ArgQL: A Declarative Language for Querying Argumentative Dialogues

Dimitra Zografistou, Giorgos Flouris, Dimitris Plexousakis

Foundation for Research and Technology
{dzograf,fgeo,dp}@ics.forth.gr

Abstract. We introduce ArgQL, a declarative query language, which performs on a data model designed according to the principles of argumentation. Its syntax is based on Cypher (language for graph databases) and SPARQL 1.1 and is adjusted for querying dialogues, composed by sets of arguments and their interrelations. We use formal semantics to show how queries in ArgQL match against data in the argumentation model. The execution is realized by translating both data and queries into standard models for storage and querying.

1 Introduction

The recent evolution of social media and debate forums has caused a reshaping of the Web, turning it into a worldwide podium, wherein humans accommodate their inherent need for socializing and expressing themselves. Due to the easiness with which users can upload digital content, as well as the ability for their involvement in public debates[1], online communities have become populated with opinions and beliefs about political or social topics, with criticisms or consultations and with reviews on products, services etc.

The process of human argumentation has been an object of longstanding theoretical studies, which have found their way into computational models [7],[13] of the area of argumentation. Roughly, these models address the representation and reasoning requirements of drawing conclusions through the process of exchanging arguments [9], with part of them to also be taking into account relevant cognitive features like preferences [1], beliefs and intentions [2]. However, the process of identifying arguments according to certain criteria and allowing for navigation in a graph of interconnected arguments has been given less attention.

To cover this gap, we introduce *ArgQL* (Argumentation Query Language) a declarative language, designed on a data model for argumentation. ArgQL constitutes a first step to understand the informational and theoretical requirements of searching in debates. Yet, there are still miscellaneous obstacles until being able to query real debates on the web, with processing textual information to be the most difficult to overcome. The fact that there is a considerable amount of people involved in argumentation, amplifies the significance for a language with relevant terminology. ArgQL offers a simple and quite elegant way to express

[1] http://www.debate.org

queries of the form: ”How an argument with a given conclusion is attacked?”. To the best of our knowledge, this is the first effort to approach argumentation from the scope of data models and query languages.

In this document, we show some initial results. The requirements we set for ArgQL, led us use a hybrid syntax, with features from both SQL-like and graph database query languages. The semantics of ArgQL are also presented. As for the execution, we currently deal with the technical as well as the theoretical issues lying in the translation of the data model and ArgQL, into RDF and SPARQL.

2 Related Work

There are no equivalent languages to directly compare the potentials of ArgQL. Nevertheless, considerable efforts have been invested, to make steps closer to the realization of the vision for a globally interconnected and computable web of opinions[3][8]. A wide number of tools have been developed, that facilitate the participation in online debates. A comprehensive review is found in[15]. Some of them offer better visualization and exploration, such as DebateGraph[2], others support reasoning like Parmenides[5], while others allow for user engagement, like Debatewise[3].

AIF (Argument Interchange Format)[6] is an RDF ontology for arguments and is considered to be the cornerstone to the realization of the opinion web. Accompanied with a set of mappings from the different argument representations [16],[17],[4] to its concepts, AIF became the interlingua, which bridges arguments among the various tools. These data are gathered into a public database AIFdb[11] and queried by several search engines, like ArgDF[14]. DiscourseDB[4], is another equally interesting platform for exploring dialogues, with the extra feature of dialogues summarization.

To sum up, there is a wide number of systems that offer visualization and semantic search in dialogues. For all of them, the process of querying dialogues, merely constitutes an application field for the traditional query languages like SQL or SPARQL. On the other hand, ArgQL highlights the different information needs of such a process and focuses on designing solutions, that cover these requirements. In the future, it could be integrated into the existing argumentation tools enriching them with extra semantics and capabilities.

3 A formal model for arguments

Our argumentation model is defined by the tuple $\mathcal{L} = (\mathcal{P}, -, \simeq, \rightarrow)$. $\mathcal{P}$ is an infinite set of propositions and $\wp(\mathcal{P})$ its powerset.

The mapping $- : \mathcal{P} \rightarrow \mathcal{P}$, represents the notion of *contrariness*. We say that, two propositions $x, y \in \mathcal{P}$ are conflicting iff $x = \overline{y}$. Contrariness mapping is

[2] http://debategraph.org/
[3] http://debatewise.org/
[4] http://discoursedb.org/

symmetric (if $x = \overline{y}$, then $y = \overline{x}$), anti-reflexive ($x \neq \overline{x}$) and non transitive (if $x = \overline{y}$ and $y = \overline{z}$, with $x \neq z$, then $x \neq \overline{z}$). Given a subset $P' \subset \mathcal{P}$, we say that P' is *inconsistent* if $\exists x, y \in P'$, s.t. $y = \overline{x}$, otherwise it is *consistent*.

The mapping $\simeq \subseteq \mathcal{P} \times \mathcal{P}$ captures the *equivalence* on the informational content between two propositions $x, y \in \mathcal{P}$ and we write it as $x \simeq y$. Equivalence retains the following properties: *(i)Reflexivity:* $x \simeq x$ *(ii)Symmetry:* if $x \simeq y$, then $y \simeq x$ *(iii)Transitivity:* if $x \simeq y$ and $y \simeq z$, then $x \simeq z$.

Mappings $-$ and $\simeq$ will be later used to define the relations between arguments. Given that they serve opposite representation needs, in order to resolve the ambiguities caused by their coexistence in the same model, we demand the satisfaction of the following constraints: $\forall (x, y, z) \in \mathcal{P}$ *(i)* if $x \simeq y$ then $y \neq \overline{x}$. *(ii)*if $x \simeq y$, then $\overline{x} \simeq \overline{y}$ and *(iii)* if $x \simeq y$ and $y = \overline{z}$, then $x = \overline{z}$

Finally, with $\rightarrow$ we denote the deduction from a set of propositions to a conclusion. For example, when we write "$a, b \rightarrow c$", we say that propositions a, b imply proposition c. Such expressions form *arguments*. In particular:

Definition 1 (Argument). *An argument is a tuple $a = \langle pr, c \rangle$, where $pr \in \wp(\mathcal{P})$ is a finite, consistent set, called premise set and $c \in \mathcal{P}$ is the main claim of an argument, called conclusion. It holds that $pr \rightarrow c$ and $c \notin pr$. With $\mathcal{A}$ we refer to the infinite set of arguments.*

The notions of contrariness and equivalence between propositions, allow us to define two types of relations between arguments, *attack* and *support*, respectively. The infinite set $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ keeps any relation between two arguments. Given a relation $r = (a_1, a_2) \in R$, we denote by $src(r) = a_1$ the source argument of r and by $dst(r) = a_2$ its destination.

Definition 2 (Attack). *Attack relation is defined as: $R^a = \{(a_1, a_2) \mid a_1, a_2 \in \mathcal{A}$, $a_1 = \langle pr_1, c_1 \rangle$, $a_2 = \langle pr_2, c_2 \rangle$ and $c_1 = \overline{c_2}$ (rebut) or $\overline{c_1} \in pr_2$ (undercut)$\}$*

Definition 3 (Support). *Support relation is defined as: $R^s = \{(a_1, a_2) \mid a_1, a_2 \in \mathcal{A}$, $a_1 = \langle pr_1, c_1 \rangle$, $a_2 = \langle pr_2, c_2 \rangle$ and $c_1 \simeq c_2$ (endorse) or $\exists p \in pr_2$, s.t. $p \simeq c_1$ (backing)$\}$*

It holds that $R = R^a \cup R^s$. It is also provable that $R^a \cap R^s = \emptyset$, but for the sake of space, the proof is omitted. A knowledge base conformed to the current model has the form of a graph and we call it debate graph. In particular:

Definition 4 (Debate graph). *A debate graph is a tuple $D{=}(A', R')$, where $A' \subset \mathcal{A}$ is the set of argument nodes, while the set of edges R', holding the relations between arguments in A', is defined as $R' = \{(a_1, a_2) | a_1, a_2 \in A'$ and $(a_1, a_2) \in \mathcal{R}\}$.*

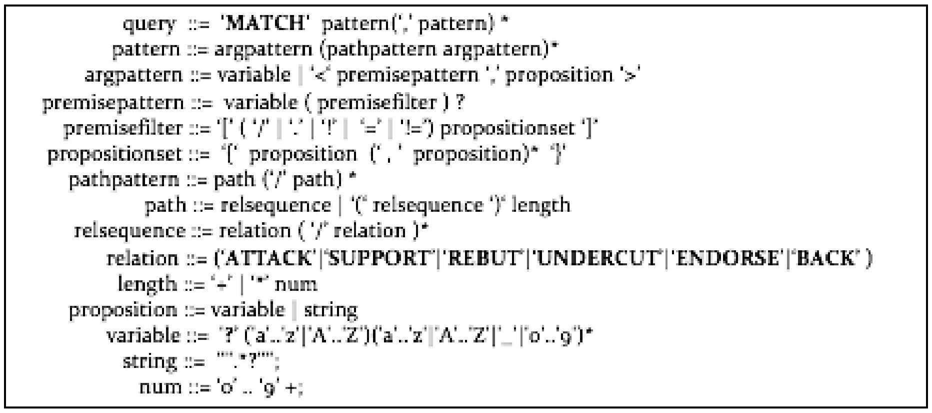
The infinite set of all possible debate graphs is denoted as $\mathcal{D}$. We define any *path* in a debate graph as follows:

Definition 5 (Path). *Given a debate graph $D = \langle A', R' \rangle$, and two arguments $a, b \in A'$, a path between a and b denoted as $P^D_{a \rightarrow b}$ is a sequence of relations $r_1, r_2 \ldots, r_n \in R'$, where $n \geq 1$, $src(r_1) = a$, $dst(r_n) = b$ and $\forall i \in \{1, .., n-1\}$ it holds that $dst(r_i) = src(r_{i+1})$. A path is written as $r_1/r_2/\ldots/r_n$.*

4 ArgQL Specification

4.1 Syntax

The syntax of ArgQL is Cypher-like[5], a language for the Neo4j graph database. The main idea is that it supports pattern expressions matching against data in a knowledge base, designed according to the model described in Section 3. Like any query language, ArgQL uses variables to bind data and they are prefixed with ?. Let $\mathcal{V}$ be the infinite set of variables. We assume the set $PV = \mathcal{P} \cup \mathcal{V}$. Constant values for the propositions in $\mathcal{P}$ are represented as literals and they are given in quotes. Figure 4.1 shows the BNF grammar. Note that, at this stage of our work, we have not studied how the results of the query are returned, thus the syntax does not contain any *select*-like statement, yet. In the future, we intent to allow for returning more complex structures, such as complete parts of dialogues. Reserved keywords of the language are in bold. They are not case sensitive, but for the sake of space the respective rules have been omitted.



```
          query ::= 'MATCH' pattern(',' pattern) *
        pattern ::= argpattern (pathpattern argpattern)*
     argpattern ::= variable | '<' premisepattern ',' proposition '>'
 premisepattern ::=  variable ( premisefilter ) ?
  premisefilter ::= '[' ( '/' | '.' | '!' |  '=' | '!=') propositionset ']'
 propositionset ::=  '{' proposition (' , ' proposition)*  '}'
    pathpattern ::= path ('/' path) *
           path ::= relsequence | '(' relsequence ')' length
    relsequence ::= relation ( '/' relation )*
       relation ::= ('ATTACK'|'SUPPORT'|'REBUT'|'UNDERCUT'|'ENDORSE'|'BACK' )
         length ::= '+' | '*' num
    proposition ::= variable | string
       variable ::=  '?' ('a'..'z'|'A'..'Z')('a'..'z'|'A'..'Z'|'_'|'o'..'9')*
         string ::=  '"'.*?'"';
            num ::= 'o' .. '9' +;
```

Fig. 1. Extended BNF for ArgQL

The main body of the query is included inside the *match* clause. ArgQL allows to state multiple pattern expressions. A single pattern expression(rule *pattern*) consists either of one argument pattern, or it can be followed by a sequence of alternations between path patterns and another argument pattern. Next we will discuss separately about the role of argument patterns and path patterns.

Argument patterns. They constitute fundamental elements of ArgQL, used to match arguments. Generally, argument patterns represent expressions like: find arguments with conclusion "some text", written as $\langle ?pr,\ \text{"}some\ text\text{"}\rangle$, or find arguments with the proposition "some text" in their premise set, that corresponds to the argument pattern $\langle ?pr[/\{\text{"}sometext\text{"}\}],\ ?\mathrm{c}\ \rangle$.

Typically an argument pattern *ap* can be either a single variable, which matches any argument in $\mathcal{A}$, or be of the form $ap = \langle pr([filter])?,\ c\rangle$, where

[5] https://neo4j.com/developer/cypher-query-language/

$pr \in V$ and $c \in PV$. Variable pr matches the premise part of arguments and essentially it takes values from $\wp(\mathcal{P})$, whereas c matches the conclusion part and may be either a variable or a constant proposition value. The occurrence of the expression $[filter]$ is optional. When existed, it adds constraints on the premises. ArgQL supports a number of filters, that correspond to such constraints like the requirement for a premise set to include some given propositions, or the premise sets of two arguments to join etc. Later versions of the language will allow for multiple filters on a premise part, but for now, we consider only a single one. The following list presents the available filters. To formally describe them, we assume the set s included in the filter, with $s = \{p_1, \dots p_n\}$, $p_i \in PV$ and $1 \leq i \leq n$.
Inclusion: $f_{incl} = \big[/s\big]$ Requirement for s to be included in pr.
Jointness: $f_{join} = \big[.s\big]$ The sets pr and s must have common elements.
Disjointness: $f_{disj} = \big[!s\big]$ The sets pr and s must be disjoint.
Equality: $f_{eq} = \big[= s\big]$ The sets pr and s must be exactly the same.
Inequality: $f_{ineq} = \big[! = s\big]$ The set pr must be different than s.

A matching instance of an argument pattern is an argument in $\mathcal{A}$. The cases where an argument pattern is a single variable, or it has the form $\langle ?pr, ?c\rangle$ are equal and they both match any argument in the knowledge base.

Path patterns. They are used to describe sequences of arguments that must be matched (and the relations that connect them). A match of such an expression is essentially a sub-graph from the debate graph. We exploit the syntax of SPARQL 1.1[10] for property paths. In particular, a path pattern may indicate a path in two ways. The direct one represents a path pattern as a sequence of relations separated by the symbol '/' (e.g. *attack/undercut/support/*...). The indirect uses one of the '+', '*' numerical indicators to restrict the length of the path. Expression *path'+'* means one or more occurrences of *path* while the expression *path'*'n*, with $n \geq 1$, requires for exactly n repetitions of *path*. These two ways can be mixed in arbitrary ways to support many complex types of path patterns.

Next, we give some representative examples of the supported match expressions. The first captures a case from Dung semantics [7].

Q1. Find arguments which "defend"(attack their attackers) all arguments with conclusion "cloning is immoral".
`match ?arg (attack/attack)+ <?pr, "cloning is immoral">`

Q2. Match argument paths which include two sequences of the path attack/support and one more attack and result to an argument, whose premise set includes the proposition "cloning contributes positively to the field of artificial insemination".
`match ?arg (attack/support)*2/attack <?pr[/{"cloning contributes positively to the field of artificial insemination"}], ?c>`

Q3. Match pairs of arguments whose premise sets join each other.
match $<?pr_1,\ ?c_1>,\ <?pr_2[.\{?pr_1\}],\ ?c_2>$

4.2 Semantics

In this section we describe the semantics of ArgQL. We define the set $\mathcal{S} = \mathcal{P} \cup \wp(\mathcal{P}) \cup \mathcal{A}$ and a *mapping function* $\mu : \mathcal{S} \cup V \mapsto \mathcal{S}$, such that $\mu(x) = x, \forall x \in \mathcal{S}$. For simplicity we will abuse notation and use μ also for argument patterns. For example, given an argument pattern $ap = \langle pr[filter], c\rangle$, we denote by $\mu(ap)$ the argument obtained by the substitution of pr with $\mu(pr)$ and c with $\mu(c)$, for which, $\mu(pr)$ satisfies its filter if existed.

For the satisfiability of the filter f by the premise set, we restrict ourselves to those mappings for pr, for which $\mu(pr) \subset \wp(\mathcal{P})$. We write $\mu(pr) \vDash f$ to describe that the matching set of propositions satisfies f. In order to describe the semantics for $\vDash$, we will need the set s, as was defined before. In accordance with the argument pattern, we abuse notation also for s and write that $\mu(s) = \{\mu(p_1), \ldots, \mu(p_n)\}$. The following list sets the conditions for each filter to be satisfied:

- for $f = f_{incl}$, $\mu(pr) \vDash f$ *iff* $\mu(s) \subseteq \mu(pr)$
- for $f = f_{eq}$, $\mu(pr) \vDash f$ *iff* $\mu(s) = \mu(pr)$
- for $f = f_{ineq}$, $\mu(pr) \vDash f$ *iff* $\mu(s) \neq \mu(pr)$
- for $f = f_{join}$, $\mu(pr) \vDash f$ *iff* $\mu(s) \cap \mu(pr) \neq \emptyset$
- for $f = f_{disj}$, $\mu(pr) \vDash f$ *iff* $\mu(s) \cap \mu(pr) = \emptyset$

Each path pattern p determines a set of all possible paths that could match with it. That set is denoted as $\odot p$. Essentially, if p has no length indicators, $\odot p$ has a single element, otherwise its cardinality is defined by the indicator. The set $\odot p$ is formally defined as:

Definition 6. *Let p, p', p'' path expressions. $\odot p$ is defined inductively as follows:*

- *If $p \in \{attack, support, undercut, rebut, endorse, back'\}$ then: $\odot p = \{p\}$*
- *If $p = p'/p''$, then: $\odot p = \{p_1/p_2 \mid p_1 \in \odot p', p_2 \in \odot p''\}$*
- *If $p = p' * 1$, then: $\odot p = \{p_1 \mid p_1 \in \odot p'\}$*
- *If $p = p' * n$ and $n > 1$, then $\odot p = \{p_1/p_2 \mid p_1 \in \odot(p' * 1), p_2 \in \odot(p' * (n-1))\}$*
- *If $p = p'+$, then: $\odot p = \{p_1 \mid p_1 \in \odot p' * k$, for some $k \geq 1\}$*

A pattern expression e may have one of the forms: $e = ap$ or $e = ap\ pe\ e'$, with e' another pattern expression. We denote as as $init(e) = ap$ the initial argument pattern for both cases. Next, we define the evaluation of a complete pattern expression e against a debate graph D.

Definition 7 (Evaluation). *Let a pattern expression e and a debate graph $D = \langle A', R'\rangle$. The evaluation of e against D, written as $\mathcal{E}val_D(e)$ is defined:*

- *If $e = ap$, with $ap =< pr[f_{ap}], c >$, then:*
  $\mathcal{E}val_D(ap) = \{\mu \mid \mu(pr) \vDash f_{ap}$ *and* $\mu(ap) \in A'\}$
- *If $e = ap\ pe\ e'$, then: $\mathcal{E}val_D(e) = \{\mu \mid \mu \in \mathcal{E}val_D(ap) \cap \mathcal{E}val_D(e')$ and $\exists p' = P^D_{\mu(ap) \to \mu(init(e'))}$ s.t. $p' \in \odot pe\}$*

4.3 Query execution

To implement our language, we chose to leverage existing and well-optimized storage schemes and languages, in particular RDF and its associated query language, SPARQL 1.1. The reasons for our choice is that, first of all, our target scenario concerns the web, where RDF/SPARQL are the standard languages. Furthermore, RDF represents a graph data model and SPARQL 1.1 provides useful property paths. Finally, SPARQL is accompanied with well-defined semantics [12], which makes the transition from ArgQL to be straightforward.

Our implementation is ongoing work, so we just present the general idea here. First of all, concepts of the data model must be translated into an RDF scheme. In order to be compatible with the state of the art and the recent tendency of the argumentation community, we turn to the RDF representation of the AIF ontology [6]. AIF defines classes equivalent with the basic concepts of the argumentation model presented in section 3. Each element in the tuple $\mathcal{L}$, is mapped to a particular set of triples in AIF. Next, ArgQL queries are translated into SPARQL and are executed against the translated data, which are now instances of AIF scheme. Each expression in the "match" clause in figure 4.1, corresponds to a different graph pattern in SPARQL. After the execution of SPARQL, RDF results are translated back into the expected form for the answers of ArgQL. The equivalence between the expected results and the results received by SPARQL has to be theoretically proved. To test the whole process of the execution in practice, we aim to perform it on the AIFdb corpora[11], an RDF knowledge base in AIF scheme, consisting of over 2,000 argument maps with over 30,000 individual nodes.

5 Research plans and Conclusions

In this work we presented ArgQL, a query language for argumentation. The first version supports a core subset of queries and it refers to a minimal data model which consists only of arguments and their interrelations. We have shown the syntax, its semantics and we briefly described the idea of its implementation.

Our future plans include the completion of both theoretic and technical issues of translation, as well as the experimentation on real datasets. Next, we intent to enrich our data model with more complicated concepts, e.g. topics of discussion and adjust the language specification to the new features. In subsequent time, ArgQL will be integrated with reasoning mechanisms, to allow for dynamic queries, which e.g. will capture well-known argumentation semantics[7], compute acceptability etc. with simpler (in terms of syntax) queries.

In order to tackle more practical, from the web perspective issues, we are going to incorporate facilities that allow "smart" searching within the textual content of argument, such as advanced keyword-searching and content-based searching, imprecise textual mappings (e.g., taking into account synonyms, or typos in the text), exploratory/navigational capabilities etc. Finally, we plan to determine formally the expressive power and complexity of ArgQL.

Aknowledgements. This work is funded by the Institute of Computer Science (ICS) of the Foundation for Research and Technology - Hellas (FO.R.T.H.). We also thank Dr. Theodore Patkos for the insightful discussions about the topic.

References

1. Trevor J. M. Bench-capon, Sylvie Doutre, and Paul E. Dunne. Value-based argumentation frameworks. In *Artificial Intelligence*, pages 444–453, 2002.
2. Tudor Berariu. *An Argumentation Framework for BDI Agents*, pages 343–354. Springer International Publishing, Cham, 2014.
3. Floris Bex, John Lawrence, Mark Snaith, and Chris Reed. Implementing the argument web. *Commun. ACM*, 56(10):66–73, October 2013.
4. Floris Bex, Henry Prakken, and Chris Reed. A formal analysis of the aif in terms of the aspic framework. In *Proceedings of the 2010 Conference on Computational Models of Argument: Proceedings of COMMA 2010*, pages 99–110, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.
5. Dan Cartwright, Katie Atkinson, and Trevor Bench-capon. Supporting argument in edemocracy. In *Proceedings of the Third conference on Electronic Democracy (EDEM 2009*, pages 15–160, 2009.
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. *Knowl. Eng. Rev.*, 21(4):293–316, December 2006.
7. 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):321–357, September 1995.
8. Giorgos Flouris, Antonis Bikakis, Patkos Theodore, and Dimitris Plexousakis. Globally interconnecting persuasive arguments: The vision of the persuasive web. Technical report, FORTH-ICS/TR-438, 2013.
9. Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory Pract. Log. Program.*, 4(2):95–138, January 2004.
10. Steve H. Garlik, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query/.
11. John Lawrence and Chris Reed. *AIFdb Corpora*, pages 465–466. Frontiers in artificial intelligence and applications. IOS Press, 2014.
12. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, September 2009.
13. Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2009.
14. Iyad Rahwan, Fouad Zablith, and Chris Reed. Laying the foundations for a world wide argument web. *Artif. Intell.*, 171(10-15):897–921, July 2007.
15. Jodi Schneider, Tudor Groza, and Alexandre Passant. A review of argumentation for the social semantic web. *Semant. web*, 4(2):159–218, April 2013.
16. Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, July 2003.
17. Douglas N. Walton. *Argumentation Schemes for Presumptive Reasoning*. L. Erlbaum Associates, 1996.