

ArgTools: a backtracking-based solver for abstract argumentation

Samer Nofal¹, Katie Atkinson² and Paul E. Dunne²

¹ Computer Science Department, German Jordanian University, Jordan

² Computer Science Department, University of Liverpool, United Kingdom

Abstract. We present ArgTools, a system for reasoning with abstract argumentation frameworks. The system solves a number of argumentation problems under preferred, stable, complete and grounded semantics. ArgTools is a C++ implementation of a backtracking algorithm.

Keywords: algorithms, argumentation semantics, automated reasoning.

1 Introduction

Abstract argumentation frameworks (AFs), introduced in [4], are an important model of automated reasoning [2]. An AF is a pair (A, R) where A is a set of abstract arguments and $R \subseteq A \times A$ is a binary relation. Argumentation semantics are concerned with defining the acceptable arguments in a given AF. There are a number of semantics for different motivations, see [1] for an overview. Several problems related to argumentation semantics are computationally hard [5]. Algorithms for solving these problems can be either direct or indirect [3]. Indirect approaches are reduction-based methods such that the problem at hand is translated to another form to be solved by an off-the-shelf system. Direct approaches are dedicated algorithms that search for a solution to the input AF. In this paper we present ArgTools (short for Argumentation Tools), a system based on backtracking algorithms for solving problems under *preferred*, *stable*, *complete* and *grounded* semantics. In section 2 we give the definition of these semantics and specify the problems solved by ArgTools. Then, in section 3 we discuss the underlying approach of ArgTools. Lastly, in section 4 we conclude the paper.

2 Problems solved by ArgTools

We recall the definition of AFs, originally introduced in [4]. An *argumentation framework* (or AF) is a pair (A, R) where A is a set of arguments and $R \subseteq A \times A$ is a binary relation, see figure 1 for an example AF. We refer to $(x, y) \in R$ as x attacks y (or y is attacked by x). We denote by $\{x\}^-$ respectively $\{x\}^+$ the subset of A containing those arguments that attack (resp. are attacked by) the argument x .

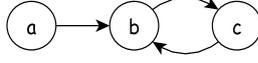


Fig. 1. An AF, as a directed graph, with $A = \{a, b, c\}$ and $R = \{(a, b), (b, c), (c, b)\}$.

Given a subset $S \subseteq A$, then

- $x \in A$ is *acceptable* w.r.t. S if and only if for every $(y, x) \in R$, there is some $z \in S$ for which $(z, y) \in R$.
- S is *conflict free* if and only if for each $(x, y) \in S \times S$, $(x, y) \notin R$.
- S is *admissible* if and only if it is conflict free and every $x \in S$ is acceptable w.r.t. S .
- S is a *preferred extension* if and only if it is a \subseteq -maximal admissible set.
- S is a *stable extension* if and only if it is conflict free and for each $x \notin S$ there is $y \in S$ such that $(y, x) \in R$.
- S is a *complete extension* if and only if it is an admissible set such that for each x acceptable w.r.t. S , $x \in S$.
- S is the *grounded extension* if and only if it is the \subseteq -least complete extension.

ArgTools solves problems under preferred, stable, complete and grounded semantics. The problems are:

- Given an AF $H = (A, R)$, ArgTools enumerates all extensions of H .
- Given an AF $H = (A, R)$, ArgTools finds an extension of H .
- Given an AF $H = (A, R)$ and an argument $a \in A$, ArgTools decides whether a is in some extension of H .
- Given an AF $H = (A, R)$ and an argument $a \in A$, ArgTools decides whether a is in all extensions of H .

3 The approach of ArgTools

To give an idea about the approach of ArgTools we present algorithm 1 that enumerates all preferred extensions of a given AF. The algorithm is a backtracking procedure that traverses an abstract binary search tree. A core notion of the algorithm is related to the use of five labels: *IN*, *OUT*, *MUST_OUT*, *BLANK* and *UNDEC*. Informally, the *IN* label identifies arguments that might be in a preferred extension. The *OUT* label identifies an argument that is attacked by an *IN* argument. The *BLANK* label is for any unprocessed argument whose final label is not decided yet. The *MUST_OUT* label identifies arguments that attack *IN* arguments. The *UNDEC* label designates arguments which might not be included in a preferred extension because they might not be defended by any *IN* argument. To enumerate all preferred extensions algorithm 1 starts with *BLANK* as the default label for all arguments. This initial state represents the root node of the search tree. Then the algorithm forks to a left (resp. right) child (i.e. state)

by picking an argument, that is BLANK, to be labeled IN (resp. UNDEC). Every time an argument, say x , is labeled IN some of the neighbour arguments' labels might change such that for every $y \in \{x\}^+$ the label of y becomes OUT and for every $z \in \{x\}^- \setminus \{x\}^+$ the label of z becomes MUST_OUT. This process, i.e. forking to new children, continues until there is no argument with the BLANK label. At this point, the algorithm captures the set $S = \{x \mid \text{the label of } x \text{ is IN}\}$ as a preferred extension if and only if there is no argument with the MUST_OUT label and S is not a subset of a previously found preferred extension (if such exists). Then the algorithm backtracks to find all preferred extensions. Figure 2 shows how algorithm 1 lists the preferred extensions of the AF of figure 1.

Algorithm 1: Enumerating all preferred extensions of an AF $H = (A, R)$.

```

1  $Lab : A \rightarrow \{IN, OUT, BLANK, MUST\_OUT, UNDEC\}; Lab \leftarrow \emptyset;$ 
2 foreach  $x \in A$  do  $Lab \leftarrow Lab \cup \{(x, BLANK)\};$ 
3 foreach  $(x, x) \in R$  do  $Lab(x) \leftarrow UNDEC;$ 
4  $E \subseteq 2^A; E \leftarrow \emptyset;$ 
5 call build-preferred-extensions( $Lab$ );
6 report  $E$  is the set of all preferred extensions;

7 procedure build-preferred-extensions( $Lab$ )
8   if  $\nexists x$  with  $Lab(x) = BLANK$  then
9     if  $\nexists x$  with  $Lab(x) = MUST\_OUT$  then
10        $S \leftarrow \{y \mid Lab(y) = IN\};$ 
11       if  $\forall T \in E S \not\subseteq T$  then  $E \leftarrow E \cup \{S\};$ 
12     else
13       select any  $x$  with  $Lab(x) = BLANK;$ 
14        $Lab' \leftarrow Lab; Lab'(x) \leftarrow IN;$ 
15       foreach  $y \in \{x\}^+$  do  $Lab'(y) \leftarrow OUT;$ 
16       foreach  $y \in \{x\}^- \setminus \{x\}^+$  do  $Lab'(y) \leftarrow MUST\_OUT;$ 
17       call build-preferred-extensions( $Lab'$ );
18        $Lab' \leftarrow Lab; Lab'(x) \leftarrow UNDEC;$ 
19       call build-preferred-extensions( $Lab'$ );
20 end procedure

```

4 Conclusion

ArgTools has been coded in the C++ language; the source code and the usage are available at <http://sourceforge.net/projects/argtools/>. For space limitation we did not discuss (in full detail) the underlying algorithms of ArgTools; for the full presentation of the algorithms we refer the reader to [7, 6]. However, ArgTools incorporates new enhancements that we plan to present in future in an extended article.

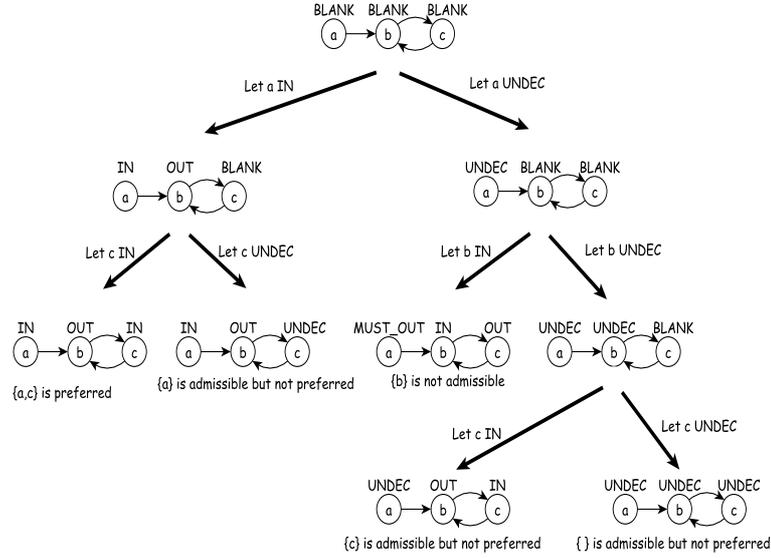


Fig. 2. Listing the preferred extensions of an AF using algorithm 1.

5 Acknowledgment

The first author, Samer Nofal, is supported by the Deanship of Scientific Research at the German Jordanian University (project number: SIC 18/2015).

References

1. Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.
2. Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.
3. Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.
4. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
5. Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artif. Intell.*, 171(10-15):701–729, 2007.
6. Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for argumentation semantics: Labeling attacks as a generalization of labeling arguments. *J. Artif. Intell. Res. (JAIR)*, 49:635–668, 2014.
7. Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.*, 207:23–51, 2014.