A principled approach to the implementation of argumentation models

Bas VAN GIJZEL [1] and Henrik NILSSON
University of Nottingham

Abstract. Argumentation theory combines philosophical concepts and computational models to deliver a practical approach to reasoning that handles uncertain information and possibly conflicting viewpoints. This paper focuses on the structured approach to argumentation that incorporates domain specific knowledge and argumentation schemes. There is a lack of implementations and implementation methods for most structured models. This paper shows how taking a principled approach, using the programming language Haskell, helps addressing this problem. We construct a framework for developing structured argumentation models and translations between models (given intertranslatability of models). We furthermore provide a methodology to quickly test and formally prove desirable properties of such implementations using a theorem prover. We demonstrate our approach on the Carneades argumentation model and Dung's abstract argumentation frameworks, implementing both the models and a translation from Carneades into AFs. We then provide implementations of correspondence properties and an initial formalisation of Dung's AFs into a theorem prover. The final result is a verified pipeline from the structured model Carneades into existing efficient SAT-based implementations of Dung's AFs.

1. Introduction

Dung's argumentation frameworks [6] have an established relationship to logic programming. It is therefore not surprising that Dung's AFs have seen significant developments in the area of efficient implementation and elegant implementation methods, particularly through implementations written in logic programming and answer set programming [7] or through implementations based on SAT-solvers [3,7]. Several other abstract models, are either direct extensions of Dung's AFs or are closely related. These models can thus also be implemented with relative ease through encoding into answer set programming clauses [4], translation through other mathematical formalisms [1], or by direct implementation into a logic programming language such as Prolog.

Specifications of structured models of argumentation are more varied, due to their possible phrasing in a specific domain, or a specific logic. This has caused implementation methodology of most structured models to lag behind. Notable

[1]Corresponding author: Bas van Gijzel, bmv@cs.nott.ac.uk.

exceptions are the implementation of Carneades[2] and assumption-based argumentation [8], the latter which, due to its strong connections with Dung's AFs and therefore logic programming, has multiple mature implementations [8,14]. However, several other models have only been implemented partially, if at all, and any implementations are typically expressed in a programming paradigm or language that is very different from the mathematical specification. The situation is similar for *translations* from structured models into other abstract/structured models.

The difference in mathematical structure between structured models means that a significant effort is required to establish formal relationships between them. This is also true for the relationships between structured and abstract models [13, 11,2]. Implementing the resulting translations consequently becomes difficult, and verifying their correctness is even harder.

We attempt to address the problem of implementing structured models and their translations by providing a framework that allows implementation close to the mathematical specification and facilitates checking and formal proof of properties, the latter being key to verification of correctness. Our choice of programming language is Haskell. This is motivated by our previous work [9,10], where we implemented the Carneades argumentation model in a way that is easily understandable to argumentation theorists with no prior Haskell knowledge.

More specifically, the contributions of this paper are the following:

- We argue that Haskell enables us to intuitively capture existing abstract and structured argumentation models, providing implementations that practically serve as a mathematical specification.
- We provide, to our knowledge, the first implementation of a translation from a structured into an abstract argumentation model. In addition, we provide techniques to check the correctness of implementations by showing how to implement desirable properties, such as correspondence properties.
- We discuss and provide a formalisation of Dung's AFs into a theorem prover, obtaining the first fully machine-checkable formalisation of an argumentation model.
- We combine all this, to provide a verified pipeline, starting from an input file reading a Carneades argument structure, resulting in a file containing a Dung AF, readable by one of the fastest current implementations [7].
- All work is open source, publicly available and immediately installable[3].

The paper is structured as follows. Section 2 introduces Haskell as a suitable implementation language for both structured and abstract models. Section 3 discusses a direct algorithmic translation from Carneades into Dung and shows how Haskell can also be used to implement translation in a satisfactory manner. In Section 4 we consider quick testing and complete formalisation of argumentation models and correctness properties. We conclude in Section 5, tying all strands of work together into one verified pipeline, and discuss future work.

[2]See `https://carneades.github.com`.

[3]The implementations and formalisations are fully documented and can be found online together with a collection of additional examples: `http://www.cs.nott.ac.uk/~bmv/COMMA/`.

2. Haskell as an implementation language for structured and abstract models

Dung's AFs and most other abstract approaches to argumentation are closely aligned with logic and/or answer set programming. Consequently, implementing an argumentation model in Haskell amounts to little more than transliteration, meaning the implementation of an argumentation model can serve as a specification in its own right. Furthermore, verification of correctness is facilitated, be it informally (by inspection) or formally (through a theorem prover like Agda).

In contrast, structured argumentation not based on logic programming still lacks a completely satisfying programming methodology. For example, VISPARTIX [4], based on answer set programming, supports a knowledge base and argument construction, but it is not yet clear if more complicated external data types such as audiences or proof standards can be handled. In the following, we argue that Haskell is a suitable language for abstract as well as structured models by implementing Dung's abstract argumentation frameworks and a structured argumentation model, Carneades.

2.1. Dung's abstract argumentation frameworks

We give some of the standard definitions of Dung's AFs [6]. For the full implementation including the semi-stable labelling algorithm, we refer to the fully documented Haskell package online[4] .

Definition 2.1 (Abstract argumentation framework)**.** An *abstract argumentation framework* is a tuple $\langle Args, Atk \rangle$, where $Args$ is a set of arguments and $Atk \subseteq Args \times Args$ is a relation on $Args$ representing attack.

The Haskell counterpart of this definition takes the form of an *algebraic data type*:

$$\textbf{data}\ DungAF\ arg = AF\ [arg]\ [(arg, arg)]$$

This is a transliteration of the mathematical definition, with lists used in place of sets. Note that the definition is parametrised on the type of argument, *arg*. Abstract arguments can be represented by strings, but we can also represent propositions or complete proof trees from a different (structured) model such as Carneades.

2.2. Carneades

Carneades is a structured argumentation model designed to capture standards and burdens of proof [12]. In previous work [9], we fully implemented the version of Carneades as given in [2] along with a small domain specific language. We review a few technical definitions that are required for Section 3.

Definition 2.2 (Carneades' 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*. Elements of $\mathcal{L}$ are 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 negative) and *con* the negation of c.

[4] `http://www.cs.nott.ac.uk/~bmv/Dung/`

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 [2], we use a dependency graph to determine acyclicity of a set of arguments.

Definition 2.3 (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 and its contrary, 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.

There are two concepts of evaluation in the Carneades model, *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. The reader is referred to the original articles [12,2] and our previous work [9] for details.

3. An algorithm and implementation for the translation of Carneades into Dung

Many of the structured approaches in argumentation can be translated into abstract models like Dung's AFs [13,1,2]. In particular, it is known that Carneades can be translated into ASPIC$^+$ [11], which in turn can be translated into AFs [15]. However, such translations, especially for models that are further removed from Dung's AFs, have rarely been *implemented*. We have taken two steps towards remedying this situation. Firstly, we give an algorithm for translating a structured model, Carneades, directly into an abstract model, Dung's AFs. Secondly, we have implemented this translation and discuss a part of it in this section. Part of this section is based on previous work in [10].

3.1. A practical algorithm for the translation of Carneades into Dung

Evaluating a Carneades model yields two results: a set of applicable arguments and a set of acceptable conclusions (Section 2.2). The target AF thus needs to include arguments representing both. Our algorithm gradually builds up Dung arguments and an attack relation, by gradually translating the applicability and acceptability part of each Carneades argument.

Algorithm 3.1. Algorithm for translation from Carneades into Dung's AFs

1. $generatedAF = \langle \{defeater\} \cup assumptions, \emptyset \rangle$.
2. $sortedArgs$ = Topological sort of *arguments* on its dependency graph.
3. **while** $sortedArgs \neq \emptyset$:
   (a) Pick the first argument in $sortedArgs$. Remove all arguments from $sortedArgs$ that have the same conclusion, c, and put them in $argSet$.
   (b) Translate applicability part of arguments in $argSet$, building on previous $generatedAF$; put generated arguments/attacks in $tempAF$.
   (c) $argSet = \emptyset$.
   (d) Repeat (a) through (c) for the arguments for opposite conclusion $\overline{c}$.

(e) Translate the acceptability part of c and $\bar{c}$ based on arguments in $tempAF$. Add the results and $tempAF$ to $generatedAF$.
(f) $tempAF = \emptyset$.

Using this algorithm we can get a one-one mapping from the union of arguments and conclusions to arguments in an AF, with the exception of one administrative node, *defeater*, that easily can be filtered out.

3.2. Step by step translation of an example

Figure 1 defines three arguments (leaving out weights and proof standards). The set of propositions $\{kill, witness, witness2, unreliable2\}$ are assumed.

Referring back to Definition 2.3, we can see that every proposition, including its negation, is present in the dependency graph for the set of three arguments defined above: see Figure 2. For reasons of presentation, we have left out the negations: all links and nodes are exactly the same for the contrary of each literal. The dependency graph makes it clear that it is necessary to translate the propositions *unreliable*, *unreliable2*, *witness* and *witness2* before *intent* and its two arguments (one pro and one con) can be translated. Figure 3 shows the resulting translation (all proposition names, including defeater, shortened to first letter).

3.3. Our implementation of the algorithm

To encode Carneades' arguments and propositions into the translated argumentation framework we could generate *String* labels from the arguments and propositions. However, we opt to instantiate the Dung AF by instead using a union of the Carneades' arguments and propositions as the framework argument.

```
type ConcreteArg = Either PropLiteral Argument
type ConcreteAF = DungAF ConcreteArg
```

Here, *Either* is a Haskell data type representing union of two data types.

For further details on the implementation, see our literate Haskell article [10] or the fully documented and open source implementation online.

4. Verification of formal properties of implementations

This section discusses two approaches to verifying the correctness of an implementation. The first is property-based testing. Given implementations of key correctness properties, tools like QuickCheck [5] can usually quickly identify any problems by picking simple counter-examples from thousands of randomly generated test cases. The second approach takes this further by formally verifying the correctness of an implementation by means of a theorem prover.

4.1. Quick testing of properties

For the translation discussed in Section 3, we can refer to existing definitions of the correspondence of applicability of arguments and acceptability of propositions (1. of Theorem 4.10 of [11]).

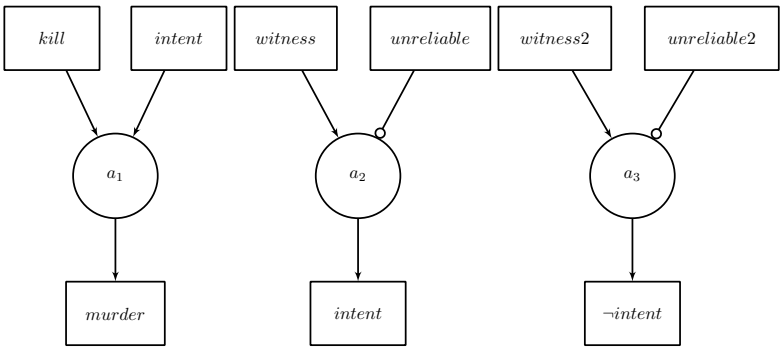

Figure 1. Three arguments in Carneades (circles denote exceptions)

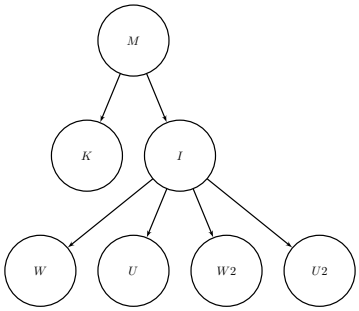

Figure 2. The dependency graph corresponding to the three arguments.

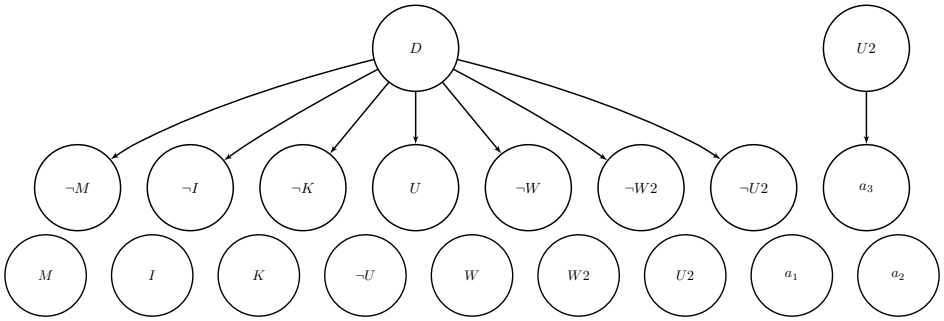

Figure 3. The corresponding Dung AF.

Theorem 4.1. *Let C be a carneades argument evaluation structure, $\langle$arguments, audience, standard$\rangle$, $\mathcal{L}_{CAES}$ the propositional language used and let the argumentation framework corresponding to C be AF. Then the following holds: An argument $a \in$ arguments is applicable in C iff there is an argument contained in the complete extension of AF with the corresponding conclusion arg_a in an AF.*

We will now sketch the implementation of the first correspondence property in Haskell. The function *corApp* takes a Carneades model and given that the translation function is a correct implementation, the Haskell implementation of correspondence of applicability should always return *True*.

$corApp :: CAES \rightarrow Bool$

```
corApp caes@(CAES (argSet, _, _)) =
  let transCAES = translate caes
      appArgs    = filter (`applicable` caes)
                          (getAllArgs argSet)
      transArgs  = stripRight (groundedExt transCAES)
  in fromList appArgs ≡ fromList transArgs
```

Here *transCAES* is the Carneades model after translation. *appArgs* are the applicable arguments in *caes* using the original definitions of applicability in the Carneades model. We then evaluate *transCAES* according to the grounded labelling (this is fine since the resulting AF is proven to be cycle-free [11]) and filter out the translated arguments using *stripRight* (discarding arguments representing propositions). The final line checks equality of the two (by making the lists into sets using *fromList*). A tool like QuickCheck can then be used to generate lots of random CAESs, and should *corApp* return *False* for any of them, a counterexample has been found. QuickCheck includes sophisticated infrastructure for tailoring the test case generation to work well also for complicated domains.

4.2. Complete formalisation in a theorem prover

While tools like QuickCheck can help finding problems automatically, firm correctness guarantees can only be obtained through formal proofs. Given that we are working in a pure, functional, strongly typed setting, theorem provers based on the Curry-Howard correspondence offer a particularly attractive approach. The idea is that types correspond to propositions and programs correspond to proofs: to prove a theorem is to implement a program having the corresponding type. We demonstrate this approach by formalising Dung's argumentation frameworks, up to grounded labelling, into Agda. Agda's syntax is very close to that of Haskell, making the step from implementation to complete formalisation relatively small. Agda checks that all functions are terminating. Thus, because we successfully implemented the grounded semantics in Agda, we immediately know that our algorithm is terminating on all (finite) inputs. Further, as a labelling is part of the output, we have actually proven that the grounded extension always exists, verifying one of Dung's original results [6].

5. Conclusions and future work

In this paper we have shown that functional programming, specifically Haskell, is very suitable for the implementation of structured and abstract models of argumentation. We gave one of the first algorithmic translations between a structured and an abstract model, implemented this, and showed how to quickly test key properties. We then took this further, taking our implementation of Dung's AFs into a theorem prover, proving termination and one of Dung's original results. Finally, we combine all this into a verified pipeline, starting from a Carneades input file, running it through our implementation of the translation, and outputting to

a file that is readable by the existing efficient implementation ASPARTIX [7]. A demonstration can be found online[5].

Future work includes extending the work on the correctness of the pipeline to complete, automatically verified proofs through a theorem prover. This would require formalising Carneades and the translation from Carneades into Dung, and then formalising correspondence properties and rationality postulates.

References

[1] 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.

[2] 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, 2010. IOS Press 2010.

[3] Federico Cerutti, Paul Dunne, Massimiliano Giacomin, and Mauro Vallati. A SAT-based approach for computing extensions in abstract argumentation. In *2nd International Workshop on Theory and Applications of Formal Argumentation (TAFA-13)*. Springer, 2013.

[4] Günther Charwat, Johannes Peter Wallner, and Stefan Woltran. Utilizing ASP for generating and visualizing argumentation frameworks. *CoRR*, abs/1301.1388, 2013.

[5] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. *Acm sigplan notices*, 46(4):53–64, 2011.

[6] 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.

[7] Wolfgang Dvořák, Matti Järvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[8] Dorian Gaertner and Francesca Toni. Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems*, 22(6):24–33, November 2007.

[9] Bas van Gijzel and Henrik Nilsson. Haskell gets argumentative. In *Proceedings of the Symposium on Trends in Functional Programming (TFP 2012), LNCS 7829*, pages 215–230, St Andrews, UK, 2013. LNCS.

[10] Bas van Gijzel and Henrik Nilsson. Towards a framework for the implementation and verification of translations between argumentation models. In *Accepted for Post Proceedings of the 25th symposium on Implementation and Application of Functional Languages (IFL 2013)*, 2014.

[11] 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.

[12] 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.

[13] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artificial Intelligence*, 195:361–397, 2013.

[14] Victor Noël and Antonis Kakas. Gorgias-c: Extending argumentation with constraint solving. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 5753 of *Lecture Notes in Computer Science*, pages 535–541. Springer Berlin Heidelberg, 2009.

[15] Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1:93–124, 2010.

[5] See: `www.cs.nott.ac.uk/~bmv/CarneadesIntoDung/Demo/`.