Carneades ICCMA: A Straightforward Implementation of a Solver for Abstract Argumentation in the Go Programming Language

Thomas F. Gordon

Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Abstract. The Carneades entry to the 2015 International Competition on Computational Models of Argument (ICCMA) is a straight-forward implementation of a solver for reasoning tasks in abstract argumentation frameworks [1]. All of the reasoning tasks (computing one or all extensions and deciding whether an argument is credulously or skeptically inferred) and Dung semantics (grounded, complete, preferred, stable) covered by the competition have been implemented. The solver has been implemented in Go, a mainstream statically-typed, procedural programming language with a C-like syntax, garbage collection, good builtin support for concurrency and a large standard library. The aim is to provide a concise and readable implementation suitable for pedagogical purposes, using only common and widely familiar programming constructs and with no dependencies on external libraries or programs.

Keywords: abstract argumentation frameworks, computational models of argument, grounded semantics

1 Introduction

The Carneades entry to the 2015 International Competition on Computational Models of Argument (ICCMA) is a straight-forward implementation of a solver for reasoning tasks in abstract argumentation frameworks [1]. While part of the open source Carneades project[1], this code was newly developed specifically for the ICCMA competition. The focus of the Carneades project has not been abstract argumentation, but rather *structured* argumentation. Nonetheless, Carneades has for some time included an implementation of a solver for Dung abstract argumentation frameworks, using grounded semantics. This solver was implemented to overcome a limitation of the original computational model of Carneades [2], which did not allow argument graphs to contain cycles, by mapping Carneades argument graphs to abstract argumentation frameworks in a manner similar to ASPIC+ [4].

[1] `https://carneades.github.io/`

The Carneades entry for the ICCMA competition implements all of the reasoning tasks (computing one or all extensions and deciding whether an argument is credulously or skeptically inferred) for all the semantics of abstract argumentation frameworks (grounded, complete, preferred, stable) covered by the competition.

2 System Architecture

The solver has been implemented in Go [6], a mainstream statically-typed, procedural programming language with a C-like syntax, garbage collection, good builtin support for concurrency and a large standard library.

The aim is to provide a concise and readable implementation suitable for pedagogical purposes, using only common and widely familiar programming constructs. Although the implementation of the tractable problems, using grounded semantics, is quite efficient, the simple generate-and-test algorithms implemented for the combinatorial problems are not expected to perform well compared to entries based on highly-optimized SAT and ASP solvers.

The implementation closely follows high-level specifications of abstract argumentation frameworks [5] and has not been optimized in any significant way, with perhaps one exception: The implementation of grounded semantics keeps track of whether a mutable labelling has changed, in its main loop, and exits the loop when no changes were made, without having to explicitly test whether two labellings are equivalent.

For the combinatorial problems, for complete, preferred and stable semantics, the solver generates and tests all subsets of the power set of the arguments in the framework. The subsets are generated using an algorithm found on the Web.[2] Using this algorithm, every subset is generated and visited exactly once.

Using Go's support for first-class and higher-order functions, procedures were implemented for finding the first subset of arguments which satisfy a given predicate and for applying some procedure to each subset. Using functions implementing predicates for complete and stable extensions, it is then simple to find the first or all complete extensions and then to filter the complete extensions to find one or more which are also stable.

Implementing preferred semantics was only a bit more difficult. While iterating over the complete extensions, a list of candidate extensions is maintained. When a new complete extension is found, it replaces every candidate which is a subset of the new extension in the list of candidates.

3 Lessons Learned and Future Work

For preferred semantics, we first tried the algorithm in [3], but found that it performed much worse than the straightforward generate-and-test algorithm we

[2] `http://www.stefan-pochmann.info/spots/tutorials/sets_subsets/`

settled on in the end. The algorithm in [3] appears to visit each member of the powerset of arguments in the framework more than once.

Argument sets are currently represented as hash tables, from arguments to boolean values. Simulating immutable operations on sets involves creating copies of the hash tables. An obvious first step toward improving the performance of this implementation would be to replace this representation with an immutable, persistent representation of argument sets optimized for subset and equality comparisons.

The Go programming language provides excellent support for concurrent algorithms, but we did not make use of this feature, since it is not yet clear to us how to refactor the program to make good use of concurrency. We considered implementing the generate and test procedures as separate tasks, communicating via a channel. We may try this, but expect that the testing task will not be able to keep up with the generation task, which takes at most only a few seconds, and thus not divide up the work well among the available processors.

We are pleased with the performance of the implementation of grounded semantics, which seems to be a bit faster than any other implementation we are familiar with, in our informal and unsystematic benchmark tests. Of course, the problems are all tractable when using grounded semantics, so this may not be especially interesting in the context of the ICCMA. Surely the combinatorial problems are more challenging and interesting for the purpose of the competition. However, in our experience grounded semantics is well suited for many practical applications of argumentation frameworks, so an efficient implementation may be of some interest for developers of such applications.

Our implementation of the combinatorial problems is not nearly as efficient as the Tweety reference implementation [7], in Java, although Go and Java have comparable performance. On the other hand, our implementation in Go is considerable shorter and more readable, presumably due to Go being a less verbose programming language. That said, in the future we may try to optimize our implementation, learning from the Tweety implementation.

4 Downloading and Installing the Code

The source code of the Carneades ICCMA entry is available on Github at `https://github.com/carneades/carneades-4`.

Prerequisites for building the system are:

- Go, `http://golang.org/`, and
- Git, `http://git-scm.com/`

To build the system:

Set the `GOPATH` environment variable to a directory for Go packages, e.g.

```
$ mkdir ~/go
$ typeset -x GOPATH=~/go
```

Use the `go` tool to get, build and install the `carneades-iccma` executable from Github:

```
$ go get github.com/carneades/carneades-4/internal/cmd/carneades-iccma
```

The `carneades-iccma` executable should now be installed in

```
$GOPATH/bin/carneades-iccma
```

You can execute the program using this full path. Alternatively, add `$GOPATH/bin` to your `PATH` environment. You can then execute the command directly, as in

```
$ carneades-iccma -p EE-GR -f ...
```

These instructions are subject to change. See the `INSTALL.md` file for current instructions.

References

1. Dung, P.M.: On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. Artificial Intelligence 77(2), 321–357 (1995)
2. Gordon, T.F., Prakken, H., Walton, D.: The Carneades Model of Argument and Burden of Proof. Artificial Intelligence 171(10-11), 875–896 (2007)
3. Modgil, S., Caminada, M.: Proof Theories and Algorithms for Abstract Argumentation Frameworks. In: Rahwan, I., Simari, G.R. (eds.) Argumentation in Artificial Intelligence, pp. 105–129. Spinger (2009)
4. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument & Computation 1, 93–124 (2010)
5. Rahwan, I., Simari, G.R.: Argumentation in Artificial Intelligence. Springer Verlag (2009)
6. Summerfield, M.: Programming in Go Creating Applications for the 21st Century. Addison-Wesley (2014)
7. Thimm, M.: Tweety-a comprehensive collection of Java libraries for logical aspects of artificial intelligence and knowledge representation. In: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR14) (2014)