Dungell: A reference implementation of Dung's argumentation frameworks in Haskell

Bas van Gijzel

Functional Programming Lab
University of Nottingham
bmv@cs.nott.ac.uk

Abstract. This paper describes Dungell, an open source Haskell implementation of Dung's argumentation frameworks capable of solving decision and enumeration problems for grounded, complete, preferred, stable and semi-stable semantics. The Dungell application and its accompanying library provide an implementation that is closely aligned to the mathematical definitions, serving as a specification in its own right.

1 Introduction

The Dungell application and its accompanying library provide a Haskell implementation of Dung's argumentation frameworks [2]. The library supplies implementations for the standard definitions and semantics, including grounded, complete, preferred and stable semantics, but also conflict-freeness, admissibility and various others.

In addition to Dung's definitions, the implementation also provides implementations for the definitions and algorithms of the labelling approach to argumentation [1]. In particular, the library implements labelling algorithms and definitions for Dung's four semantics and the semi-stable semantics.

2 Design philosophy

There are various other efficient solvers that exist for AFs, which are most likely faster than Dungell. Instead, the approach taken in the implementation of Dungell, is to provide an implementation that is as clear and close to the mathematical definitions as possible. The main reasons this approach is taken, is that:

- the library becomes intuitive, reproducible and easier to verify;
- the implemented definitions can more easily be converted to and proven correct in a theorem prover;
- the library can be used as a translation target.

The library also provides output formats readable by the current fastest implementations.

The combination of these features is intended to allow implementers of structured argumentation models to use Dungell to implement a translation from a structured model into Dungell, possibly performing formal verification as well. Further details of the implementation and the general design can be found in previous papers [3, 4, 5, 6].

3 Implementation of the enumeration of complete and preferred extensions

This section gives an illustration of the approach taken in Dungell by demonstrating how the enumeration of complete and preferred extensions are implemented in (a slightly stylised version of) Haskell. The code in this section is almost self-contained[1].

An abstract argumentation framework consists of a set of arguments and a binary relation on these arguments, representing *attack*. The Haskell counterpart of this definition takes the form of an *algebraic data type*:

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

Note how this essentially is a transliteration of the mathematical definition, even if lists are used in place of sets. Additionally, the definition is parametrised on the type of argument, arg.

Fig. 1. An (abstract) argumentation framework

Given an argumentation framework using labels as the type of abstract argument we can represent this in Haskell using *String*s.

$$\textbf{type}\ \mathit{AbsArg} = \mathit{String}$$
$$a, b, c :: \mathit{AbsArg}$$
$$a = \texttt{"A"};\ b = \texttt{"B"};\ c = \texttt{"C"}$$
$$\mathit{AF}_1 :: \mathit{DungAF}\ \mathit{AbsArg}$$
$$\mathit{AF}_1 = \mathit{AF}\ [a, b, c]\ [(a, b), (b, c)]$$

Haskell can define functions by pattern matching on the shape of an algebraic data type, splitting the definition into multiple lines for each type of shape. For example, the powerset function on lists, can be recursively defined as follows:

$$\begin{aligned}&\mathit{powerset} :: [a] \to [[a]]\\ &\mathit{powerset}\ [\,] \quad\ = [[\,]]\\ &\mathit{powerset}\ (x : \mathit{xs}) = \mathit{powerset}\ \mathit{xs} +\!\!+ \mathit{map}\ (x:)\ (\mathit{powerset}\ \mathit{xs})\end{aligned}$$

[1] The complete source code of this section can be run by the Haskell compiler and can be downloaded at: `http://www.cs.nott.ac.uk/~bmv/Code/dungell_iccma.lhs`.

Given an argumentation framework for which we can check whether arguments are equal ($Eq\ arg$), it is possible to verifiy whether a list of arguments is conflict-free by checking that the list of attacks between those arguments is empty by using *null*.

$$
\begin{aligned}
&conflictFree :: Eq\ arg \Rightarrow DungAF\ arg \to [arg] \to Bool \\
&conflictFree\ (AF\ \_\ def)\ args \\
&\quad = null\ [(a, b) \mid (a, b) \leftarrow def, a \in args, b \in args]
\end{aligned}
$$

Acceptability of an argument w.r.t. a set of arguments in an AF can be determined by verifying that all its attackers are in return attacked by an attacker in that set. The call $setAttacks\ af\ args\ b$ returns *True* if any argument in the set *args* attacks the argument b; the *and* function below is being equivalent to $\bigwedge$.

$$
\begin{aligned}
&acceptable :: Eq\ arg \Rightarrow DungAF\ arg \to arg \to [arg] \to Bool \\
&acceptable\ af@(AF\ \_\ def)\ a\ args \\
&\quad = and\ [setAttacks\ af\ args\ b \mid (b, a') \leftarrow def, a \equiv a']
\end{aligned}
$$

The characteristic function of an argumentation framework, calculates the set of arguments acceptable w.r.t. a given set of arguments. Admissibility of a set of arguments is then determined by verifying that the set is conflict-free and a subset of the characteristic function applied to that set. The $Ord\ arg$ requires the argument type to be comparable, implying the existence of an equality ($Eq\ arg$) between arguments.

$$
\begin{aligned}
&f :: Eq\ arg \Rightarrow DungAF\ arg \to [arg] \to [arg] \\
&f\ af@(AF\ args'\ \_)\ args = [a \mid a \leftarrow args', acceptable\ af\ a\ args] \\
&admissible :: Ord\ arg \Rightarrow DungAF\ arg \to [arg] \to Bool \\
&admissible\ af\ args = conflictFree\ af\ args \wedge args \subseteq f\ af\ args
\end{aligned}
$$

Given an argumentation framework, the set of complete extensions can be calculated by taking all sets of arguments of the powerset of arguments of that AF, given that they are admissible and $f_{AF}\ x \equiv x$.

$$
\begin{aligned}
&completeF :: Ord\ arg \Rightarrow DungAF\ arg \to [[arg]] \\
&completeF\ af@(AF\ args\ \_) = \\
&\quad \textbf{let}\ f_{AF} = f\ af \\
&\quad \textbf{in}\ filter\ (\lambda x \to admissible\ af\ x \wedge f_{AF}\ x \equiv x)\ (powerset\ args)
\end{aligned}
$$

Finally, the set of preferred extensions can be obtained by applying an appropriate filter on the complete extension, i.e. a complete extension is also a preferred extension if it is *not* a subset of one the other complete extensions.

$$
\begin{aligned}
&isPreferredExt :: Ord\ arg \Rightarrow DungAF\ arg \to [[arg]] \to [arg] \to Bool \\
&isPreferredExt\ af\ exts\ ext = all\ (\neg \circ (ext \subseteq))\ (delete\ ext\ exts) \\
&preferredF :: Ord\ arg \Rightarrow DungAF\ arg \to [[arg]] \\
&preferredF\ af@(AF\ args\ \_) = \\
&\quad \textbf{let}\ cs = completeF\ af \\
&\quad \textbf{in}\ filter\ (isPreferredExt\ af\ cs)\ cs
\end{aligned}
$$

4 Installation and usage instructions

The source code of the Dungell application is available under an open source license (BSD3). It can be downloaded and installed:

- as a bundled zip file, including external libraries at: `http://www.cs.nott.ac.uk/~bmv/Code/dungell_bundled.zip`;
- or as a Haskell library at: `https://github.com/nebasuke/DungICCMA`.

The user is required to install a recent Haskell distribution (GHC 7.8.4 or higher for the bundled zip file). After downloading the source files the the Dungell executable can be compiled by using **cabal install** in the toplevel directory containing the Dungell.cabal file. Detailed usage and installation instructions can be found at: `www.cs.nott.ac.uk/~bmv/DungICCMA/`.

Acknowledgements

I would like to thank Tom Gordon for his feedback during the development of the implementation.

References

[1] Martin Caminada. An algorithm for computing semi-stable semantics. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 222–234. Springer, 2007.

[2] 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.

[3] Bas van Gijzel. Tools for the implementation of argumentation models. In Andrew V. Jones and Nicholas Ng, editors, *2013 Imperial College Computing Student Workshop*, volume 35 of *OpenAccess Series in Informatics (OASIcs)*, pages 43–48, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] 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.

[5] Bas van Gijzel and Henrik Nilsson. A principled approach to the implementation of argumentation models. In *Proceedings of the Fifth International Conference on Computational Models of Argument (COMMA 2014)*, pages 293–300. IOS Press, 2014.

[6] Bas van Gijzel and Henrik Nilsson. Towards a framework for the implementation and verification of translations between argumentation models. In *Proceedings of the 25th Symposium on Implementation and Application of Functional Languages*, IFL '13, pages 93:93–93:103, New York, NY, USA, 2014. ACM.