Carneades WebApp 0.7 User Manual

Tom Gordon

Contents

1 Introduction **4**

2 Getting Started **4**

2.1 Downloading Carneades 4

2.2 License 5

2.3 Binary Installation 5

2.3.1 Using the Web Application Locally 6

3 The Projects Page **6**

4 Argument Graphs **8**

4.1 Data Model 9

4.2 Statement Properties 11

4.3 Argument Properties 11

4.4 Premise Properties 12

4.5 Metadata 12

5 Browsing, Visualizing and Evaluating Arguments **13**

5.1 The Outline Page 13

5.2 Using Hypertext to Browse an Argument Graph 15

5.2.1 Statement Pages 17

5.2.2 Argument Pages 18

5.3 Visualizing Argument Graphs in Argument Maps 20

5.4 Evaluating Arguments 21

6 Editing Argument Graphs **24**

6.1 Reconstructing Arguments 26

6.2 Entering New Statements 27

6.2.1 Main Issue 28

6.2.2 Statement Text 28

6.2.3 Atom: Formalizing Statements using Predicate Logic . . . 29

6.2.4 Proof Standard 29

6.2.5 Statement Weight 30

6.2.6 Statement Metadata 30

6.3 Editing Statements 30

6.4 Deleting Statements 31

6.5 Entering New Arguments 31

6.5.1 Using Argumentation Schemes 31

6.5.2 Entering and Deleting Premises 33

6.5.3 Strict or Defeasible Arguments 33

6.5.4 Argument Direction 34

6.5.5 Argument Weight 34

6.5.6 Argument Metadata 34

6.6 Editing Arguments 35

6.7 Deleting Arguments 35

6.8 Evaluating Argument Graphs 35

7 Argumentation Schemes **36**

7.1 Modeling Argumentation Schemes 37

8 System Administration **42**

8.1 Building and Installing from Source Code 42

8.2 System Configuration 42

8.3 Project Structure and Configuration 43

9 Builtin Argumentation Schemes 44

9.1 Argument from Position to Know . . . 44
9.2 Argument from Credible Source . . . 45
9.3 Argument from Witness Testimony . . . 45
9.4 Argument from Expert Opinion . . . 46
9.5 Argument from Analogy . . . 46
9.6 Argument from Precedent . . . 46
9.7 Argument from Verbal Classification . . . 47
9.8 Argument from Definition to Verbal Classification . . . 47
9.9 Defeasible Modus Ponens . . . 47
9.10 Argument from an Established Rule . . . 48
9.11 Argument from Positive Consequences . . . 48
9.12 Argument from Negative Consequences . . . 48
9.13 Argument from Practical Reasoning . . . 48
9.14 Argument from Cause to Effect. . . . 49
9.15 Argument from Correlation to Cause . . . 49
9.16 Argument from Sunk Costs . . . 50
9.17 Argument from Appearance . . . 50
9.18 Argument from Ignorance . . . 50
9.19 Argument from Abduction . . . 50
9.20 Ethotic Argument . . . 51
9.21 Slippery Slope Argument . . . 51
9.21.1 Base Case . . . 51
9.21.2 Inductive Step . . . 51

10 Credits 52

11 User Manual License 52

References 53

1 Introduction

The Carneades argumentation system is series of open source, research prototypes freely available for downloading at http://carneades.github.com.

The version of Carneades described in this user manual, Carneades WebApp 0.7, provides web-based, collaborative software tools for:

- (re)constructing arguments using a rulebase of argumentation schemes
- visualizing arguments in diagrams (argument maps)
- critically evaluating arguments, using a formal model of structured argument based on argumentation schemes and with support for variable proof standards (Gordon, Prakken, and Walton 2007).

The next chapter, Getting Started, explains how to download, install and run the Carneades system locally on your personal computer. The projects chapter explains how to create and access projects, where each project can contain any number of argument graphs. The argument graph chapter provides an overview of the underlying data model used by all the tools. The browsing, visualization and evaluation chapter explains how to read and view argument graphs. The chapters on editing argument graphs and modeling argumentation schemes are somewhat more demanding and some basic prior knowledge of argumentation theory (Walton 2006) is recommended. The system administration chapter describes how to build and install the system from its source files, as well as how to configure the system. The final chapter shows how some well-known argumentation schemes (Walton, Reed, and Macagno 2008) can been represented using the system.

2 Getting Started

This chapter explains how to download, install and run the Carneades system locally on your personal computer. For information about how to build the system from source code and advanced configuration options see the "System Administration" chapter.

2.1 Downloading Carneades

The Carneades Argumentation System is open source software available at https://github.com/carneades.

You can download binaries and source code of Carneades releases at from the releases page. To use the version of Carneades described in this manual, download the release named "Carneades WebApp 0.7".

2.2 License

The source code of the Carneades system is licensed using the Mozilla Public License (MPL) version 2.0. An English version of the license is distributed with the software, in the `/licenses` directory. The MPL license is certified by the Open Source Initiative (OSI).

> The MPL is a simple copyleft license. The MPL's "file-level" copyleft is designed to encourage contributors to share modifications they make to your code, while still allowing them to combine your code with code under other licenses (open or proprietary) with minimal restrictions.

In particular, if you write an application which links to Carneades as a library, you are free to use any license you wish for your own code.

See http://www.mozilla.org/MPL/2.0/FAQ.html for more information.

2.3 Binary Installation

Prerequisites:

- Version 7 or better of a Java Runtime Environment.

Installation Procedure:

1. Download the `carneades-webapp.0.7.0.zip` file from Carneades WebApp 0.7 release on the releases page.
2. Unzip the `carneades-webapp.0.7.0.zip` Zip archive file using some Zip tool. This will create a directory (folder) with the following hierarchical structure
    - carneades-webapp
        - carneades-webapp-0.7.0.jar
        - config ** carneades.clj
        - doc ** manual.pdf ** timestamp.txt
        - projects
        - README.txt

You can move this directory to some other location on your file system, at any time.

This creates a standard installation, with the default configuration.

2.3.1 Using the Web Application Locally

To start the Carneades web application server double click on the `carneades-webapp-0.7.0.jar` file in your file system browser, for example the “Finder” on Mac OS X or the “Windows Explorer” on Windows PCs.

To start the server to from a command line, for local use, type

```
$ java -jar carneades-webapp.0.7.0.jar
```

Either way, after the server starts it will open up the projects page of the Carneades web application in your default web browser.

Depending on your operating system and how you started the server, the Carneades web application can be shut down by either quitting the Carneades application or, if you started the server from a command line, using a terminal application, by ending this process, typically by typing `control-c` in the terminal.

3 The Projects Page

The projects page is the first page shown in your Web browser when you start Carneades. It lists example projects and the projects you have created, and provides a way to create new projects.

To access a project, click on its name in the list of projects.

To create a new project, click on the “New Project” button, complete the form and then click on the “Save” button.

The “theory” field allows you to specify which argumentation schemes to use to construct and reconstruct arguments. It is initialized to use model of Walton’s argumentation schemes distributed with the system, from the default project: “default/walton_schemes”.

Projects are store in files on your file system the directory specified in the “carneades.clj” file in your home directory. See the System Administration chapter for further configuration details.

You can change the theory to use other argumentation schemes, including ones you have written yourself. See the “Argumentation Schemes” chapter for information on how to write theories with your own argumentation schemes. Save or move the theory to the “theories” directory of your project, in the projects directory specified in your “carneades.clj” configuration file. Suppose you have named this theory “my_schemes.clj” and stored it in the theories directory of your project. To have this theory used by your project, instead of

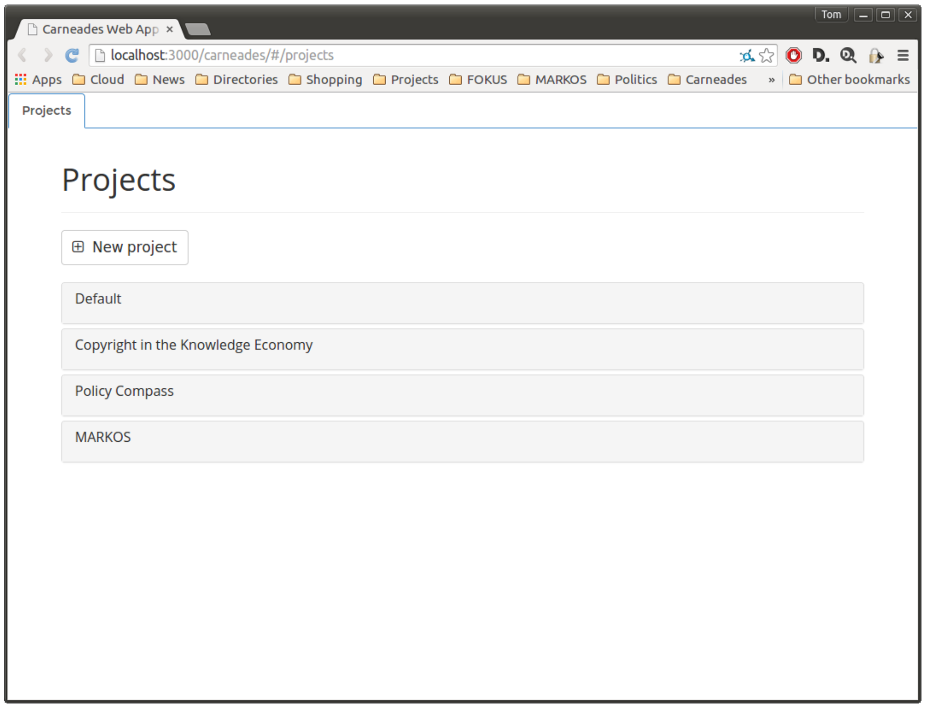


Figure 1: Example Projects Page

"default/walton_schemes", edit the "properties.clj" file in the project directory and then change the theory to "my_schemes". Notice that you need not name the project, since the theory is in this project, and you also should not add the ".clj" (Clojure) file extension. (Theories are implemented using a rule language embedded in the Clojure programming language.) This must be done locally on the computer running the Carneades server. There is currently no Web user interface for editing the properties of projects or managing the theories of projects.

4 Argument Graphs

Argument graphs model relationships among arguments and statements (claims, propositions). The arguments represented in an argument graph can be new (invented) arguments *constructed* from knowledge, evidence and facts, or *re-constructed* from arguments in source documents, such as court opinions or newspaper commentaries.

Argument reconstruction is a challenging task requiring the interpretation of natural language texts. Consider the following simple legal argument: Johnny violated the law by riding his skateboard in the park.

This same argument can be expressed in many different ways in natural langauge. Here are some examples:

1. Because Johnny rode his skateboard in the park he violated the law.
2. Vehicles are prohibited in the park. Someone who does something which is prohibited violates the law. Johnny rode his skateboard in the park. A skateboard is a vehicle. Therefore Johnny violated the law.
3. Johnny hat gegen das Gesetz verstoßen, weil er sein Skateboard im Park benutzt hat.

The first example just reordered the premise and the conclusion of the argument, putting the premise first. The second example reveals some implicit premises of the original formulation of the argument. The third example is a German translation of the original formulation of the argument.

All four of these texts, including the original formulation, express the *same* argument, but in different ways. In a large-scale debate, for example about European policy issues, the same argument might be expressed in *thousands* of different ways in many different languages. (The European Union has 23 official languages.)

One of the purposes of argument graphs is to provide a way to summarize the arguments put forward in complex debates with exactly one node in the graph for each argument, no matter how often or in how many ways it has been expressed. A single argument graph is used to represent all the arguments put forward in a debate, from all participants. The nodes can quote one or more source documents, and include links to these source documents, so no information need be lost and all voices can still be heard, inclusively. Grouping the different formulations of an argument together into a single node in the graph, abstracting away details, makes it possible to quickly obtain an overview of the arguments and to obtain a clearer picture of relationships among arguments. A hypertext or map of the source documents directly, without an argument graph, would make it difficult to “see the forest for the trees”.

Argument graphs are also useful as a decision-support tool when analysing problems and issues, such as legal issues, whether by an invidual or in a group. When used this way, the arguments in the graph represent inference steps and dependencies between propositions. Original arguments can be entered or edited directly by users, whether or not the arguments have first been formulated in some natural language text. If the relevant knowledge of the application domain has been formally represented, using a knowledge-based system, some arguments can be automatically found and entered into the graph. Carneades provides a knowledge representation language for argumentation schemes and an inference engine which can be used to find arguments in this way. Both methods can be used together. Carneades is an interactive argumentation assistent, not a fully automatic problem-solver. Arguments which have been found automatically can be manually edited by users.

4.1 Data Model

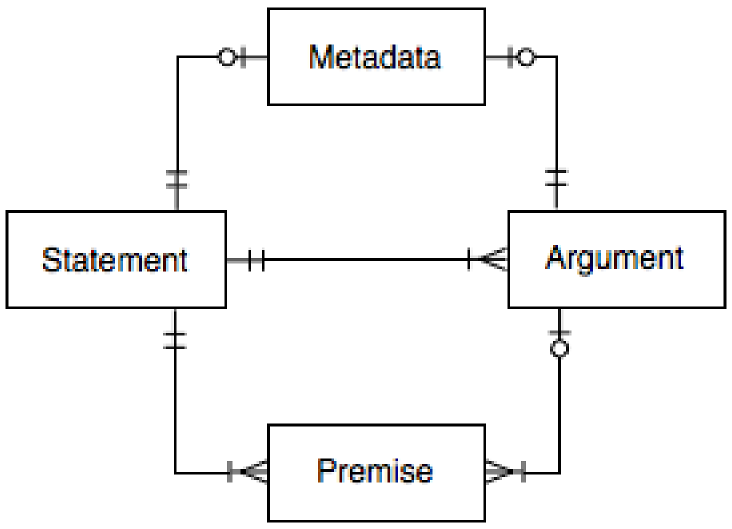


Figure 2: Entity-Relationship Diagram

The entity-relationship diagram above shows the elements of argument graphs and their connections. (The figure does not visualize a particular argument graph, but rather relationships between the elements of argument graphs in general.)

The two main elements of argument graphs are statements and arguments. Statements represent propositions, claims and assertions. Arguments represent simple inferences from one or more premises to a single conclusion. Again, there should be only one statement or argument in the graph for each statement and argument in the source documents, no matter how many different ways the statement or argument has been expressed in source documents. Some or all formulations of the statement or argument can be quoted or referenced in the metadata of the statement or argument node. See the discussion of metadata below for further information.

As can be seen in the entity-relationship diagram, arguments are linked to statements in two ways in argument graphs. Each argument has exactly one conclusion, which is a statement, and zero or more premises, where each premise has exactly one statement node. A statement may the conclusion or premise of more than one argument.

A statement may be both a conclusion and a premise, resulting in complex argument graphs, representing chains or trees of reasoning. Argument graphs may contain cycles. A simple cycle would result if a statement and a premise of the same argument. There are methods for resolving these cycles when evaluating argument graphs.

A statement in an argument graph represents a propositional variable, whose value may be true (in), false (out) or unknown (undecided). To continue with our example, the sentences "Johnny rode his skateboard in the park" and "Johnny did not ride his skateboard in the park" would be represented by a single statement in an argument graph. Conclusions and premises of argument can negated using con arguments and negative premises, respectively. There are two kinds of arguments, pro and con. An argument is pro if its conclusion claims the statement is true and con if it claims the statement is false. Similarly, there are two kinds of premises, positive and negative. A positive premise holds if its statement is "in". Conversely, a negative premise holds only if its statement is "out".

Prior models of argument graphs do not distinguish pro and con arguments or positive and negative premises. Rather, in these prior approaches all argument nodes are pro and all premises are positive. Our approach has the advantage of reducing the number of statements up to 50%, resulting in more compact summaries of the arguments.

4.2 Statement Properties

id : A Uniform Resource Name (URN) serving as a unique identifier for the statement, world-wide.

text : A concise formulation of the statement, written by the analyst who reconstructed the arguments from the source documents. Paraphrases the various formulations of the statement in the sources. Translations of the text in several languages may be included in the model. Compare with the "description" property of the metadata of the statement, which can be used to quote some or all of the formulations of the statement in the sources and provide translations in several languages.

weight : A real number in range of 0.0-1.0 representing the degree to which the statement is accepted as true by users ("audience"). If the weight is 0.0, the statement is false (rejected by the users). If the weight is greater than 0.0 but less or equal to 0.25, the statement is *assumed* to be false. If the weight is greater than or equal to 0.75 and less than 1.0, the statement is assumed to be true. If the weight is 1.0, the statement is true (accepted by the users). Otherwise the truth or falsity of the statement is at issue. In application scenarios with many users, the weights can be collected via polls.

proof standard : The method used to combine pro and con arguments. Several proof standards are supported by the system. For most purposes, the "preponderance of the evidence" standard should suffice. See the Editing Argument Graphs Chapter for details.

value : A real number in the range 0.0-1.0, storing the output of the argument evaluation process, where 0.0 means the statement is *out* (false or presumably false), 1.0 means the statement is *in* (true or presumably true) and all other values mean the arguments, facts and assumptions are insufficient for drawing any conclusions about the truth or falsity of the statement (undecided, unknown). (Real numbers are used to allow experimentation with other models of argument evaluation.)

atom : An optional formal representation of the statement in predicate logic. (This feature is for analysts and need not interest public users.)

main : A Boolean value (true or false) used to indicate whether the statement is one of the main issues modeled by the argument graph. (Since argument graphs can contain cycles and not limited to trees, we cannot use the roots of trees for this purpose.)

4.3 Argument Properties

id : A Uniform Resource Name (URN) serving as a unique identifier for the argument, world-wide.

direction : Pro or con.

strict : A Boolean value (true or false) expressing whether the conclusion of the argument is necessarily true when its premises are true (strict arguments) or only presumably true. Nonstrict arguments are called “defeasible” arguments.

scheme : The name of the argumentation scheme applied, if any. Optional. Examples: “argument from credible source”, “argument from practical reasoning”.

weight : A real number in range of 0.0-1.0, representing the relative weight of the argument, compared to other arguments pro and con the conclusion of the argument. The weight is assessed by users. If there are multiple users, the weights can be collected using polls.

value : A real number in in the range 0.0-1.0, used to record the output of the argument evaluation process, where 0.0 means the argument is *out* (not acceptable), 1.0 means the argument is *in* (acceptable) and all other values mean the arguments in the graph, taken together, are insufficient for determining the acceptability of this argument (undecided/unknown). (Real numbers are used to allow experimentation with other models of argument evaluation.)

4.4 Premise Properties

polarity : Positive or negative.

role : The role of the premise in the argumentation scheme applied. Examples: “minor”, “major”.

implicit : A Boolean value (true or false). Can be used to note that the premise was not explicit in the source documents from which the argument node was reconstructed.

4.5 Metadata

The argument graph as as whole, as well as each of its statements and arguments, can be annotated with metadata, using the Dublin Core. There are 15 elements in the Dublin Core. Each element may contain zero or more values. Here is a list of the Dublin Core elements:

1. Title
2. Creator
3. Subject
4. Description
5. Publisher
6. Contributor
7. Date
8. Type
9. Format

10. Identifier
11. Source
12. Language
13. Relation
14. Coverage
15. Rights

See Dublin Core for a detailed description and usage guidelines for each element. The Dublin Core is intended to be useful for describing a wide range of "resources" on the World-Wide Web.

In addition, Carneades allows each metadata record to be assigned an optional "key", a string which can be used as a label to refer to the metadata record, such as "BenchCapon:2008", similar to the way citation keys are used in bibliographic databases such as BibTeX.[1] At most one key should be provided.

Carneades provides special support for providing description elements of the Dublin Core in multiple languages (English, German, French, ...) and for formatting these descriptions using the Markdown language. This feature can be used to include quotations from and links to source documents in the descriptions of both statements and arguments.

5 Browsing, Visualizing and Evaluating Arguments

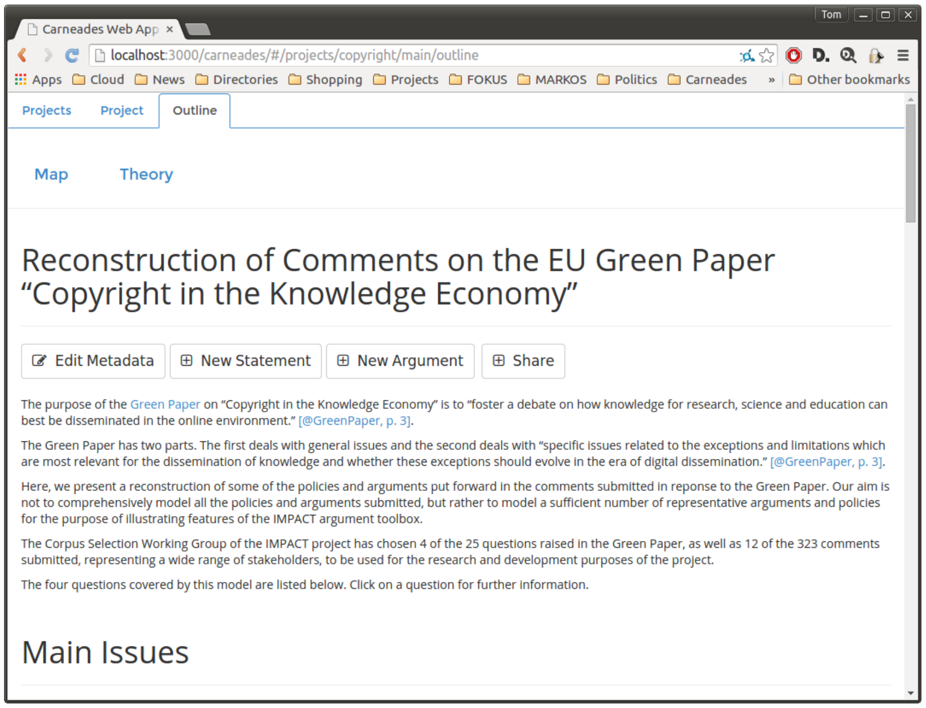
This chapter of the Carneades user manual explains how to:

- Access the outline page to view the title, description, issues and outline of the top-level arguments of the argument graph.
- Use hypertext in web pages to browse an argument graph.
- Visualizing argument graphs in diagrams, called "argument maps", and using these maps to navigate to more detailed views of statements and arguments.
- Evaluate arguments to reveal missing premises, check the form of arguments, ask critical questions and assess the acceptability of statements.

5.1 The Outline Page

An outline page consists of the following parts:

[1] http://en.wikipedia.org/wiki/BibTeX



Reconstruction of Comments on the EU Green Paper "Copyright in the Knowledge Economy"

The purpose of the Green Paper on "Copyright in the Knowledge Economy" is to "foster a debate on how knowledge for research, science and education can best be disseminated in the online environment." [@GreenPaper, p. 3].

The Green Paper has two parts. The first deals with general issues and the second deals with "specific issues related to the exceptions and limitations which are most relevant for the dissemination of knowledge and whether these exceptions should evolve in the era of digital dissemination." [@GreenPaper, p. 3].

Here, we present a reconstruction of some of the policies and arguments put forward in the comments submitted in reponse to the Green Paper. Our aim is not to comprehensively model all the policies and arguments submitted, but rather to model a sufficient number of representative arguments and policies for the purpose of illustrating features of the IMPACT argument toolbox.

The Corpus Selection Working Group of the IMPACT project has chosen 4 of the 25 questions raised in the Green Paper, as well as 12 of the 323 comments submitted, representing a wide range of stakeholders, to be used for the research and development purposes of the project.

The four questions covered by this model are listed below. Click on a question for further information.

Main Issues

Figure 3: An Outline Page

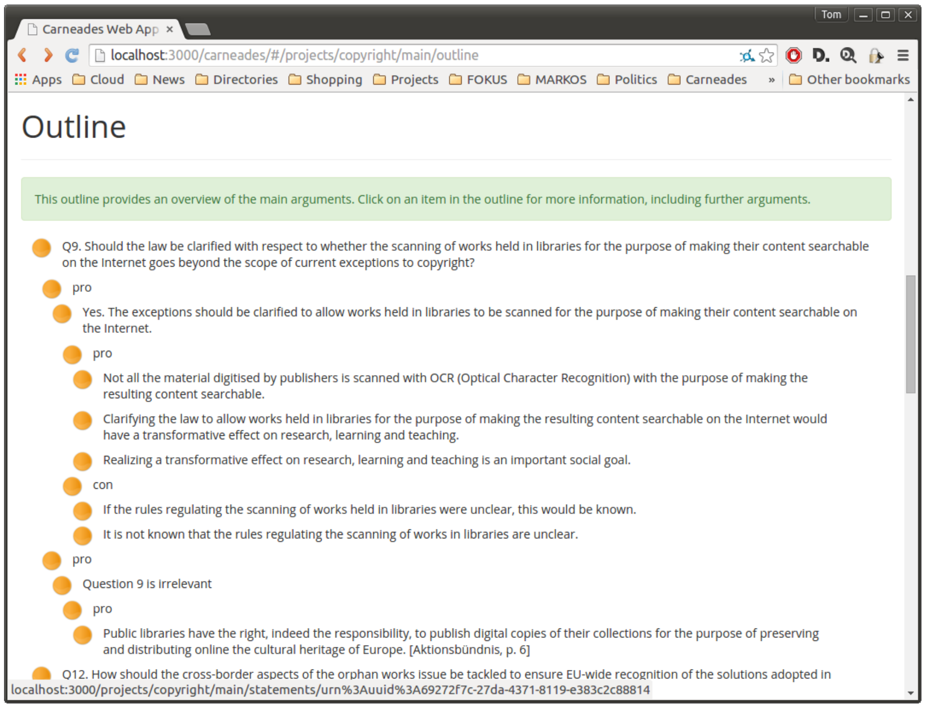
- The *title* of the argument graph. This title usually includes the topic of the discussion or debate.
- A *menu bar* of commands.
- A *description* of the topic of the discussion modeled in the argument graph. The description can be arbitrarily long and include multiple sections, paragraphs, images, hyperlinks, lists and other content.
- A list of the *main issues* of the discussion. Each item in the list is linked to a page providing detailed information about the statement in the argument graph at issue.
- The outline, which presents the top five levels of the arguments in the argument graph. The first level of the outline lists the main issues (again). The second level lists the arguments pro and con each issue. The third level lists the premises of each of these arguments. The fourth level lists the argument pro and con each premise. Finally, the fifth level lists the premises of these arguments. Deeper levels of the argument graph can be navigated to by first clicking on a statement or argument in the outline and then following the links on the next page. Since argument graphs may contain cycles and are not restricted to trees, some items may appear multiple times in the outline.
- A list of *references* to the source documents used to construct the argument graph. For documents available on the Web, the reference will include a hyperlink to the source document.

Notice that the issues and items in the outline are prefixed with colored dots. These indicate the current status (label) of the statements and arguments in the graph, where:

Yellow means the statement or argument is undecided or unknown. Green means the statement or argument is acceptable (in, true or presumably true). Red
means the statement or argument is not acceptable (out, false or presumably false).

5.2 Using Hypertext to Browse an Argument Graph

There is a web page for each statement and argument in the argument graph providing detailed information about the element along with links to related statements and arguments in the graph. You can use these pages to navigate from node to node in the argument graph, by simply clicking on the links in the usual way. To go back to previous pages, use the back button of your web browser.


Carneades Web App
localhost:3000/carneades/#/projects/copyright/main/outline
Apps Cloud News Directories Shopping Projects FOKUS MARKOS Politics Carneades » Other bookmarks
Tom

Outline

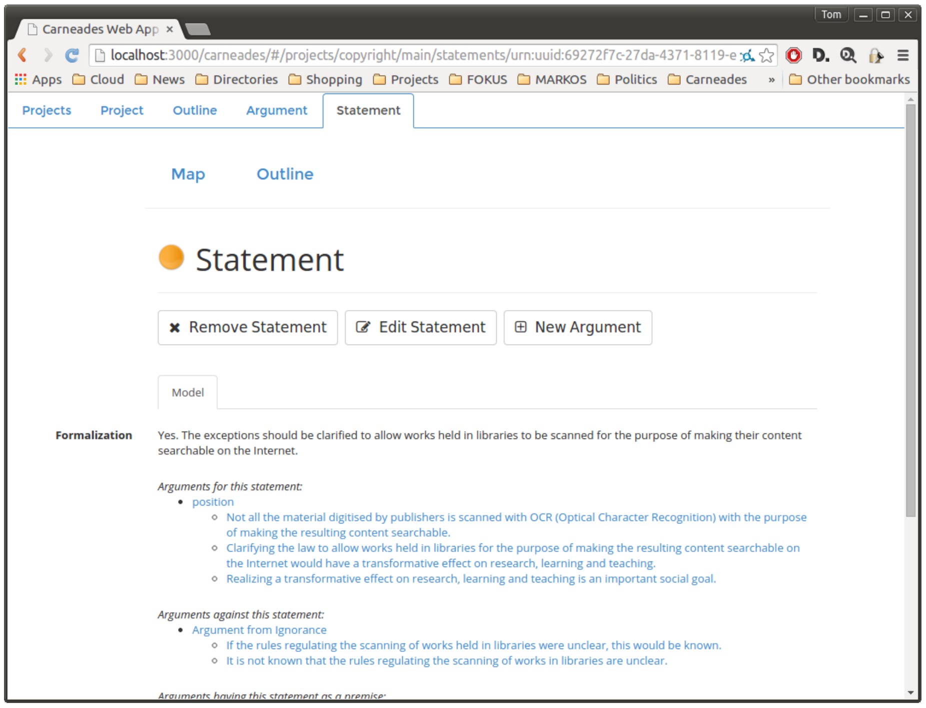
This outline provides an overview of the main arguments. Click on an item in the outline for more information, including further arguments.

- Q9. Should the law be clarified with respect to whether the scanning of works held in libraries for the purpose of making their content searchable on the Internet goes beyond the scope of current exceptions to copyright?
  - pro
    - Yes. The exceptions should be clarified to allow works held in libraries to be scanned for the purpose of making their content searchable on the Internet.
      - pro
        - Not all the material digitised by publishers is scanned with OCR (Optical Character Recognition) with the purpose of making the resulting content searchable.
        - Clarifying the law to allow works held in libraries for the purpose of making the resulting content searchable on the Internet would have a transformative effect on research, learning and teaching.
        - Realizing a transformative effect on research, learning and teaching is an important social goal.
      - con
        - If the rules regulating the scanning of works held in libraries were unclear, this would be known.
        - It is not known that the rules regulating the scanning of works in libraries are unclear.
  - pro
    - Question 9 is irrelevant
      - pro
        - Public libraries have the right, indeed the responsibility, to publish digital copies of their collections for the purpose of preserving and distributing online the cultural heritage of Europe. [Aktionsbündnis, p. 6]
- Q12. How should the cross-border aspects of the orphan works issue be tackled to ensure EU-wide recognition of the solutions adopted in

localhost:3000/projects/copyright/main/statements/urn%3Auuid%3A69272f7c-27da-4371-8119-e383c2c88814

Figure 4: The Outline of an Argument Graph Page

5.2.1 Statement Pages



Statement

Formalization Yes. The exceptions should be clarified to allow works held in libraries to be scanned for the purpose of making their content searchable on the Internet.

Arguments for this statement:

- position
  - Not all the material digitised by publishers is scanned with OCR (Optical Character Recognition) with the purpose of making the resulting content searchable.
  - Clarifying the law to allow works held in libraries for the purpose of making the resulting content searchable on the Internet would have a transformative effect on research, learning and teaching.
  - Realizing a transformative effect on research, learning and teaching is an important social goal.

Arguments against this statement:

- Argument from Ignorance
  - If the rules regulating the scanning of works held in libraries were unclear, this would be known.
  - It is not known that the rules regulating the scanning of works in libraries are unclear.

Arguments having this statement as a premise:

Figure 5: A Statement Page

The proof standards available are dialectical validity, preponderance of the evidence, clear and convincing evidence, and beyond reasonable doubt. See the section on Evaluating Arguments for further details about proof standards.

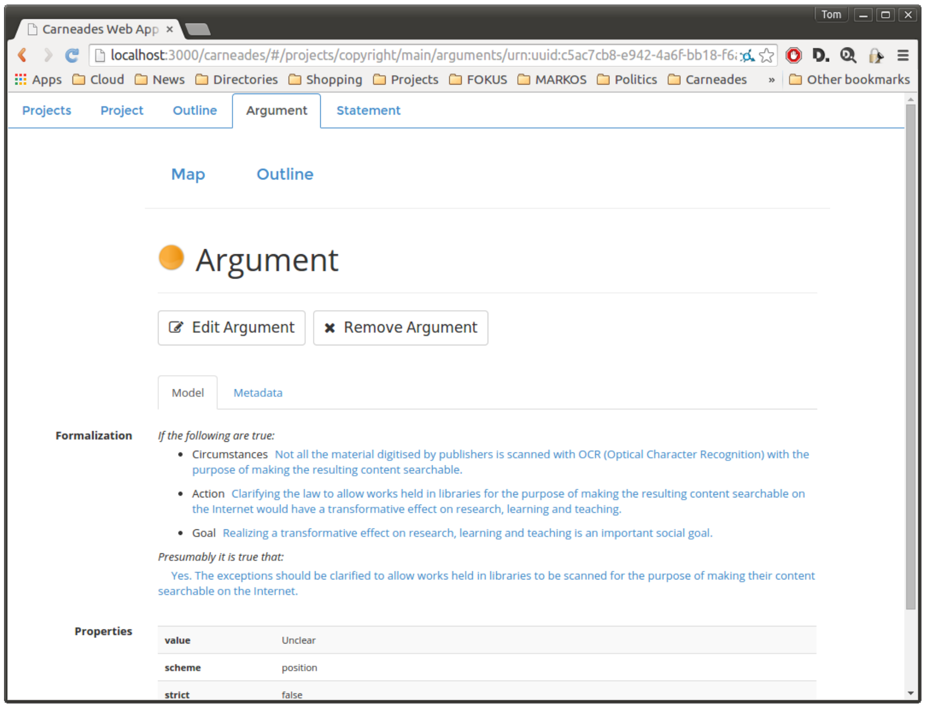
The next section displays the content of the statement in natural langauge. This formulation of the statement is written by the analyst or analysts who reconstructed the arguments to build the argument graph, or generated from a template in a theory, when arguments are constructed automatically using the Carneades inference engine.

Next, arguments pro and con the statement are displayed, as well as arguments which use this statement (or its negation) as a premise. The premises of these arguments are also displayed. This makes it possible to navigate to nearby arguments and statements in the argument graph, by simply clicking on the links in these lists.

The bottom of the statement page displays the properties of the statement: its value, proof standard (default value: preponderance of the evidence), weight, whether or not it is a main issue, its formal representation as a logical atom (optional) and its id.

If metadata had been provided for the statement, it would be displayed in a separate tab on the page. Descriptions may be provided in multiple natural langauges, with a tab for selecting the description in each language.

5.2.2 Argument Pages



Formalization

If the following are true:

- Circumstances Not all the material digitised by publishers is scanned with OCR (Optical Character Recognition) with the purpose of making the resulting content searchable.
- Action Clarifying the law to allow works held in libraries for the purpose of making the resulting content searchable on the Internet would have a transformative effect on research, learning and teaching.
- Goal Realizing a transformative effect on research, learning and teaching is an important social goal.

Presumably it is true that:

Yes. The exceptions should be clarified to allow works held in libraries to be scanned for the purpose of making their content searchable on the Internet.

Properties

value	Unclear
scheme	position
strict	false

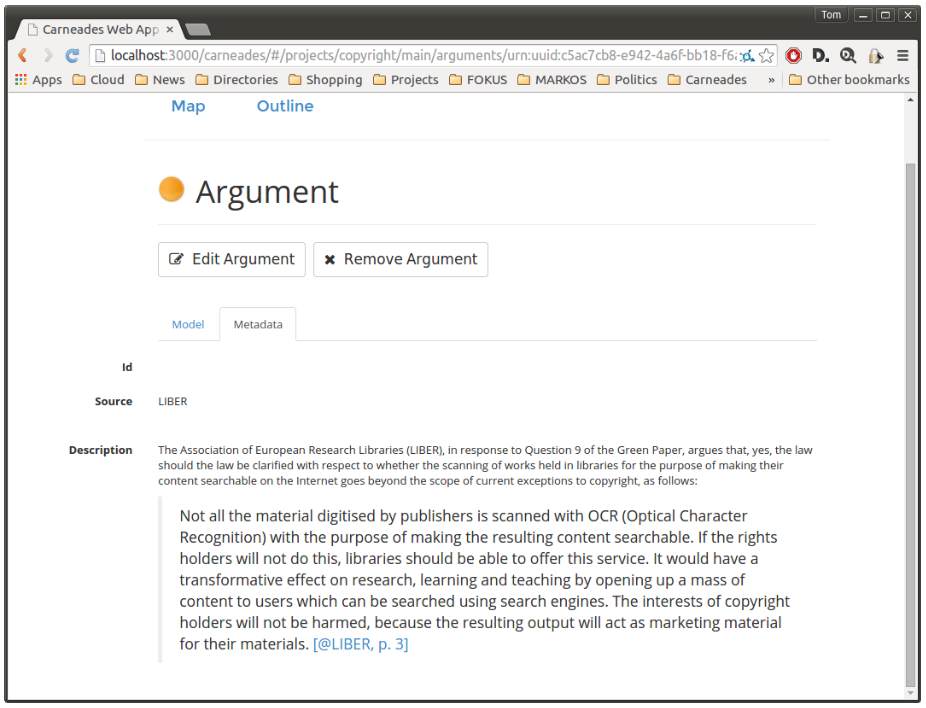
Figure 6: An Argument Page

Argument pages are quite similar to statement pages.

At the top the page, the premises and conclusion of the argument are shown. If available, the role of each premise in the argumentation scheme applied is shown (e.g. “major” or “minor”).

The bottom of the page displays the properties of the argument: its id, the argumentation scheme applied (if any), whether it is a strict or defeasible argument, its weight and value. The argumentation scheme contains a hyperlink (*Note: not yet implemented*). Click on the link to view a description of the scheme.

If metadata had been provided for the argument, it would be displayed next. Descriptions can include quotations of one or more source texts expressing the argument, along with hyperlinks to the sources on the Web.


Carneades Web App
localhost:3000/carneades/#/projects/copyright/main/arguments/urn:uuid:c5ac7cb8-e942-4a6f-bb18-f6
Apps Cloud News Directories Shopping Projects FOKUS MARKOS Politics Carneades Other bookmarks
Map
Outline
Edit Argument
Remove Argument
Model
Metadata

Argument

Id

Source LIBER

Description The Association of European Research Libraries (LIBER), in response to Question 9 of the Green Paper, argues that, yes, the law should the law be clarified with respect to whether the scanning of works held in libraries for the purpose of making their content searchable on the Internet goes beyond the scope of current exceptions to copyright, as follows:

> Not all the material digitised by publishers is scanned with OCR (Optical Character Recognition) with the purpose of making the resulting content searchable. If the rights holders will not do this, libraries should be able to offer this service. It would have a transformative effect on research, learning and teaching by opening up a mass of content to users which can be searched using search engines. The interests of copyright holders will not be harmed, because the resulting output will act as marketing material for their materials. [@LIBER, p. 3]

Figure 7: Metadata Tab of an Argument Page

5.3 Visualizing Argument Graphs in Argument Maps

The menus of the argument graph page, statement pages and argument maps include "map" and "outline" buttons. Clicking on the "map" button generates a diagram, called an "argument map", which visualizes the argument graph as a network (directed graph) of statement nodes and argument nodes connected by links. Statement nodes are shown as boxes; argument nodes with circles and boxes with rounded corners.

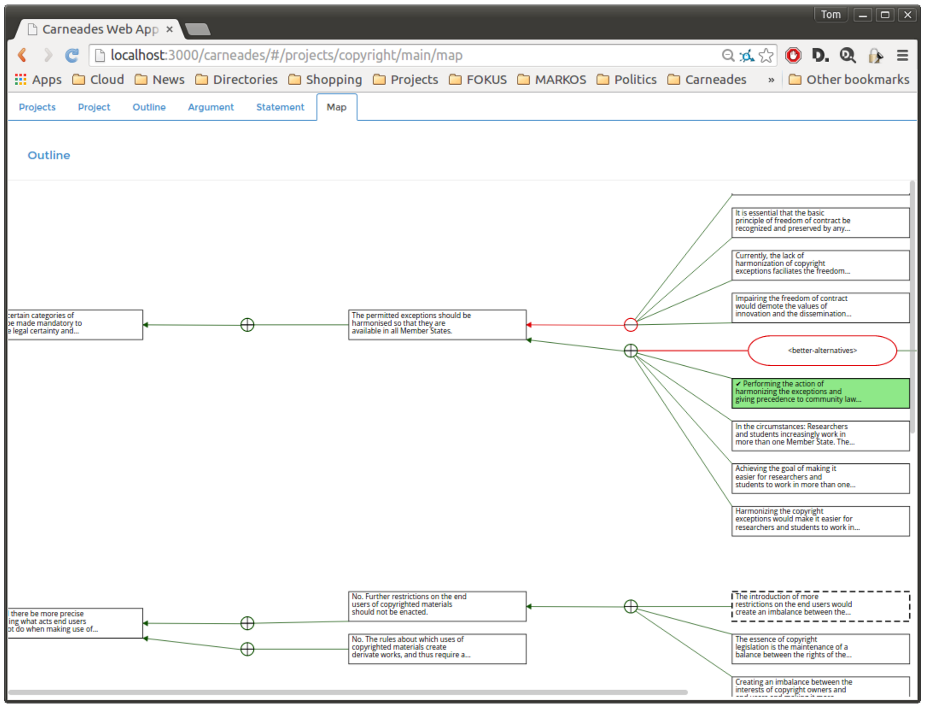

Carneades Web App
localhost:3000/carneades/#/projects/copyright/main/map
Apps
Cloud
News
Directories
Shopping
Projects
FOKUS
MARKOS
Politics
Carneades
Other bookmarks
Projects
Project
Outline
Argument
Statement
Map
Outline
certain categories of be made mandatory to e legal certainty and...
The permitted exceptions should be harmonised so that they are available in all Member States.
It is essential that the basic principle of freedom of contract be recognized and preserved by any...
Currently, the lack of harmonization of copyright exceptions faciliates the freedom...
Impairing the freedom of contract would demote the values of innovation and the dissemination...
<better-alternatives>
✓ Performing the action of harmonizing the exceptions and giving precedence to community law...
In the circumstances: Researchers and students increasingly work in more than one Member State. The...
Achieving the goal of making it easier for researchers and students to work in more than one...
Harmonizing the copyright exceptions would make it easier for researchers and students to work in...
there be more precise ing what acts end users ot do when making use of...
No. Further restrictions on the end users of copyrighted materials should not be enacted.
No. The rules about which uses of copyrighted materials create derivate works, and thus require a...
The introduction of more restrictions on the end users would create an imbalance between the...
The essence of copyright legislation is the maintenance of a balance between the rights of the...
Creating an imbalance between the interests of copyright owners and

Figure 8: An Argument Map

For statement nodes, the text of the statement is shown inside the box, possibly truncated if the text is too long. In argument nodes, the circle is filled with a plus sign, if the argument is a pro argument, or a minus sign, for con arguments. The edges (links) between argument nodes and statement nodes show the premises and conclusion of the argument. The conclusion of the argument is the statement node pointed to by the edge with the normal arrowhead. The other statement nodes linked to the argument, without arrowheads, are its premises. Negative premises are displayed with a circular (dot) arrowhead on the statement side of the edge.

The statement and argument nodes in argument maps contain hyperlinks. Clicking on a statement or argument node displays the details of the node in a statement or argument page, respectively.

In argument maps, argument nodes whose conclusion is another argument node, rather than a statement node, visualize “undercutting” arguments. These are arguments which question the applicability of another argument. This is the only case where two nodes of the same time are directly connected in the map.

Argument maps are represented using structured vector graphics (SVG) not bitmaps. You can zoom the map in or out, to any scale, without loss of resolution. How this zooming is done depends on your device and web browser.

When argument graphs have been evaluated, the status of the argument and statement nodes is visualized in argument maps using both color and icons. Nodes which are “in” are filled with a green background and contain a checked box. Nodes which are “out” are shown with a red background and contain a crossed box (a box filled with an X). Nodes which are neither in nor out are filled with white background color and contain an empty checkbox. The colors are redundant to accommodate black and white printing and color-blind users.

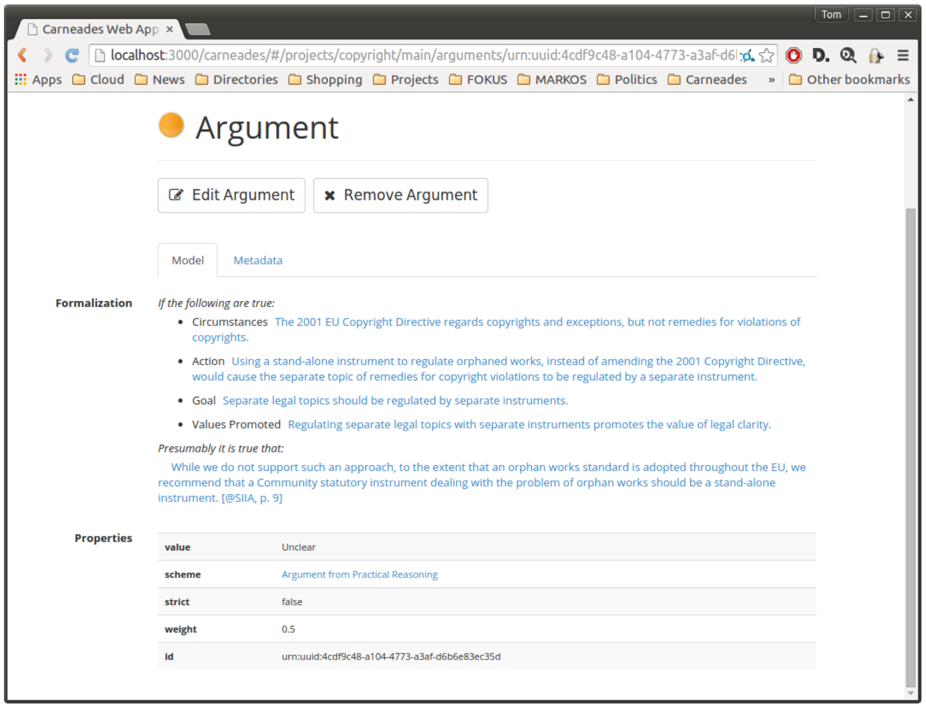
5.4 Evaluating Arguments

By argument “evaluation” we mean the process of critically assessing arguments by:

1. revealing implicit premises
2. validating whether the arguments are formally correct, by instantiating accepted argumentation schemes
3. asking appropriate critical questions, depending on the schemes applied
4. and determining which claims are acceptable, taking into consideration the assumptions of users and their collective assessment of the relative weights of conflicting pro and con arguments.

The first three of these tasks can be accomplished by comparing the argument with its argumentation scheme. On the argument page, click on the argumentation scheme to view a description of the scheme. Now you can check whether any of the premises listed in the scheme are missing from the argument. The argument is formally valid if all the premises of the scheme are explicitly provided by matching premises of the argument and the conclusion of the argument matches the conclusion of the scheme.

Argumentation schemes define exceptions and assumptions which can be used to ask critical questions. The exceptions provide reasons for not applying the argument, undercutting it. If an exception is true, this doesn’t mean that the conclusion of the argument is false, but only that the argument does not provide a good reason to presume the conclusion to be true. The assumptions of the scheme are implicit premises which need to be proven only if they are called into question. So, if you think an assumption does not hold, you should make



Argument

Formalization

If the following are true:

- Circumstances The 2001 EU Copyright Directive regards copyrights and exceptions, but not remedies for violations of copyrights.
- Action Using a stand-alone instrument to regulate orphaned works, instead of amending the 2001 Copyright Directive, would cause the separate topic of remedies for copyright violations to be regulated by a separate instrument.
- Goal Separate legal topics should be regulated by separate instruments.
- Values Promoted Regulating separate legal topics with separate instruments promotes the value of legal clarity.

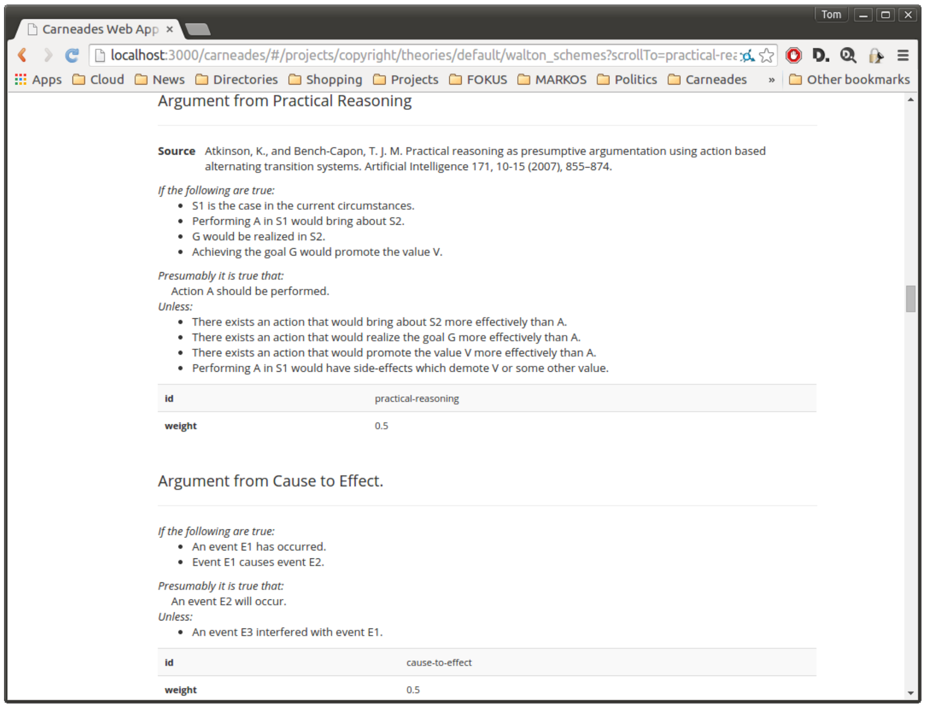
Presumably it is true that:

While we do not support such an approach, to the extent that an orphan works standard is adopted throughout the EU, we recommend that a Community statutory instrument dealing with the problem of orphan works should be a stand-alone instrument. [@SIIA, p. 9]

Properties

value	Unclear
scheme	Argument from Practical Reasoning
strict	false
weight	0.5
id	urn:uuid:4cdf9c48-a104-4773-a3af-d6b6e83ec35d

Figure 9: An Argument from Practical Reasoning



Carneades Web App

localhost:3000/carneades/#/projects/copyright/theories/default/walton_schemes?scrollTo=practical-rea

Apps Cloud News Directories Shopping Projects FOKUS MARKOS Politics Carneades » Other bookmarks

Argument from Practical Reasoning

Source Atkinson, K., and Bench-Capon, T. J. M. Practical reasoning as presumptive argumentation using action based alternating transition systems. Artificial Intelligence 171, 10-15 (2007), 855–874.

If the following are true:

- S1 is the case in the current circumstances.
- Performing A in S1 would bring about S2.
- G would be realized in S2.
- Achieving the goal G would promote the value V.

Presumably it is true that:
Action A should be performed.
Unless:

- There exists an action that would bring about S2 more effectively than A.
- There exists an action that would realize the goal G more effectively than A.
- There exists an action that would promote the value V more effectively than A.
- Performing A in S1 would have side-effects which demote V or some other value.

id	practical-reasoning
weight	0.5

Argument from Cause to Effect.

If the following are true:

- An event E1 has occurred.
- Event E1 causes event E2.

Presumably it is true that:
An event E2 will occur.
Unless:

- An event E3 interfered with event E1.

id	cause-to-effect
weight	0.5

Figure 10: A Scheme for Argument from Practical Reasoning

an issue of it by adding arguments pro or con the assumed statement in the argument graph.

Any time an argument graph is modified, it is automatically reevaluated to update the values of its arguments and statements. A computational model of argument is used to compute the values (Gordon, Prakken, and Walton 2007).

If you do not agree with the result of the evaluation, there are at least three reasons why you may be right and the system's evaluation wrong:

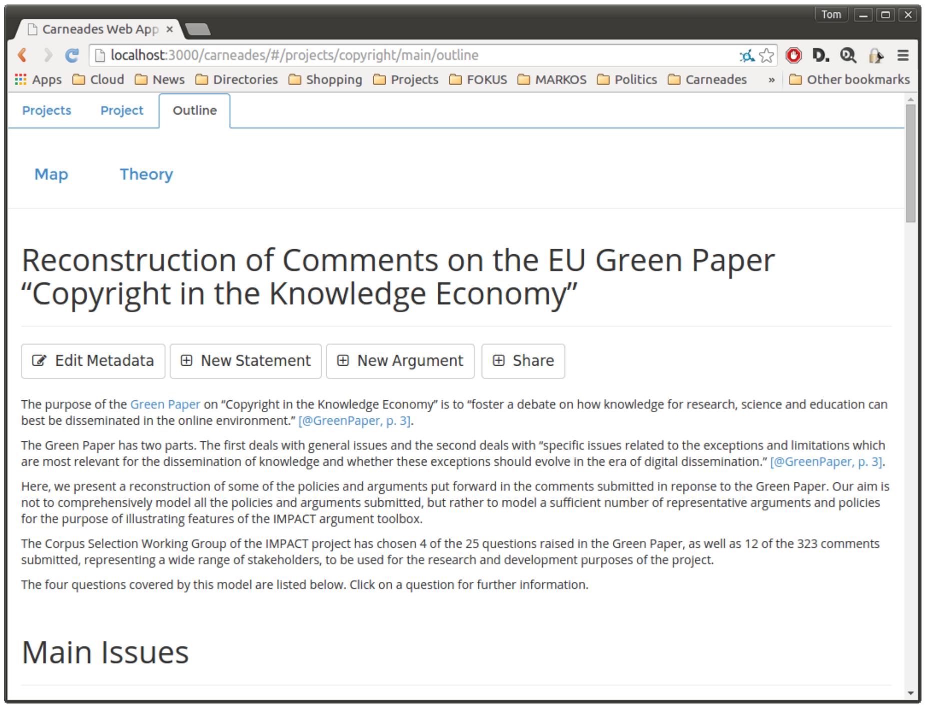
1. Not all relevant arguments have been included in the model, or put forward in the debate. If you are aware of some missing argument, consider adding it to the argument graph.
2. The weights assigned to the statements and arguments may not reflect your opinion. If these weights are averages of weights assigned collectively, by a group of users, you may disagree with the collective view. Minority views can be correct.
3. The formal model of argument we are we are using to compute acceptability may be incorrect. Of course, specialist knowledge is required to assess the correctness of the model. If you, like most people, do not have this knowledge, then we recommend a skeptical but respectful attitude. If you agree with the results of the model, then the model gives you a reason to have more confidence in your opinion. If you do not agree with the results of the model, then you may want to take pause to reconsider your views, even if in the end you do not change your mind.

6 Editing Argument Graphs

To edit an argument graph, first go to its outline, as described in the Chapter entitled Browsing, Visualizing and Evaluating Arguments. An example argument graph page is shown in the figure below.

You can add nodes to the graph by clicking on the "New Statement" or "New Argument" buttons. In both cases you will be presented with a form to enter the required information. The form will be displayed in a new tab, so that you can toggle back and forth between different views of the argument graph will editing. (**Warning**: A known bug in the user interface causes your work to be lost if you change to another tab without first saving your work.) See the Entering New Statements or Entering New Arguments sections for further information.

To add a new argument pro or con some existing statement, go to the statement page and click on the "New Argument" button. The conclusion of the new argument will be set to the existing statement. Then complete the rest of the form as described in the Entering New Arguments section.


Carneades Web App
localhost:3000/carneades/#/projects/copyright/main/outline
Apps Cloud News Directories Shopping Projects FOKUS MARKOS Politics Carneades Other bookmarks
Projects
Project
Outline
Map
Theory
Edit Metadata
New Statement
New Argument
Share

Reconstruction of Comments on the EU Green Paper “Copyright in the Knowledge Economy”

The purpose of the Green Paper on “Copyright in the Knowledge Economy” is to “foster a debate on how knowledge for research, science and education can best be disseminated in the online environment.” [@GreenPaper, p. 3].

The Green Paper has two parts. The first deals with general issues and the second deals with “specific issues related to the exceptions and limitations which are most relevant for the dissemination of knowledge and whether these exceptions should evolve in the era of digital dissemination.” [@GreenPaper, p. 3].

Here, we present a reconstruction of some of the policies and arguments put forward in the comments submitted in reponse to the Green Paper. Our aim is not to comprehensively model all the policies and arguments submitted, but rather to model a sufficient number of representative arguments and policies for the purpose of illustrating features of the IMPACT argument toolbox.

The Corpus Selection Working Group of the IMPACT project has chosen 4 of the 25 questions raised in the Green Paper, as well as 12 of the 323 comments submitted, representing a wide range of stakeholders, to be used for the research and development purposes of the project.

The four questions covered by this model are listed below. Click on a question for further information.

Main Issues

Figure 11: An Argument Graph Page

To edit or delete existing statements or arguments, first go to the page of the statement or argument and then click on the "Edit" or "Remove" button. Editing statements and arguments is done using the same forms used to create new statements and arguments. Deleting a statement will also delete the arguments pro or con the statement. Deleting an argument does not delete the conclusion or premises of the argument. This can leave some statements in the argument graph being unused in any argument.

Warning: There is no undo function, so all editing and delete operations are permanent. However, you will be asked to confirm all delete operations and have the option of cancelling or saving editing operations.

6.1 Reconstructing Arguments

To reconstruct a *new* argument in some source text, one which is not already in the argument graph, follow the procedure below. If instead the source text is *another formulation* of an argument already in the argument graph, you can modify the description of the existing argument to also quote this as an additional source of the argument.

1. Click on the "New Argument" button on the outline page of the argument graph.
2. Copy the text of the argument from the source document and paste it into the description field of the "Metadata"tab of new argument form. You can quote the text of the argument, using Markdown, by preceding each line with a ">" symbol.
3. Go the "Model" tab of argument editor and choose an argumentation scheme to apply, from the pull down list of schemes. The documentation of the selected scheme will be shown and the form will be customized, with fields intialized for each of the premises of the scheme. You can modify the argument however you want, unconstrained by the chosen scheme, for example by deleting or adding premises, or renaming premise roles. The schemes are there to help you, not constrain you. See the Argumentation Schemes chapter for documentation of the initial set of schemes provided with Carneades. These schemes may be modified or extended, or replaced entirely, as described in the Modeling Policies and Argumentation Schemes chapter.
4. Enter the conclusion of the argument, by choosing a statement already in the graph with the same meaning as the conclusion of the argument in the source text. If the needed statement is not listed, create one first following the instructions for entering new statements below.
5. Similarly, add the premises of the argument, by choosing existing statements in the argument graph from the pull-down lists below the role of each premise in the form. Negative premises can be entered by changing the "Positive" property of the premise to "False". If the premise is implicit in

the source text, you can note this by changing the “Implicit” property of the premise to “True”.

6. At the bottom of the form, click the “Save Argument” or “Cancel” button. **Warning:** Any changes you make using the form will not be saved to the argument graph until you execute the “Save Argument” command by clicking on this button.

6.2 Entering New Statements

The form for entering new statements is shown when you click on the “New Statement” button on the outline page of the argument graph page.

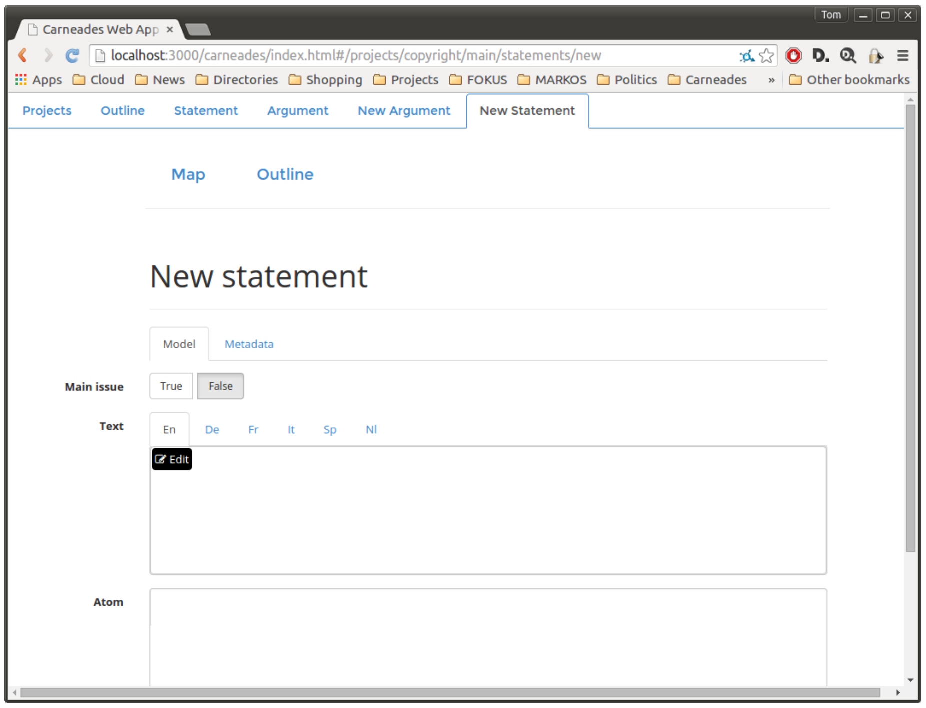


Figure 12: New Statement Form

A new statement should not be entered into the argument graph if the negation of the statement is already in the argument graph. That is, for every proposition P, a single statement node should be entered into the graph to represent *both* P and $\neg P$. It doesn’t matter whether the positive or negative form of the statement is included explictly in the argument graph. Typically the positive form is used, but in some cases you may prefer the negative form. For example, you may prefer to include the sentence “The payment was illegal” explicitly in the argument graph and then represent arguments pro the legality of the payment as argument con the claim the payment was illegal. The best choice may depend on the procedural context of the debate.

Only one form of the statement is required, since arguments pro the negation of the statement are equivalent to arguments con the statement. And premises of arguments can be explicitly negated in argument graphs, and in the forms for entering and editing arguments. This approach has the advantage of reducing the number of statements in the graph up to 50%.

The new statement form is divided into two sections, named “Model” and “Metadata”. On smaller screens these two sections are shown in separate tabs, labelled accordingly. The “model” consists of the text of the statement, in one or more languages, along with some additional properties, described in detail below. The “metadata” consists of generic properties, not specific to statements, such as a title and description. Be careful not to confuse the description of the statement with the text of the statement. The text is a formulation of the statement, whereas as the description provides further information about the statement.

6.2.1 Main Issue

An argument graph should have at least one main issue. These are the issues which are central to the debate. All the other issues are subsidiary issues only important to the extent that they are relevant for resolving one of the main issues.

6.2.2 Statement Text

The “text” property of a statement node of an argument graph is for expressing the statement concisely in natural language. You should always provide such a text. As for descriptions, translations of the text in several languages may be included in the model and the statment may be structured using Markdown. This is the text that will appear in statement boxes in argument maps and in hypertext views of the argument graph. Whereas descriptions are optional and part of the *metadata* about the statement, this text is the content representing the statement itself.

Translations of the text can be entered in several natural languages:

En	English
De	German
Fr	French
It	Italian
Nl	Dutch
Sp	Spanish

6.2.3 Atom: Formalizing Statements using Predicate Logic

Optionally, statements can be formalized in predicate logic, by providing a value for the "atom" property in the form. This is an advanced feature that may be needed only for more specialized application scenarios.

Atoms are formalized in Carneades using the prefix notation of s-expressions ("symbolic expressions"), as in the Lisp family of programming languages.

For example, Socrates is a man could be formalized as `(man Socrates)`. Here, `man` is a unary predicate symbol. The fact that Socrates died in 399 B.C. could be represented using a binary predicate, for example as `(died Socrates -399)`.

The language for Atoms is quite expressive:

- The arity of atoms can be 0 or more, as in predicate logic, not restricted to unary and binary predicates, as in description logic, RDF and OWL. Examples: `(good)`, `(between 0 1 2)`.
- Compound terms may be used. For example `(= 2 (+ 1 1))`. The is, the atom language is not restricted to Datalog.
- Atoms can be higher-order. Example: `(believes Gerald (and (made-of Moon green-cheese) (really-exists Yeti)))`. Notice that compound propositions can be reified as terms, as in this example.

6.2.4 Proof Standard

The proof standard of a statement determines how much proof is required for the statement to be deemed acceptable (presumably true). The proof standard is used by the computational model to argument to compute the acceptability of the statement. Several proof standards are available:

Dialectical Validity (DV) This standard is the only one that does not make use of arugment weights. It is satisfied if at least one pro argument is *in* and no con argument is *in*.

Preponderance of Evidence (PE) This standard is met if at least one pro argument is *in* that weighs more than any *in* con argument.

Clear and Convincing Evidence (CE) This standard is satisfied if the preponderance of evidence standard is met and, in additional, the difference between the strongest *in* pro argument and the strongest *in* con argument is above a certain threshold.

Beyond Reasonable Doubt (BRD) This standard is met if the clear and convincing evidence standard is meet and, in addition, the weight of the weakest *in* con argument is below a certain threshold.

The default proof standard is preponderance of the evidence, and for most applications this proof standard should be sufficient. Note that the preponderance of evidence standard is met whenever the dialectical validity standard is met. If arguments are not weighed, the dialectical validity and preponderance of evidence standards will give the same results. The preponderance of evidence, clear and convincing evidence and beyond reasonable doubt standards are ordered by the amount of proof required, with beyond reasonable doubt requiring the most proof. Whenever one of these standards is met, all of weaker standards are also meet.

6.2.5 Statement Weight

The weight of a statement encodes the extent to which users agree or disagree with the statement, on a scale of 0.0 to 1.0, where 0.0 represents disagree (reject), 0.5 means no opinion and to 1.0 is denotes agree (accept). Statements with a weight greater then or equal to 0.75 and less than 1.0 are *assumed* to be true, until an argument pro or con the statement has been added to the argument graph. Conversely, statements with a weight less than or equal to 0.5 and greater than 0.0 are assumed to be false, until an argument pro or con the statement has been added to the graph.

6.2.6 Statement Metadata

Statements can be annotated with metadata properties, using the Dublin Core metadata elements. The Dublin Core elements are briefly summarized in the Metadata section of this manual.

To add metadata to a statement, go to the “Metadata” tab of the statement editor and complete the form. Multiple values of an element can be entered by listing. Separate the items in the list with semicolon (“;”) characters.

You can use the *description* field of the “Metadata” tab of the statement editor to provide whatever background information you want about the statement. You can structure and format the description, including headers, lists, quotations, hypertext links and other elements, using the Markdown wiki language. The form includes a Markdown editor to make this easier for you.

As for the statement text, translations of descriptions can be entered in several natural languages.

6.3 Editing Statements

To edit an existing statement in an argument graph, first navigate to the statement page and click “Edit Statement” button. This will reveal a form for modifying the statement. This is the same form used to enter new statements, described in the Entering New Statements section of this chapter, but with the fields of the form filled in with the current values of the properties of the statement. After making changes, click on the “Save” button at the bottom of the form, to have the changes stored in the argument graph database, or the “Cancel” button to abort the editing process and retain the prior values of the properties of the statement.

Warning: The changes cannot be undone after saving them.

6.4 Deleting Statements

To delete a statement from an argument graph, first navigate to the statement page and click of the “Remove Statement” button. You will be asked to confirm or cancel the deletion.

Warning: The deletion cannot be undone after it has been confirmed. Deleting a statement also deletes all arguments pro or con this statement, i.e. with this statement as the conclusion of the argument. The statements for the premises of these arguments are not deleted.

6.5 Entering New Arguments

The form for entering new arguments is shown when you click on the “New Argument” button in the menu bar of the argument graph page or a statement page. When used on a statement page, the form for entering the new argument will be initialized with the statement as the conclusion of the argument.

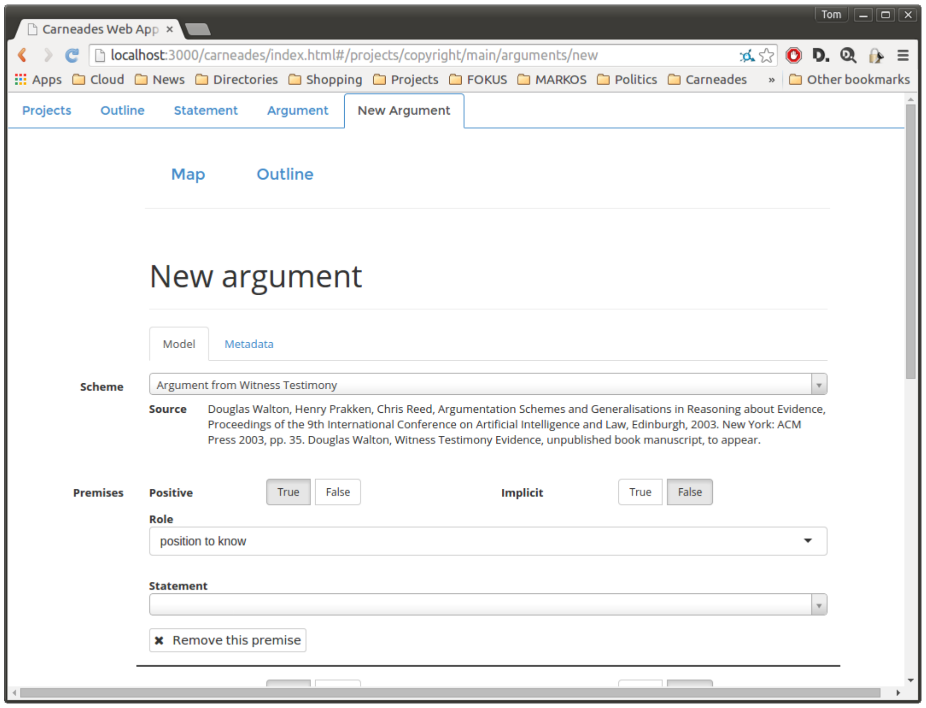
See the section on Reconstructing Arguments for tips about how to proceed when interpreting and modeling arguments in source texts.

6.5.1 Using Argumentation Schemes

Arguments can be entered into the system, without using argumentation schemes. Their use is entirely optional. That said, argumentation schemes can be helpful for interpreting source texts when trying to reconstruct arguments. They serve as templates which can guide you in your task of understanding the source text. For example, if the argument looks like it might be an argument from expert opinion, using the expert opinion scheme will help you to remember the conventional form of this kind of argument. Can the text reasonably be understood as including all the premises of the expert witness scheme? Were any missing premises left implicit by the author, or would this be reading too much between the lines? Does the argument perhaps better fit some other scheme?

See the section on Reconstructing Arguments for tips about how to proceed when interpreting and modeling arguments in source texts.

You can view a list of the available argumentation schemes using the pull-down menu. Type in a string, such as “expert”, to display the schemes containing


Carneades Web App
localhost:3000/carneades/index.html#/projects/copyright/main/arguments/new
Apps Cloud News Directories Shopping Projects FOKUS MARKOS Politics Carneades Other bookmarks
Projects Outline Statement Argument New Argument
Map Outline

New argument

Model | Metadata

Scheme Argument from Witness Testimony

Source Douglas Walton, Henry Prakken, Chris Reed, Argumentation Schemes and Generalisations in Reasoning about Evidence, Proceedings of the 9th International Conference on Artificial Intelligence and Law, Edinburgh, 2003. New York: ACM Press 2003, pp. 35. Douglas Walton, Witness Testimony Evidence, unpublished book manuscript, to appear.

Premises **Positive** [✓ True] False **Implicit** True [✓ False]

Role position to know

Statement

Remove this premise

Figure 13: New Argument Form

the string in the title. Then select the desired scheme from the list. This will display a description of the scheme, perhaps with example arguments, along with references to publications about the scheme. You can go back and select another scheme at any time.

Whenever a scheme is selected, the form will customized to include premises and exceptions fields for the chosen scheme. The roles of the premises and exceptions will be modified to match the selected scheme. The premise roles can be edited. You are not constrained by the scheme.

If you change your mind and select another scheme, any statements you have selected for the conclusion and premises of the argument will remain, but you will need to check whether they are still appropriate and add, modifiy or delete premises as necessary.

Again, the argumentation schemes are meant to be helpful, not get in your way. They do not constrain you to enter arguments matching the schemes. You can, however, check whether arguments correctly instantiate the schemes they have been assigned. See the Validating Arguments section for further information.

6.5.2 Entering and Deleting Premises

In the premise of the argument editor, you can choose an existing statement in the argument graph from the pull down list. You can filter the statements in the list by typing some text in the text field at the top of the list.

You can additional premises by clicking on the "Add Premise" button. Premises can be deleted by clicking on the "Remove this premise" button immediately below each premise.

6.5.3 Strict or Defeasible Arguments

An argument may be *strict* or *defeasible* (not strict). An argument is strict if and only if its conclusion must be true, with no exceptions, if its premises are true, whether or not its premises are in fact true. For example the following arguments are strict:

- The moon is made of green cheese since the moon is made of green cheese and unicorns have horns.
- The figure is a triangle since it has three sides.

The first example argument is of course cyclic, not to mention fanciful. But it is strict nonetheless.

Arguments are defeasible if they are not strict. With defeasible arguments, the conclusion is only *presumably* true when the premises are true. The argument gives us a reason to accept the conclusion, but there may be exceptions or other reasons leading to the opposite conclusion. Here are a couple of famous examples from the computational models of argument literature:

- The object is red since it looks red.
- Tweety flies sinces Tweety is a bird.

The object may look red because, for example, if is being illuminated by a red light. And Tweety the bird may be a penguin, or have a broken wing, and so on.

The conclusion of a defeasible argument need not be *probably* true, in some statistical sense. It is not necessary to have good empirical data allowing us to draw conclusions about what is probably the case in order to make a defeasible argument. The argument only needs to give us some good reason to believe the conclusion, at least until we have heard arguments giving us reasons to the contrary.

Select whether the argument is strict in the form using the radio buttons. By default, arguments are defeasible.

6.5.4 Argument Direction

Arguments can be *pro* or *con* their conclusion. An argument con a statement is semantically equivalent to an argument pro the negation of the statement.

6.5.5 Argument Weight

Argument weights only become important when two or more arguments with contradictory conclusions, that is arguments pro and con some conclusion, come into play. Such arguments are called “rebuttals”. The conflict between the rebuttals is resolved by weighing the arguments and applying proof standards. See the Proof Standard of this chapter for further information.

The weight can be entered directly in the argument editor, using the slider.

6.5.6 Argument Metadata

Arguments can be annotated with metadata properties, using the Dublin Core metadata elements, in the same way as statements. See the Statement Metadata section for further instructions.

You can use the *description* field of the argument to provide whatever background information you want about the argument, including quotations of formulations of the argument in source texts, along with hyperlinks to the sources.

You have the same possibilities for structuring the text of the description, using the Markdown wiki language, and entering translations of the description as you have for statement descriptions. See the Statement Description section of this chapter for further information.

6.6 Editing Arguments

To edit an existing argument in an argument graph, first navigate to the argument page and then click on the "Edit Argument" button. This will open a new tab with an editor for modifying the argument. This is the same form used to enter new arguments, described in the Entering New Arguments section of this chapter, but with the fields of the form filled in with the current values of the properties of the argument. After making changes, click on the "Save" button at the bottom of the form, to have the changes stored in the argument graph database, or the "Cancel" button to abort the editing process and retain the prior values of the properties of the statement.

Warning: The changes cannot be undone after saving them.

6.7 Deleting Arguments

To delete an argument from an argument graph, first navigate to the argument page and then click of the "Remove Argument" button. You will be asked to confirm or cancel the deletion.

Deleting an the argument does not delete the statements used in its conclusion or premises.

Any arguments undercutting this argument will also be deleted, since the conclusion of such an undercutter refers to this argument specifically and can serve no function once the argument has been deleted.

Warning: The deletion cannot be undone after it has been confirmed.

6.8 Evaluating Argument Graphs

The term "argument evaluation" has two different but related meanings. The broader meaning, described in the Evaluating Arguments section of this manual, concerns the process of critically assessing the validity of arguments, by

1. revealing implicit premises
2. validating whether the arguments are formally correct, by instantiating accepted argumentation schemes
3. asking appropriate critical questions, depending on the schemes applied

4. and determining which claims are acceptable, taking into consideration the assumptions of the users and their collective assessment of the relative weights of conflicting pro and con arguments.

The final step of this process, determining which claims are acceptable, is what we mean by argument evaluation in this section. Perhaps a better term for this narrower conception of argument evaluation, to avoid confusion, would be "argument labeling", since the result of this kind of argument evaluation is to label the statement and argument nodes of an argument graph as being "in", "out" or "undecided", where:

in means the statement is *acceptable* (presumably true) or the argument is *applicable*, because all of its premises are in and it has not been undercut by an in argument.

out means the negation of the statement is acceptable (equivalently: the statement itself is not acceptable) or the argument is not applicable.

undecided means the statement or argument is neither in nor out.

Argument graphs are evaluated using the computational model of structured argument presented in (Gordon, Prakken, and Walton 2007). The argument graph is automatically re-evaluated, updating the labels of the argument and statement nodes, whenever the argument graph is modified. This includes not only changes to the weights of the statements and arguments, but also changes to the assigned proof standards and, the addition or deletion of any arguments. (Additional arguments, not only deletions, can cause the labels to change, since acceptability is a *nonmonotonic* inference relation.)

7 Argumentation Schemes

Argumentation schemes are represented using a high-level scheme (rule) language embedded in the Clojure programming language.

Argumentation schemes generalize the notion of an inference rule to cover defeasible as well as strict reasoning patterns. We use the term "scheme" instead of "rule" to emphasize that the rules are usually defeasible. The scheme language is expressive enough to represent axiomatizations (i.e. axioms and inferences rules) of theories in many domains, including laws, regulations and policies, in addition to argumentation schemes per se.

Computational models of theories have been called many things in computer science, including "knowledge bases" and "deep conceptual" or "semantic" models. We prefer the term "theory" to "knowledge-base" because "knowledge" may suggest consensus or truth, while it is clearer that theories may be controversial or contested.

The scheme language is similar to logic programming languages such as Prolog. Any Prolog rule ("clause") can be represented in Carneades in a straight-forward way. The rule language has some additional features for representing argumentation schemes, such as scheme variables ranging over atoms and a means to represent premise roles (e.g. "major", "minor"). Moreover, schemes in this language can be annotated with meta-data and documentation, in multiple natural languages.

Carneades includes an inference engine which is able to automatically apply theories to construct arguments. Using a high-level declarative language for representing theories, and generating arguments from these theories, makes it easier for domain experts to read and validate the theories.

Theories may organized in a hierarchical structure of sections and subsections, with schemes included at any level. Like schemes, the theory as a whole and each of its sections can be annotated with its own meta-data and natural language description. These features facilitate self-documenting and "isomorphic" modeling. The source text of the schemes, policies or legislation can be included within the model, in the same files, in such as way as to preserve the hierarchical structure of the sources. This makes it easier to maintain the model as the source documents are modified, since there can be a one-to-one correspondance between sections of the source text and sections of the model.

Theories are represented in Clojure source code files. Clojure packages can be used to modularize, combine and reuse theories.

The theory language is an executable knowledge-representation language, with its own semantics and an inference engine implementing this semantics. It is not intended to be used as an "interchange format" for exporting and importing theories among diverse formalisms with varied semantics.

The semantics of theories are defined by mapping instantiations of argumentation schemes to argument graphs and, in turn, evaluating these graphs using the computational model of structured argument presented in (Gordon, Prakken, and Walton 2007).

7.1 Modeling Argumentation Schemes

In this section we illustrate how to use the language to represent a version of the scheme for arguments from practical reasoning.

A domain theory is represented by first defining a *language* (dictionary) of symbols, denoting predicates and terms, and then a set of inference rules, called *schemes*, using this language.

The language is represented as a map from symbols to predicates and individuals, in the Clojure programming language. Each symbol of the language is mapped to a structure with fields for the symbol of the predicate (redundantly), the arity of the predicate (i.e. the number of columns in a tabular representation of the relation denoted by the predicate) and an optional number of forms for expressing statements and questions about this predicate in one or more natural languages.

To illustrate, below are the definition of some of the predicates of the language used a version of the schemes for practical reasoning.

```
(def L
  {'circumstances
    (make-predicate
     :symbol 'circumstances
     :arity 1
     :forms
     {:en (make-form
           :positive "The circumstances are %s."
           :negative "The circumstances are not %s."
           :question "Are the circumstances %s?")})

   'should-do
   (make-predicate
    :symbol 'should-do
    :arity 2
    :forms
    {:en (make-form
          :positive "In cirumstances %s, we should do %s."
          :negative "In circumstances %s, we should not do %s."
          :question "In circumstances %s, should we do %s?")}),

   'results-in
   (make-predicate
    :symbol 'results-in
    :arity 3
    :forms
    {:en (make-form
          :positive "In circumstances %s doing %s would
             result in circumstances %s."
          :negative "In circumstances %s doing %s would
                    not result in circumstances %s."
          :question "In circumstances %s would doing %s
                    result in circumstances %s?")})

   'avoids
   (make-predicate
    :symbol 'avoids
    :arity 3
```

```
 :forms
 {:en (make-form
       :positive "In circumstances %s not doing %s
                  would avoid circumstances %s."
       :negative "In circumstances %s not doing %s
                  would not avoid circumstances %s."
       :question "In circumstances %s would not doing %s
                  avoid circumstances %s?")})

'realises
(make-predicate
 :symbol 'realises
 :arity 2
 :forms
 {:en (make-form
       :positive "Circumstances %s would realize goal %s."
       :negative "Circumstances %s would not realize goal %s."
       :question "Would circumstances %s realize goal %s?")})

'promotes
(make-predicate
 :symbol 'promotes
 :arity 2
 :forms
 {:en (make-form
       :positive "Achieving %s would promote the value %s."
       :negative "Achieving %s would not promote the value %s."
       :question "Would achieving %s promote the value %s?")}),

'demotes
(make-predicate
 :symbol 'demotes
 :arity 2
 :forms
 {:en (make-form
       :positive "Achieving goal %s would demote the
                  value %s."
       :negative "Achieving  goal %s would not demote the
                  value %s."
       :question "Would achieving goal %s demote the
                  value %s?")})

})
```

An argumentation scheme is represented as a structure with six fields:

1. id
2. header
3. conclusion
4. premises
5. exceptions, and
6. assumptions.

The id is a term in the language used to reify argumentation schemes and represent statements about argumentation schemes in domain models. The header enables metadata about the scheme to be represented (e.g. title, description). Descriptions can be represented in multiple natural languages. The conclusion is a formula schema, which may contain schema variables. Schema variables are represented by symbols beginning with a question mark, e.g `?Ag`, `?A`, and `?G`, and range over both terms and propositions. Thus, the conclusion of an argumentation scheme can be a schema variable. This feature is needed for representing schemes, such as arguments from expert witness testimony, whose conclusion may be any proposition whatsoever.

The premises, exceptions and assumptions fields of schemes are vectors of premise structures, where each premise has the following properties:

role A string naming the role of the premise in the argumentation scheme, e.g. “major”, “minor”, “circumstances”, “goal”.

positive Boolean. False if the premise is negated. Default: true.

statement An atom formalizing the propositional content of the statement.

Next, using this language we formally define positive and negative versions of a scheme for practical reasoning. Schemes may be organized in an hierarchy of *sections*, each section with its own metadata. But since there are only four schemes in this example, sections are not illustrated here.

```
(def S
  [(make-scheme
    :id 'pras1
    :header (make-metadata :title "Practical Reasoning Scheme")
    :conclusion '(should-do ?S1 ?A)
    :premises
    [(make-premise :role "circumstances"
                   :statement '(circumstances ?S1))
     (make-premise :role "action"
                   :statement '(results-in ?S1 ?A ?S2))
     (make-premise :role "goal" :statement '(realizes ?S2 ?G))
     (make-premise :role "value" :statement '(promotes ?G ?V))])

   (make-scheme
    :id 'pras2
    :header
    (make-metadata :title "Negative Practical Reasoning Scheme")
    :conclusion '(not (should-do ?S1 ?A))
    :premises
    [(make-premise :role "circumstances"
                   :statement '(circumstances ?S1))
     (make-premise :role "action" :statement '(avoids ?S1 ?A ?S2))
     (make-premise :role "goal" :statement '(realizes ?S2 ?G))
     (make-premise :role "value" :statement '(demotes ?G ?V))])
   ])
```

Now, let's us package the language and schemes together in a theory. Conceptually, a theory is a set of propositions. But since the set may be infinite, it is more convenient to represent theories intensionally, as a set of axioms and inference rules. We call the inference rules "argumentation schemes", because they may be defeasible, to distinguish them from the inference rules of classical logic, which are all non-defeasible (strict).

A theory is modelled as a structure having the following fields:

header metadata (e.g. title, description) about the theory. Descriptions can be in multiple languages and can be arbitrarily long, structured texts, represented using the Markdown wiki language.

language The formal language of the theory; a dictionary mapping symbols to terms and predicates.

schemes Strict and defeasible inference rules.

sections A sequence of sections, which in turn consist of a header, schemes and (sub)sections, enabling theories to be organized hierarchically, similar to the hierarchical structure of books and articles.

references A sequence of metadata structures, for providing bibliographic information about source documents.

Next, we complete this illustration of how to implement argumentation schemes, by defining `theory1` to be the following theory:

```
(def theory1
  (make-theory
   :header
   (make-metadata
    :title "Reconstruction of Liverpool Argumentation Schemes"
    :creator "Tom Gordon"
    :publisher "Fraunhofer FOKUS"
    :date "2012"
    :description {:en "This is a reconstruction of the version of
```

```
                the Liverpool schemes in [@Atkison2012a]."})

   :language L
   :schemes S
   :references
   {"Atkison2012a"
    (make-metadata
     :creator "Katie Atkinson, Adam Wyner and Trevor Bench-Capon"
     :title "Report No. D5.2 -- Report on Prototype 1 of
         Structured Consultation Tool"
     :publisher "IMPACT"
     :date "2012")}))
```

8 System Administration

8.1 Building and Installing from Source Code

1. Use a Git client to clone a copy of the Carneades source code from the Carneades GitHub repository.
2. Follow the instructions in the following README file:

`carneades/src/CarneadesWeb/README.md`

8.2 System Configuration

To modify the configuration, edit the `config/carneades.clj` file in the installation directory.

The configure files are Clojure source files, with the properties represented as a Clojure map.

The `:projects` property provides the full path name of the directory used to store Carneades projects. The default directory is the `projects` directory of the installation directory.

Example: `:projects "/usr/local/carneades/projects`

Additional properties can be configured for each project, as described in the next section.

8.3 Project Structure and Configuration

Carneades projects are stored in the directory stated in the `.carneades.clj` configuration file. Each project is a directory with the following structure:

```
properties.clj
databases/
theories/
documents/
```

The `properties.clj` file defines the attributes of the project. The properties are represented as a Clojure map in the file. Here is the contents of an example properties.clj file:

```
{
:title "My First Carneades Project"
:creation-date "January 10, 2015"
:theory "default/walton_schemes"
:description {:en "This is my first Carneades project."
              :de "Dies ist mein erstes Carneades Projekt."}
}
```

The `:title` and `:creation-date` properties should be self-explanatory. Their values are strings. (The creation date is not constrained to some specific format for dates.)

The `:description` is a Clojure map, to allow multiple descriptions in different languages. In the example there are descriptions in English and German. Descriptions can formatted using Markdown syntax.

The `:theory` property specifies which theory to use to construct and reconstruct arguments, using the rule-based inference engine, or interactively, using the argument editor.

Theories are represented using Clojure data structures, in Clojure files with the usual ".clj" filename extension. In the property map, the values of the `:theory` property should be the file name of the Clojure file containing the theory, but without the ".clj" file name extension. The theory name is resolved relative to the "theories" directory of the given project, or the current project being configured if no project is named. In the example, the `:theory` property is "default/walton_schemes". This refers to the theory in the "default/theories/walton_schemes.clj" file in the projects directory.

The `databases/` directory stores all the database files of the project, including a database for each argument graph of they project. (A project may have than one argument graph. For example, when using the policy analysis tool, an argument graph is created for each case.) The database files are in the format used by the H2 database engine for the Java Virtual Machine.

The `theories/` directory contains the Clojure source files of the theories of the project.

Finally, the `documents/` directory can be used to store copies of source documents, documentary evidence, or any other project files.

<!– include “scenarios.md” –!

9 Builtin Argumentation Schemes

This chapter presents the argumentation schemes included with the distribution of the Carneades system. The system is pre-configured to use these schemes, but you can configure the system to use other schemes, or modify these schemes to meet your requirements.

The argumentation schemes are shown here in pseudocode for readabililty. See the Modeling Argumentation Schemes chapter for a description of the syntax used to formally define the schemes.

The schemes can be viewed online, using Carneades, by clicking on the “Theory” link near the top of an argument outline page, if the project has not been configured to use some other schemes.

Most of the schemes here are derived the book “Argumentation Schemes” (Walton, Reed, and Macagno 2008). The schemes for arguments from credible source and practical reasoning are based on (Wyner, Atkinson, and Bench-Capon 2012) and (Atkinson and Bench-Capon 2007), respectively.

The schemes from these sources been modified to fit the Carneades computational model of argument. For example, generic critical questions which undermine premises, undercut the argument or rebut its conclusion, have been omitted, since these critical questions apply to all defeasible arguments in Carneades.

All the argumentation schemes presented here are defeasible *unless* they have been explicitly declared to be strict.

9.1 Argument from Position to Know

```
id: position-to-know

conclusion: S

premises:
    major: W is in a position to know about things in a certain
```

```
        subject domain D.
    minor: W asserts that S is true.
    domain: S is in domain D.

exceptions:
    CQ1: W is dishonest.
```

9.2 Argument from Credible Source

```
id:  credible-source

conclusion: S

premises:
    source: W is a credible source about domain D.
    assertion: W asserts S.
    domain: S is in domain D.

exceptions:
    CQ1: W is biased.
    CQ2: W is dishonest.
    CQ3: Other credible sources disagree with S.
```

9.3 Argument from Witness Testimony

```
id: witness-testimony

conclusion: A

premises:
    position to know: W is in a position to know about things in a
        certain subject domain A.
    truth-telling: Witness W believes A to be true.
    minor: W asserts that A is true.

assumptions:
    CQ1: A is internally consistent.

exceptions:
    CQ2: A is inconsistent with the facts.
    CQ3: A is inconsistent with the testimony of other witnesses.
    CQ4: W is biased.
    CQ5: A is implausible.
```

9.4 Argument from Expert Opinion

```
id: expert-opinion

conclusion: A

premises:
   major: Source E is an expert in subject domain S.
   domain: A is in domain S.
   minor: E asserts that A is true.

exceptions:
    CQ1: E is untrustworthy.
    CQ2: A is inconsistent with the testimony of other witnesses.
    CQ3: A is based on evidence.
```

9.5 Argument from Analogy

```
id: analogy

conclusion: S

premises:
    major: Case C1 is similar to the current case.
   case: S is true in case C1.
   minor: E asserts that A is true.

exceptions:
    CQ1: There are relevant differences between case C1 and the
        current case.
    CQ2: S is false in case C1, which is more on point
        than case   C2.
```

9.6 Argument from Precedent

```
id: precedent

conclusion: S

premises:
    major: Case C1 is similar to the current case.
    ratio: Rule R is the ratio decidendi of case C1.
    conclusion: Rule R has conclusion S.
```

```
exceptions:
    CQ1: There are relevant differences between case C1 and the
        current case.
    CQ2: Rule R is inapplicable in this case.
```

9.7 Argument from Verbal Classification

```
id: definition-to-verbal-classification

strict: true

conclusion: O is an instance of class G.

premises:
    individual: O satisfies definition D.
    classification: Objects which satisfy definition D are
        classified as instances of class G.
```

9.8 Argument from Definition to Verbal Classification

```
id: definition-to-verbal-classification

strict: true

conclusion: O is an instance of class G.

premises:
   individual: O satisfies definition D.
   classification: Objects which satisfy definition D are
        classified as instances of class G.
```

9.9 Defeasible Modus Ponens

```
id: defeasible-modus-ponens

conclusion: B

premises:
    major: If A is true then presumably B is also true.
    minor: A
```

9.10 Argument from an Established Rule

```
id: established-rule

conclusion: C

premises:
    major: Rule R has conclusion C.
    minor: Rule R is applicable.

assumptions:
    CQ1: Rule R is valid.
```

9.11 Argument from Positive Consequences

```
id: positive-consequences

conclusion: Action A should be performed.

premises:
    major: Performing action A would have positive consequences.
```

9.12 Argument from Negative Consequences

```
id: negative-consequences

conclusion: Action A should not be performed.

premises:
    major: Performing action A would have negative consequences.
```

9.13 Argument from Practical Reasoning

```
id: practical-reasoning

conclusion: A1 should be performed.

premises:
    circumstances: S1 is currently the case.
    action: Performing A1 in S1 would bring about S2.
    goal: G would be realized in S2.
    value: Achieving G would promote V.
```

```
assumptions:
    CQ1: V is indeed a legitimate value.
    CQ2: G is a worthy goal.
    CQ3: Action A1 is possible.

exceptions:
    CQ4: There exists an action that, when performed in S1, would
        bring about S2 more effectively than A1.
    CQ5: There exists an action that, when performed in S1,  would
        realize G more effectively than A1.
    CQ6: There exists an action that, when performed in S1, would
        promote V more effectively than A1.
    CQ7: Performing A1 in S1 would have side-effects
        which demote V or some other value.
```

9.14 Argument from Cause to Effect.

```
id: cause-to-effect

conclusion: Event E2 will occur.

premises:
    minor: An event E1 has occurred.
    major: Event E1 causes event E2.

exceptions:
    CQ1: An event E3 interferred with E1.
```

9.15 Argument from Correlation to Cause

```
id: correlation-to-cause

conclusion: Event E1 causes event E2.

premises:
    major: Events E1 and E2 are correlated.

assumptions:
    CQ1: There exists a theory explaining the correlation
        between E1 and E2.

exceptions:
    CQ2: E3 causes E1 and E2.
```

9.16 Argument from Sunk Costs

```
id: sunk-costs

conclusion: Action A should be performed.

premises:
    costs: The costs incurred performing A thus far are C.
    waste: The sunk costs of C are too high to waste.

assumptions:
    CQ1: Action A is feasible.
```

9.17 Argument from Appearance

```
id: appearance

conclusion: O is an instance of class C.

premises:
    minor: O looks like a C.
```

9.18 Argument from Ignorance

```
id: ignorance

conclusion: S

premises:
    major: S would be known if it were true.
    minor: S is known to be true.

exceptions:
    CQ1: The truth of S has not been investigated.
```

9.19 Argument from Abduction

```
id: abduction

conclusion: H

premises:
    observation: observed ?S
    explanation: Theory T1 explains S.
    hypothesis: T1 contains H as a member.

exceptions:
    CQ1: T2 is a more coherent explanation than T1 of S.
```

9.20 Ethotic Argument

```
id: ethotic

conclusion: S

premises:
    assertion: P asserts that S is true.
    trustworthiness: P is trustworthy.
```

9.21 Slippery Slope Argument

This version of the slippery slope scheme is intended to be used together with the argument from negative consequences schema, to derive the conclusion that the action should not be performed.

Notice that the scheme is represented by *two* Carneades schemes, one for the base case and one for the inductive step.

9.21.1 Base Case

```
id: slippery-slope-base-case

conclusion: Performing action A would have negative consequences

premises:
    realization: Performing A would realize event E.
    horrible costs: Event E would have horrible costs.
```

9.21.2 Inductive Step

```
id: slippery-slope-inductive-step

conclusion: Event E1 would have horrible consequences

premises:
    causation: Event E1 causes E2.
    consequences: Event E2 would haves horrible consequences.
```

10 Credits

- Conception and Design
  - Tom Gordon
  - Douglas Walton
- Programming
  - Pierre Allix
  - Stefan Ballnat
  - Tom Gordon
  - Matthias Grabmair
  - Sebastian Kaiser

- Funding and Support
  - Canadian Social Science and Humanities Research Council, 2007-09 and 2012-2017, with Douglas Walton
  - Standardized Transparent Representations in order to Extend Legal Accessibility (Estrella, FP6-IST-027655), 2006-2008.
  - Quality Platform for Open Source Software (Qualipso, FP6-IST-034763), 2006-2010.
  - Google Summer of Code, 2008.
  - Integrated Method for Policy making using Argument modelling and Computer assisted Text analysis (IMPACT, FP7-IST-247228), 2010-2012.
  - Simple Procedures for Cross-Border Services (SPOCS, CIP-238935), 2009-2012
  - The MARKet for Open Source – An Intelligent Virtual Open Source Marketplace (MARKOS, FP7-ICT-317743), 2012-2015.
  - Policy Compass (FP7-ICT-612133), 2013-2016.
  - Enhanced Government Learning (EAGLE, FP7-ICT-619347), 2014-2017.

11 User Manual License

References

Atkinson, Katie, and Trevor J. M. Bench-Capon. 2007. “Practical Reasoning as Presumptive Argumentation Using Action Based Alternating Transition Systems.” *Artificial Intelligence* 171 (10-15): 855–74.

Gordon, Thomas F, Henry Prakken, and Douglas Walton. 2007. “The Carneades Model of Argument and Burden of Proof.” *Artificial Intelligence* 171 (10-11): 875–96.

Walton, Douglas. 2006. *Fundamentals of Critical Argumentation*. Cambridge, UK: Cambridge University Press.

Walton, Douglas, Chris Reed, and Fabrizio Macagno. 2008. *Argumentation Schemes*. Cambridge University Press.

Wyner, Adam, Katie Atkinson, and Trevor Bench-Capon. 2012. “A Functional Perspective on Argumentation Schemes.” In *Proceedings of the 9th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2012)*, edited by Peter McBurney, Simon Parsons, and Iyad Rahwan, 203–22.