mwp > α) ∧ (mwp − mwc > β)
   where
     mwp = maxWeightPro p caes
     mwc = maxWeightCon p caes
```

Beyond reasonable doubt has one further requirement: the maximal strength of an argument con p must be less than a given positive constant γ; i.e., there must be no reasonable doubt:

```
beyond_reasonable_doubt :: ProofStandard
beyond_reasonable_doubt p caes
 = clear_and_convincing p caes ∧ (maxWeightCon p caes < γ)
```

Finally dialectical validity requires at least one applicable argument pro p and no applicable arguments con p:

```
dialectical_validity :: ProofStandard
dialectical_validity p caes
   = scintilla p caes ∧ ¬ (scintilla (negate p) caes)
```

3.5 Convenience functions

We provide a set of functions to facilitate construction of propositions, arguments, argument sets and sets of assumptions. Together with the definitions covered so far, this constitute our DSL for constructing Carneades argumentation models.

```
mkProp        :: String → PropLiteral
mkArg         :: [String] → [String] → String → Argument
mkArgSet      :: [Argument] → ArgSet
mkAssumptions :: [String] → [PropLiteral]
```

A string starting with a `'-'` is taken to denote a negative atomic proposition.

To construct an audience, native Haskell tupling is used to combine a set of assumptions and a weight function, exactly as it would be done in the Carneades model:

```
audience :: Audience
audience = (assumptions, weight)
```

Carneades Argument Evaluation Structures and weight functions are defined in a similar way, as will be shown in the next subsection.

Finally, we provide a function for retrieving the arguments for a specific proposition from an argument set, a couple of functions to retrieve all arguments and propositions respectively from an argument set, and functions to retrieve the (not) applicable arguments or (not) acceptable propositions from a CAES:

```
getArgs            :: PropLiteral → ArgSet → [Argument]
getAllArgs         :: ArgSet              → [Argument]
getProps           :: ArgSet              → [PropLiteral]
applicableArgs     :: CAES                → [Argument]
nonApplicableArgs  :: CAES                → [Argument]
acceptableProps    :: CAES                → [PropLiteral]
nonAcceptableProps :: CAES                → [PropLiteral]
```

3.6 Implementing a CAES

This subsection shows how an argumentation theorist given the Carneades DSL developed in this section quickly and at a high level of abstraction can implement a Carneades argument evaluation structure and evaluate it as well. We revisit the arguments from Section 2 and assume the following:

$$\begin{aligned} \mathit{arguments} &= \{\mathit{arg1}, \mathit{arg2}, \mathit{arg3}\}, \\ \mathit{assumptions} &= \{\mathit{kill}, \mathit{witness}, \mathit{witness2}, \mathit{unreliable2}\}, \\ \mathit{standard}(\mathit{intent}) &= \mathit{beyond\text{-}reasonable\text{-}doubt}, \\ \mathit{standard}(x) &= \mathit{scintilla}, \text{ for any other proposition x}, \\ \alpha &= 0.4,\ \beta = 0.3,\ \gamma = 0.2. \end{aligned}$$

Arguments and the argument graph are constructed by calling *mkArg* and *mkArgSet* respectively:

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

argSet :: ArgSet
argSet = mkArgSet [arg1, arg2, arg3]
```

The audience is implemented by defining the *weight* function and calling *mkAssumptions* on the propositions which are to be assumed. The audience is just a pair of these:

```
weight :: ArgWeight
weight arg | arg ≡ arg1 = 0.8
weight arg | arg ≡ arg2 = 0.3
weight arg | arg ≡ arg3 = 0.8
weight _                = error "no weight assigned"

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

audience :: Audience
audience = (assumptions, weight)
```

Finally, after assigning proof standards in the *standard* function, we form the CAES from the argument graph, audience and function *standard*:

```
standard :: PropStandard
standard (_, "intent") = beyond_reasonable_doubt
standard _             = scintilla

caes :: CAES
caes = CAES (argSet, audience, standard)
```

We can now try out the argumentation structure. Arguments are pretty printed in the format *premises* $\sim$ *exceptions* $\Rightarrow$ *conclusion*:

```
getAllArgs argSet
 > [["witness2"]       ~["unreliable2"] ⇒ "-intent",
    ["witness"]        ~["unreliable"]  ⇒ "intent",
    ["kill", "intent"]~[]               ⇒ "murder"]
```

As expected, there are no applicable arguments for $-$*intent*, since *unreliable2* is an exception, but there is an applicable argument for *intent*, namely *arg2*:

```
filter (`applicable` caes) $ getArgs (mkProp "-intent") argSet
 > []
```

```
filter (`applicable` caes) $ getArgs (mkProp "intent") argSet
 > [["witness"] ⇒ "intent"]
```

However, despite the applicable argument *arg2* for *intent*, *murder* should not be acceptable, because the weight of $arg2 < \alpha$. Interestingly, note that we can't reach the opposite conclusion either:

```
acceptable (mkProp "murder") caes
 > False
acceptable (mkProp "-murder") caes
 > False
```

As a further extension, one could for example imagine giving an argumentation theorist the means to see a trace of the derivation of acceptability. It would be straightforward to add further primitives to the DSL and keeping track of intermediate results for acceptability and applicability to achieve this.

4 Related Work

In this section we consider related work of direct relevance to our interests in DSLs for argumentation theory, specifically efforts in the field of computational argumentation theory to implement argumentation frameworks, and DSLs in closely related areas with similar design goals to ours.

For a general overview of implementations and a discussion of limitations regarding experimental testing, see Bryant and Krause [4]. Most closely related to the work presented in this paper is likely the well-developed implementation [14] of Carneades in Clojure[3]. However, the main aim of that implementation is to provide efficient tools, GUIs, and so on for non-specialists, not to express the implementation in a way that directly relates it to the formal model. Consequently, the connection between the implementation and the model is not immediate. This means that the implementation, while great for argumentation theorists only interested in modelling argumentation problems, is not directly useful to a computational argumentation theorist interested in relating models and implementations, or in verifying definitions. The Clojure implementation is thus in sharp contrast to our work, and reinforces our belief in the value of a high-level, principled approach to formalising argumentation theory frameworks.

One of the main attempts to unify work in argumentation theory, encompassing arguments from the computational, philosophical and the linguistic domains, is the Argument Interchange Format (AIF) [6, 26]. The AIF aims to capture arguments stated in the above mentioned domains, while providing a common core ontology for expressing argumentative information and relations. Very recent work has given a logical specification of AIF [1], providing foundations for interrelating argumentation semantics of computational models of argumentation, thereby remedying a previous weaknesses of AIF. Our implementation tackles

[3] http://carneades.berlios.de/

the problem from another direction, starting with a formal and computationally oriented language instead.

Walkingshaw and Erwig [29, 10] have developed an EDSL for neuron diagrams [21], a formalism in philosophy that can model complex causal relationships between events, similar to how premises and exceptions determine a conclusion in an argument. Walkingshaw and Erwig extend this model to work on non-Boolean values, while at the same time providing an implementation, thereby unifying formal description and actual implementation. This particular goal is very similar to ours. Furthermore, the actual formalisms of neuron diagrams and the Carneades argumentation model are technically related: while an argument on its own is a simple graph, the dependency graph corresponding to the whole Carneades argument evaluation structure is much more complex and has a structure similar to a full neuron diagram. Arguments in Carneades could thus be seen as an easy notation for a specific kind of complex neuron diagrams for which manual encoding would be unfeasible in practice. However, due to the complexity of the resulting encoding, this also means that for an argumentation theorist, neuron diagrams do not offer directly relevant abstractions. That said, Walkingshaw's and Erwig's EDSL itself could offer valuable input on the design for a DSL for argumentation.

Similarly, causal diagrams are a special case of Bayesian networks [22] with additional constraints on the semantics, restricting the relations between nodes to a causal relation (causal diagrams are a graphic and restricted version of Bayesian networks). Building on the already existing relation between Carneades and Bayesian networks [19], we can view the neuron diagrams generalised to non-Boolean values in Carneades by generalising the negation relation and proof standards to non-Boolean values in the obvious way, and picking scintilla of evidence as the proof standard for all propositions. So, in a way, neuron diagrams are a specific case of arguments, using scintilla of evidence as the proof standard. Finally, to compute an output for every combination of inputs, as is done for neuron diagrams, we can vary the set of assumptions accordingly.

However, formal connections between Bayesian networks and (dialectical) argumentation are still in its infancy; most of the work such as Grabmair [19], Keppens [20] and Vreeswijk [28] are high level relations or comparisons, containing no formal proofs.

5 Conclusions and Future Work

In this paper we have discussed the Carneades argumentation model and an implementation of it in Haskell. This paper should be seen as a case study and a step towards a generic DSL for argumentation theory, providing a unifying framework in which various argumentation models can be implemented and their relationships studied. We have seen that the original mathematical definitions can be captured at a similar level of abstraction by Haskell code, thereby allowing for greater understanding of the implementation. At the same time we obtained a domain specific language for the Carneades argumentation framework, allowing argumentation theorists to realise arguments essentially only using a vocabulary with which they are already familiar.

The initial experience from our work has been largely positive. Comments from Tom Gordon [15], one of the authors and implementers of the Carneades argumentation model, suggest that our implementation is intuitive and would even work as an executable specification, which is an innovative approach in argumentation theory as a field. However, our implementation was not seen as usable to non-argumentation theorists, because of the lack of additional tools. We do not perceive this as a worrying conclusion; our framework's focus is on computational argumentation theorists. We rather envision our implementation being used as a testing framework for computational argumentation theorists and as an intermediate language between implementations, providing a much more formal alternative to the existing Argument Interchange Format [26].

One avenue of future work is the generalisation of our DSL to other related argumentation models. It is relatively common in argumentation theory to define an entirely new model to realise a small extension. However, this hurts the meta-theory as lots of results will have to be re-established from scratch. By reducing such an extension to an existing implementation/DSL such as ours, for instance by providing an implementation of an existing formal translation such as [13, 23], we effectively formalise a translation between both models, while gaining an implementation of this generalisation at the same time.

This could be taken even further by transferring the functional definitions of an argumentation model into an interactive theorem prover, such as Agda. First of all, the formalisation of the model itself would be more precise. While the Haskell model might seem exact, note that properties such as the acyclicity of arguments, or that premises and exceptions must not overlap, are not inherently part of this model. Second, this would enable formal, *machine-checked*, reasoning *about* the model, such as establishing desirable properties like consistency of the set of derivable conclusions.

Then, if *multiple* argumentation models were to be realised in a theorem prover, relations *between* those models, such as translations, could be formalised. As mentioned in the introduction, there has recently been much work on formalisation of translations between conceptually very different argumentation models [2, 13, 23, 18]. But such a translation can be very difficult to verify if done by hand. Using a theorem prover, the complex proofs could be machine-checked, guaranteeing that the translations preserve key properties of the models. An argumentation theorist might also make use of this connection by inputting an argumentation case into one model and, through the formal translation, retrieve a specification in another argumentation model, allowing the use of established properties (such as rationality postulates [5]) of the latter model.

Finally, we are interested in the possibility of mechanised argumentation as such; e.g., as a component of autonomous agents. We thus intend to look into realising various argumentation models efficiently by considering suitable ways to implement the underlying graph structure and exploiting sharing to avoid unnecessarily duplicated work. Ultimately we hope this would allow us to establish results regarding the asymptotic time and space complexity inherent in various argumentation models, while providing a framework for empirical evaluations and testing problems sets at the same time. Especially the latter is an area that has only recently received attention [4, 2], due to the lack of implementations and automated conversion of problem sets.

Acknowledgments

The authors would like to thank the anonymous reviewer from the initial TFP feedback round whose very constructive comments helped to significantly improve this paper. We would furthermore like to thank Tom Gordon and the anonymous reviewers of the second feedback round for their helpful suggestions.

References

1. Floris Bex, Sanjay Modgil, Henry Prakken, and Chris Reed. On logical specifications of the Argument Interchange Format. *Journal of Logic and Computation*, 2012.
2. Gerhard Brewka, Paul E. Dunne, and Stefan Woltran. Relating the semantics of abstract dialectical frameworks and standard AFs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 780–785, 2011.
3. Gerhard Brewka and Thomas F. Gordon. Carneades and abstract dialectical frameworks: a reconstruction. In Massimiliano Giacomin and Guillermo R. Simari, editors, *Computational Models of Argument. Proceedings of COMMA 2010*, pages 3–12, Amsterdam etc, 2010a. IOS Press 2010.
4. Daniel Bryant and Paul Krause. A review of current defeasible reasoning implementations. *Knowl. Eng. Rev.*, 23:227–260, September 2008.
5. Martin Caminada and Leila Amgoud. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171:286–310, April 2007.
6. Carlos Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.
7. Carlos Iván Chesñevar, Ana Gabriela Maguitman, and Ronald Prescott Loui. Logical models of argument. *ACM Comput. Surv.*, 32(4):337–383, December 2000.
8. 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.
9. Martin Erwig. Inductive graphs and functional graph algorithms. *Journal Functional Programming*, 11(5):467–492, September 2001.
10. Martin Erwig and Eric Walkingshaw. Causal Reasoning with Neuron Diagrams. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 101–108, 2010.
11. Arthur M. Farley and Kathleen Freeman. Burden of proof in legal argumentation. In *Proceedings of the 5th International Conference on Artificial Intelligence and Law (ICAIL-05)*, pages 156–164, New York, NY, USA, 1995. ACM.

12. Kathleen Freeman and Arthur M. Farley. A model of argumentation and its application to legal reasoning. *Artificial Intelligence and Law*, 4:163–197, 1996. 10.1007/BF00118492.
13. Bas van Gijzel and Henry Prakken. Relating Carneades with abstract argumentation via the ASPIC$^+$ framework for structured argumentation. *Argument & Computation*, 3(1):21–47, 2012.
14. Thomas F. Gordon. An overview of the Carneades argumentation support system. In Chris Tindale and Chris Reed, editors, *Dialectics, Dialogue and Argumentation. An Examination of Douglas Walton's Theories of Reasoning*, pages 145–156. College Publications, 2010.
15. Thomas F. Gordon. Personal communication, 2012.
16. Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-15):875–896, 2007.
17. Thomas F. Gordon and Douglas Walton. Proof burdens and standards. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 239–258. Springer US, 2009.
18. Guido Governatori. On the relationship between carneades and defeasible logic. In Tom van Engers, editor, *Proceedings of the 13th International Conference on Artificial Intelligence and Law (ICAIL 2011).* ACM Press, 2011.
19. Matthias Grabmair, Thomas F. Gordon, and Douglas Walton. Probabilistic semantics for the carneades argument model using bayesian networks. In *Proceedings of the 2010 conference on Computational Models of Argument: Proceedings of COMMA 2010*, pages 255–266, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.
20. Jeroen Keppens. Argument diagram extraction from evidential bayesian networks. *Artificial Intelligence and Law*, 20:109–143, 2012.
21. David Lewis. Postscripts to 'Causation'. In *Philosphical Papers, Vol. II*, pages 196–210. Oxford University Press, 1986.
22. Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, 1985.
23. Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1:93–124, 2010.
24. Henry Prakken and Gerard A W Vreeswijk. Logics for defeasible argumentation. *Handbook of Philosophical Logic*, 4(5):219–318, 2002.
25. Iyad Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, Peter Mcburney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *Knowl. Eng. Rev.*, 18(4):343–375, December 2003.
26. Iyad Rahwan and Chris Reed. The argument interchange format. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 383–402. Springer US, 2009.
27. Chris Reed and Glenn Rowe. Araucaria: software for argument analysis, diagramming and representation. *International Journal of AI Tools*, 13(4):961–980, 2004.
28. Gerard A. W. Vreeswijk. Argumentation in bayesian belief networks. In *Proceedings of the First international conference on Argumentation in Multi-Agent Systems*, ArgMAS'04, pages 111–129, Berlin, Heidelberg, 2005. Springer-Verlag.
29. E. Walkingshaw and M. Erwig. A DSEL for Studying and Explaining Causation. In *IFIP Working Conference on Domain Specific Languages (DSL'11)*, pages 143–167, 2011.