Legal Reasoning with Argumentation Schemes

Thomas F. Gordon
Fraunhofer FOKUS
Berlin, Germany
thomas.gordon@fokus.fraunhofer.de

Douglas Walton
University of Windsor
Centre for Research on Reasoning,
Argumentation, and Rhetoric
Windsor, Canada
dwalton@uwindsor.ca

ABSTRACT

Legal reasoning typically requires a variety of argumentation schemes to be used together. A legal case may raise issues requiring argument from precedent cases, rules, policy goals, moral principles, jurisprudential doctrine, social values and evidence. We present an extensible software architecture which allows diverse computational models of argumentation schemes to be used together in an integrated way to construct and search for arguments. The architecture has been implemented in Carneades, a software library for building argumentation tools. The architecture is illustrated with models of schemes for argument from ontologies, rules, cases and testimonial evidence and compared to blackboard systems for hybrid reasoning.

1. INTRODUCTION

We present an extensible software architecture which allows diverse computational models of argumentation schemes to be used together in an integrated way to construct and search for arguments. To make this paper self-contained, we begin by summarizing the mainstream modern conception of argument in philosophy [37], and the computational model of argument we have developed for the Carneades system[17].[1]

An argument links a set of statements, the premises, to another statement, the conclusion. The premises may be labelled with additional information, about their role in the argument. Aristotle's theory of syllogism, for example, distinguished major premises from minor premises. The basic idea is that the premises provide some kind of support for the conclusion. If the premises are accepted, then the argument, if it is a good one, lends some weight to the conclusion. Unlike instances of valid inference rules of classical logic, the conclusion of an argument need not be necessarily true if the premises are true. Moreover, some of the premises of an argument may be implicit. An argument with implicit premises is called an *enthymeme* [37, p. 178].

Arguments have been modeled in various ways computationally: abstractly, with no structure [12], as sets of assumed propositions [31, 15, 7], and as proofs or derivations in some logical calculus [25]. We prefer to model arguments in a way which is closer to the concept of an argument in philosophy, as tuples of the type (list[premise], statement) where list[premise] denotes the type of a list of premises, and the statement is the conclusion of the argument. A premise is either a statement, exception or assumption. A statement is some representation of a proposition. We assume a complement function of type statement $\Rightarrow$ statement such that complement(p) denotes the proposition which is the logical negation of the proposition denoted by p and complement (complement(p)) equals p. Exceptions and assumptions are denoted unless p and assuming p, respectively, where p is a statement, and represent the *critical questions* which may be asked to undercut an argument [17]. An argument is a structure (P,c), denoted c since [p1, ..., pn].[2]

Let a1 be the argument q since P and let r = complement(q). We say a1 is an argument *pro* q and *con* r. Note that a1 is not a con argument for *every* proposition inconsistent with q since checking consistency is either undecidable, in the case of first-order logic, or at least intractable, in the case of propositional logic. We assume there is a simple syntactic test of checking whether r = complement(q). If two propositions are inconsistent but not complements, the interested party has the burden of proving this by asserting an argument of the form $\neg$p since [$\neg$p $\vee$ $\neg$q, q] and then proving the premises. This is an example of our general strategy of allocating intractable and undecidable problems to the parties.

To give an example, the classic argument about Socrates being mortal can be denoted as mortal(Socrates) since man(Socrates). This is an example of an ethymeme, since the major premise, $\forall$ x . man(x) $\rightarrow$ mortal(x), is implicit.

Argumentation is the *process* of putting forth arguments to determine the acceptability of propositions. Argumentation processes are typically dialogs involving two or more agents, but may also be monological. Procedural norms, called argumentation protocols, regulate the process, to help promote values such as rationality, fairness, efficiency and transparency. Which protocol is appropriate depends on

[1]The Carneades system is open source software available at http://carneades.berlios.de.

[2]The notation used in this article for logic and mathematics is intended to make formulas more pleasant to read using long, mnemonic identifiers. This article contains no programming language code.

the type of the process and its goals. At a very high level of abstraction, an argumentation process can be viewed as a sequence of stages, where each stage consists of the set of arguments which have been put forward thus far in the process, along with other information, such as a record of claims or *commitments* [21].

At each stage of the argumentation process, an effective method (decision procedure) is needed for testing whether some proposition at issue is presumably true given the arguments of the stage and a set of assumptions. The assumptions could represent undisputed facts, the current consensus of the participants, or the commitments or beliefs of some agent, depending on the task. This determination may depend on the *proof standard* applicable to the proposition at issue, given the dialogue type and its protocol. What is needed, simplifying somewhat, is a decidable function, let us call it acceptable, of the type (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ Boolean. An acceptability function of this type is provided by the Carneades model of argument [17].[3]

Carneades' "relational core", stripped of its support for multiple proof standards and allocating various kinds of burden of proof, was shown by Prakken in [17] to be very similar to the ambiguity-blocking variant of Defeasible Logic (DL) [24]. Governatori [18] investigated the relationship between DL and Dung's abstract argumentation framework [12]. He proved that the ambiguity-propagating variant of DL instantiates Dung's grounded semantics, for a version of DL without strict rules. He also proved this result for the ambiguity-blocking variant of DL, but to obtain this result he had to change Dung's notion of acceptability of an argument with respect to a set of arguments. Prakken conjectures that, because of its ambiguity-blocking character, the relational core of Carneades cannot be proven to instantiate any of the four semantics of Dung's abstract argumentation framework without changing Dung's notion of acceptability.

At each stage of an argumentation process, a common task will be to try to find or construct additional arguments which can be put forward to make an acceptable statement unacceptable, or vice versa. Whereas the problem of checking the acceptability of a statement given a finite set of arguments is decidable, the problem of finding or constructing arguments is in general ill-defined, open and undecidable. While models of sources of arguments can be constructed, for example a rule-base representing knowledge about some domain, and these models can be used to generate arguments, it is not usually possible, as a practical matter, to model all relevant sources of arguments. Moreover, models are always abstractions and thus subject to critical questions about their adequacy for the task at hand. Despite these limitations, models are essential for constructing arguments. Since models are abstractions developed for particular purposes and provide only specialized views onto reality, several models may be relevant and useful for an argumentation process, depending on the issues.

For example, a legal case may raise issues requiring argument from precedent cases, rules, policy goals, moral principles, jurisprudential doctrine, social values and evidence. Argumentation tools are needed for helping people to argue their cases effectively in court and administrative proceedings. Tools for reconstructing, visualizing and evaluating arguments, while important, are not sufficient in this context. A party to a legal or administrative proceeding is not in the role of an analyst trying to understand a previous dialogue, but rather in the role of an advocate needing to construct and put forward effective arguments as the dialogue unfolds.

How can arguments constructed from multiple, hybrid models be integrated, aggregated and evaluated? Our thesis is that argumentation schemes enable the methods used to contruct arguments to be separated and abstracted from the form and content of arguments. Argumentation frameworks [12, 17] for aggregating and evaluating arguments depend only on relationships between arguments, not the methods used to construct them. This separation makes it possible to use a variety of hybrid methods to construct arguments and then to aggregate and evaluate these hybrid arguments using a common argumentation framework. Each argumentation scheme, in its role as a method, implements a common protocol for mapping an issue and a model of some information or knowledge to a set of arguments about the issue. A set of such argumentation schemes induces a search space over sets of arguments. Heuristic methods can be used together with an argumentation framework, such as Carneades, to search this space for sets of arguments in which some goal statement or argument is acceptable.

The rest of this article is organized as follows. The next section introduces the concept of argumentation schemes from philosophy and presents our computational model of schemes as implementations of an abstract protocol for constructing arguments from models. The following sections illustrate this idea with outlines of computational models of schemes for argument from ontologies, rules, cases and witness testimony, respectively. A section on related work compares our work with other architectures for hybrid reasoning, in particular blackboard systems. The article concludes by reiterating the main results and suggesting ideas for future work.

2. ARGUMENTATION SCHEMES

An argumentation scheme is a pattern which can be used both to create arguments, by instantiating the pattern, and to classify arguments, by matching a given argument to the pattern.[4] For example, here is a version of the scheme for argument from position to know [37, p. 85].

Argument from Position to Know

Position to Know Premise. Person p is in a position to know whether the statement s is true of false.

Assertion Premise. p asserts that s is true (false).

Conclusion. s is true (false).

Just as 'argument' can mean both an argumentation *process* and a *relationship* between a set of premises and a conclusion, so too can argumentation schemes be viewed from

[3]The decidability of the acceptable function does not imply decidability for the task of determining whether some statement is acceptable given all possible arguments which can be constructed from a body of information, using all possible argumentation schemes. Again, intractable and undecidable tasks are allocated to the parties in our approach.

[4]A scheme with free variables is equivalent to the set of schemes which can be generated by systematically instantiating the variables by constants denoting individuals.

two perspectives, as tools for 1) argument *reconstruction* and classification, and 2) argument *construction* or generation.

The classification function of argumentation schemes plays an important role during the process of reconstructing arguments from natural language texts, for example to help identify implicit premises. Argument reconstruction can be viewed as an application of abductive reasoning. Argumentation schemes are used as patterns to construct a set of alternative interpretations of the text, where each interpretation is an argument instantiating some scheme. These interpretations form the set of hypotheses for abductive reasoning. The task is then to choose the interpretation among the hypotheses which best explains the text and other evidence. Once the argument has been reconstructed, the scheme can also be used to help identify missing premises needed to evaluate the argument. For example, the scheme for argument from position to know could be used to help interpret the text "Markley lives in California and tells me the weather is beautiful there." as an argument for the proposition that the weather is good in California since Markley has asserted this and is in a position to know this. One kind of computational model suitable for supporting this classification task would be a formal ontology of argumentation schemes, represented in some version of description logic [4], such as the Web Ontology Language [23]. Rahwan and Banihashemi have developed a model of argumentation schemes of this type [29].

The other way to look at argumentation schemes is as tools for constructing or inventing new arguments to put forward in dialogues. For example, if the quality of the weather in California is at issue, one could apply the scheme for argument from position to know by interviewing people who live in California about the weather there. From this perspective, argumentation schemes are methods for constructing arguments. The two perspectives are complementary. The result of applying a scheme qua method is an argument which is an instance of the scheme qua pattern. When necessary to distinguish these two meanings, we use the terms "argument generator" and "argument patterns" for the method and template senses of argumentation schemes, respectively.

The focus in this article is on computational models of argumentation schemes in the sense of argument generators. To allow models of diverse argumentation schemes to be used together, we first develop a protocol for argument generators and require every argument generator to implement this protocol. Intuitively, the job of an argument generator is to construct a sequence of arguments which *may* be useful for proving or disproving a goal statement. A set of argument generators induces a search space. Each argument produced by an argument generator can be used to construct a successor state in the space. The space can be searched heuristically for states in which the statement at issue is either acceptable or not, depending on whether the goal is to prove or disprove the statement.

More formally, an argument generator is a function of the type (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ stream[argument], where the type stream[argument] denotes a possibly infinite sequence of arguments.[5]

[5]The signature of argument generators in Carneades is actually a bit more complicated than this, since some heuristic control information is passed to the generator and a set of substitutions, mapping logical variables to terms, is returned along with each argument in the resulting stream of arguments. To keep things simpler, for expository purposes, we assume in this paper that statements are represented as ground formulas in first-order predicate calculus, i.e. formulas which do not contain variables, but only constants.

3. ARGUMENT FROM ONTOLOGY

In computer science, an ontology is a representation of concepts and relations among concepts, typically expressed in some decidable subset of first-order logic, such as description logic [4]. Such ontologies play an important role in integrating systems, by providing a formal mechanism for sharing terminology, and also in the context of the so-called Semantic Web [5] for providing machine-processable metadata about web resources and services. There is a World Wide Web standard for modeling ontologies, based on description logic, called the Web Ontology Langauge (OWL) [23]. In this section, we outline one way to construct arguments from ontologies modeled using the Description Logic Programming (DLP) subset of description logic [19].

Suppose we have the following simple DLP ontology, represented using standard description logic syntax:

$$\begin{aligned}&\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male}\\&\mathsf{Woman} \equiv \mathsf{Human} \sqcap \mathsf{Female}\\&\mathsf{mother} \sqsubseteq \mathsf{parent}\\&\mathsf{parent} \sqsubseteq \mathsf{ancestor}\end{aligned}$$

This defines the concept, Man to be equivalent to male humans. Woman is defined analogously. The mother role, is subrole of parent, which in turn is a subrole of ancestor. Description logic concepts and roles would be represented as unary and binary predicates, respectively, in predicate logic.

Assuming statements in arguments are represented using first-order logic, as is the case in Carneades, a scheme for argument from ontologies within the DLP subet of description logic can be implemented using the DLP translation between description logic and the Horn clause subset of first-order predicate calculus.

To implement this argumentation scheme, satisfying the protocol, we need a function, let us call it argument-from-ontology, of type ontology ⇒ (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ stream[argument]. Suppose we want to find arguments about whether or not one of Max's ancestors is Gertrude. Let p be a predicate calculus formula representing this goal statement, ancestor(Max,Gertrude). Let family-relations be the ontology defined above. Let family-relations-scheme be argument-from-ontology(family-relations). Let G be a set of arguments and A an empty set of assumptions. Then the value of family-relations-scheme(G,A,p) is the following finite stream of arguments:

stream(ancestor(Max,Gertrude) since [parent(Max,Gertrude)])

The protocol does not *require* argumentation schemes to return only arguments which would make the statement at issue acceptable or unacceptable, when added to the prior arguments, G in the example above. If acceptable(G,A,parent(Max,Gertrude)) is not true, the heuristic search procedure can apply other argumentation schemes to try to find further arguments for parent(Max,Gertrude), in a backwards-chaining, goal-directed way, which together with the previous arguments do make the goal statement acceptable.

We do not have space to show how to implement this argumentation scheme in detail, but it is not difficult, at least not for the DLP subset of description logic. Some systematic way for mapping predicate symbols to the names of concepts and properties in ontologies is required. OWL provides a way to achieve this, using Universal Resource Identifiers (URIs). This is an example of the kind of integration problem for which OWL was developed to solve.

One issue is whether or not arguments from ontologies should be defeasible, since ontologies are typically defined using some subset of first-order logic, which is of course monotonic. One might claim that all communication presumes a shared ontology which is not subject to debate. Our view is that arguments from ontology are defeasible, in the same way that arguments from theory are defeasible. Even if one accepts that a community in principle shares some ontology, this does not imply that a model of this ontology in some representation language, such as OWL, is adequate or beyond dispute. And even if there has been an explicit agreement within a community to accept an ontology as a standard, or some institional authority has declared the ontology to be binding, arguments from such agreements and authority are also defeasible and subject to critical questions.

The second author has defined a scheme for argument from verbal classification [37, p. 128–132] which can be viewed as a kind of argument from ontology.

Argument from Verbal Classification

Individual Premise. a has property f.

Classification Premise. For all x, if x has property f, then x can be classified as having property g.

Conclusion. a has property g.

Arguments from verbal classification are defeasible. Here is one of the scheme's critical questions: "Is the verbal classification in the classification premise based merely on a stipulative or biased definition that is subject to doubt?". One can imagine other critical questions. Our aim here is only to provide evidence for the claim that arguments from ontology are defeasible, not to explicate these critical questions.

Critical questions can be included in the arguments returned by an argumentation scheme using exceptions and assumptions. For example, the bias critical question above should be modeled as an exception if the burden of producing arguments pro bias should be on the party challenging the argument from ontology, rather than requiring the party who made the argument from ontology to produce arguments showing the lack of bias on the part of the developers of the ontology. Whether a critical question should be an exception or assumption is a policy issue that needs to be addressed by the developers of the argumentation scheme.

To include the bias critical question in the arguments returned by the scheme, the ontology needs to be reified by assigning it an identifier. Indeed every OWL ontology has a URI which can be used to reference it. Let o1 be the URI of the example above and biased be a unary predicate, possibly defined in some other ontology. Then the bias critical question can be included in the arguments returned in response to the example query above as follows:

```
stream(human(Joe) since [man(Joe), unless biased(o1)],
       human(Joe) since [woman(Joe), unless biased(o1)])
```

As a practical matter, including such critical questions can vastly increase the size of the search space for arguments. Indeed, we conjecture that one reason ordinary premises and critical questions are distinguished in argumentation schemes is as a heuristic for reducing the size of the search space. Thus, the control component of a search engine for arguments should provide some way for users to control which kinds of critical questions are asked, perhaps on an issue-by-issue basis.

One more point: Implementing an argumentation scheme for a more expressive description logic than DLP is surely a difficult task. Description logic theorem provers are complex technology and, to our knowledge, typically do not produce proof trees which could be used to extract arguments. The developers of the DLP subset of description logic claim that it is expressive enough for most purposes. But whether or not one agrees with this assessment, from an argumentation perspective this is just an example of the necessary practical limits of all models for generating arguments. Models are abstractions which leave out information which could be relevant for resolving some issue. A DLP version of a richer ontology is an example of a model which abstracts away some information for practical reasons.

4. ARGUMENT FROM RULES

There are many kinds of rules. The common sense, dictionary meaning of rule is "One of a set of explicit or understood regulations or principles governing conduct within a particular sphere of activity." [1]. In classical logic, rules can be inference rules or material implications. In computer science, rules can be production rules, grammar rules, or rewrite rules. When we use the term 'rule' in this section, unless otherwise stated, we mean rule in the regulatory sense.

Rules express not only regulations about how to act, but also regulate how to argue or reason when planning actions or determining whether or not some action or state complies with the rules. For example, the criminal law rule against murder, defined as the "unlawful killing of a human being with malice aforethought", expresses not only a general policy against such killings, but also a policy to presume that a murder has taken place given proof that a human being was intentionally killed. There are exceptions, such as self defense, but the rules are formulated so as to deter killings by increasing the probability that persons will presume that some contemplated killing would be illegal.

Since argumentation schemes express reasoning norms and conventions of a community, argumentation schemes and rules appear to have much in common. Recall that argumentation schemes can be viewed from two perspectives, as argument *patterns* and as argument *generators*. Rules represent argument patterns in a way which enables them to be used to generate arguments. Other representations of argument patterns may be better suited to the task of reconstructing arguments from natural language texts.

In the field of artificial intelligence and law, there is now much agreement about the structure and properties of rules. [15, 27, 20, 33]:

1. Rules have properties, such as their date of enactment, jurisdiction and authority.

2. When the antecedent of a rule is satisfied by the facts of a case, the conclusion of the rule is only presumably true, not necessarily true.
3. Rules are subject to exceptions.
4. Rules can conflict.
5. Some rule conflicts can be resolved using rules about rule priorities, e.g. *lex superior*, which gives priority to the rule from the higher authority.
6. Exclusionary rules provide one way to undercut other rules.
7. Rules can be invalid or become invalid. Deleting invalid rules is not an option when it is necessary to reason retroactively with rules which were valid at various times over a course of events.
8. Rules do not counterpose. If some conclusion of a rule is not true, the rule does not sanction any inferences about the truth of its premises.

One consequence of these properites is that rules cannot be modeled adequately as material implications in predicate logic. Rules need to be *reified* as terms, not formulas, so as to allow their properties, e.g. date of enactment, to be expressed and reasoned about for determing their validity and priority.

Rules can be modelled as tuples of the type (name: symbol, premises: list[statement], exceptions: list[statement], assumptions: list[statement], conclusions: list[statement]), denoted r: c1, . . . , cn ⇐ p1, . . . , pn., where r is the name of the rule, p1, . . . , p2 are the premises, exceptions and assumptions of the rule, in any order, and c1, . . . , cn are the conclusions of the rule. Exceptions and assumptions in rules are denoted unless p and assuming p, respectively, to distinguish them from ordinary premises.

In the Pleadings Game [15], the first author presented one of the first computational models of a scheme for argument from rules. At about the same time, similar work was published by Hage and Verheij [20, 33], and Prakken and Sartor [27]. From this work the following scheme for arguments from rules can be distilled.

Argument from Rules

Let (r1,P,E,A,C) be a rule.

Rule Premises. Let p1, . . . , pn be the premises in P.

1. p1 is true.
2. . . .
3. pn is true.

Rule Exceptions. Is some e in E true?

Rule Assumptions. Is every a in A true?

Validity Assumption. Is valid(r1) true?

Exclusionary Exception. Is excluded(r1,c) true, for the conclusion c in C at issue?

Priority Exception. For the conclusion c at issue, is there a rule r2 such that priority(r2,r1,c)?

Conclusions. Let c1, . . . , cn be the statements in C.

1. applicable(r1)
2. c1 is true.
3. . . .
4. cn is true.

The priority exception represents the critical question for asking whether there is some other applicable rule r2 of higher priority that can be used to reach a conclusion c2 which is complementary, and thus contradictory, to the conclusion c of r at issue. Let a1 be the argument c1 since P1 and a2 the argument c2 since P2. If c1 and c2 are contradictory then, in Pollock's [25] terms, a1 and a2 *rebut* each other. Some way is needed to resolve conflicts among rebuttals. Carneades uses proof standards for this purpose [17]. The priority exception provides an alternative, more specific way to resolve conflicts among arguments from rules, by *undercutting* arguments from lower priority rules. This approach is one way to enable issues about rule priorities to be raised and resolved via argumentation, in a uniform way, just like other issues.

To illustrate, here is a small rulebase about German family law:

§1589: direct-lineage(x,y) ⇐ ancestor(x,y).
§1601: obligated-to-support(x,y) ⇐ direct-lineage(x,y).
§1602: not obligated-to-support(x,y) ⇐ not needy(x).
§1611: excluded(§1601, obligated-to-support(y,x)) ⇐
                    neediness-caused-by-own-immoral-behavior(x).

This models the following rules. §1589 states that ancestors are persons in direct lineage. §1601 states the general rule that persons in direct lineage are obligated to support each other. §1602 states an exception: there is no obligation to support a person who is not needy. §1611 excludes from §1601 needy relatives whose neediness was caused by their own immoral behavior.

To implement the scheme for argument from rules, satisfying the protocol, we need a function, let us call it argument-from-rules, of type list[rule] ⇒ (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ stream[argument]. Let family-support-law be the list of rules defined above. Let family-support-scheme be argument-from-rules(family-support-law). Let G be a set of arguments and A an empty set of assumptions.

Suppose we want to use this scheme to find arguments for Max being obligated to support Gertrude. Let p be the goal statement obligated-to-support(Max,Gertrude). Then family-support-scheme(G,A,p) generates the following argument:

```
stream(obligated-to-support(Max,Gertrude) since
          [direct-lineage(Max,Gertrude),
           assuming valid(§1601),
           unless excluded(§1601,
                       obligated-to-support(Max,Gertrude)),
           unless priority(§1602,§1601,
                       obligated-to-support(Max,Gertrude)])
```

The priority exception in this example is more specific than necessary. The argument can be undercut by any rule having priority over §1601, not just §1602. But stating this exception more generally would require us to violate the simplifying assumption restricting statements to ground formulas. The Carneades implementation can handle variables and the simplification was made only for expository purposes.

5. ARGUMENT FROM CASES

There are various forms of case-based reasoning. The simplest forms are variations of the scheme for argument from analogy, which use some similarity measure to compare cases. More complex schemes compare theories constructed from a set of cases, and order competing theories by their coherence. In this section, we present Wyner and Bench-Capon's reconstruction of the CATO [2] model of analogical case-based reasoning as a set of argumentation schemes [38]. CATO, in turn, is a refinement of Ashley's work on HYPO [3].

A basic scheme for argument from analogy [37, p. 96] is:

Argument From Analogy

Similarity Premise. Case c1 is similar to case c2.

Base Premise. Proposition p is true (false) in case c1.

Conclusion. p is true (false) in c2.

The challenge when modeling reasoning by analogy is to operationalize the concept of similarity. In CATO, a case-base is about a particular *issue*, such as, in a case-base about family law, whether providing support to a family member would cause undue hardship. A case is modeled as a set of propositional *factors*, arranged in a *factor hierarchy*. Each factor favors one side of the issue. Factors in favor of the proposition at issue are called "plaintiff factors"; factors against the proposition at issue are called "defendant factors". In our family law example, a short expected duration of support is a defendant factor, while irreparable harm to a person's relationship with his immediate family is a plaintiff factor. Two cases are considered similar if they have factors in common. If two conflicting precedents are similar to the current case, the argument from the more 'on-point' case is stronger. Let cc be the current case. Define more-on-point to be a function of type (pc1: case, pc2: case) $\Rightarrow$ Boolean where more-on-point(pc1,pc2) is true if factors(pc1) $\cap$ factors(pc2) $\supset$ factors(pc2) $\cap$ factors(cc).

Arguments are constructed by comparing the set of factors of the current case with the factors of precedent cases. Each precedent case is modeled as a set of factors together with the decision of the case regarding the issue of the case-base, undue hardship in our example. Let factors, pfactors and dfactors be functions of type case $\Rightarrow$ set[factor] for selecting all factors, the plaintiff factors and the defendent factors, respectively, of a case. Let decision be a function of type case $\Rightarrow$ {plaintiff, defendant} such that decision(c) equals the party in whose favor the issue was decided. Let other-party be a function of type party $\Rightarrow$party such that other-party(defendant) = plaintiff and other-party(plaintiff) = defendant.

Wyner and Bench-Capon defined seven *partitions* of the set of factors of a precedent case compared to the current case. Each partition is a function of type case $\Rightarrow$ set[factor]. Let pc be a precedent case. For example, partition1(pc) is the intersection of the plaintiff factors in pc and the current case. Similarly, partition2(pc) is the intersection of the defendant factors of pc and the current case.

Wyner and Bench-Capon defined six case-based argumentation schemes using these partitions. The three example schemes below are based on Wyner and Bench-Capon's, but reflect more closely their implementation in Carneades.

AS1. Factor Comparison Scheme

Let cc be the current case and q be the proposition at issue in the casebase.

Premise. The factors of the current case favor party p, denoted factors-favor(p).

Conclusion. q, if p = plaintiff, otherwise complement(q).

AS2. Preference from Precedent Scheme

Let pc1 be some precedent case and p be a party.

Outcome Premise. decision(pc1) = p.

Counterexample Exception. There exists a precedent case, pc2, such that is-counterexample(pc2,pc1).

Conclusion. factors-favor(p)

Counterexample Scheme

Let pc1 and pc2 be precedent cases and p be a party.

Premise. decision(pc1) = p

Premise. decision(pc2) = other-party(p)

Premise. more-on-point(pc2,pc1).

Conclusion. is-counterexample(pc2,pc1)

To illustrate these schemes, let us define a simple case base about undue-hardship. There are five factors, three for the plaintiff and two for the defendant:

Plaintiff Factors

1. has-already-provided-much-support
2. never-had-parent-child-relationship
3. would-cause-irreparable-harm-to-family

Defendant Factors

1. expected-duration-of-support-is-short
2. has-not-provided-care

The casebase, cb1, consists of only three precedent cases, Müller, Bauer and Schmidt:

Müller. Decided for plaintiff. Factors: {never-had-a-parent-child-relationship}.

Schmidt. Decided for defendant. Factors: {never-had-a-parent-child-relationship, expected-duration-is-short}.

Bauer. Decided for plaintiff. Factors: {never-had-a-parent-child-relationship, expected-durations-is-short, would-cause-irreparable-harm-to-family}

Let argument-from-cases be a function of type list[case] ⇒ (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ stream[argument], matching the protocol for argumentation schemes. Let argument-from-cb1 be argument-from-cases(cb1).

Let A be a set of factors assumed to be true in the current case: { not has-already-provided-much-support, expected-duration-of-support-is-short, never-had-parent-child-relationship, would-cause-irrepairable-harm-to-family, not has-not-provided-care}.

Let G be a set of arguments. The factors of the current case, cc, used for comparison with precedent cases, are the propositions which are acceptable in the set of arguments G, given the assumptions, i.e. factors(cc) = {p | acceptable(G,A,p) }.

We can use the argument-from-cb1 instantiation of the argument-from-cases scheme to construct arguments for undue-hardship in the current case, with argument-from-cb1(G,factors(cc),undue-hardship), which equals:

```
stream(undue-hardship since
        [factors-favor(plaintiff,undue-hardship)])
```

The argument returned was constructed using the AS1 (Factor Comparison) argumentation scheme. We can use the arguments-from-cb1 argumentation scheme again, backward chaining, to construct the following argument from the premise of this argument:

```
argument-from-cb1(G,
             factors(cc),
             factors-favor(plaintiff,undue-hardship)) =

stream(factors-favor(plaintiff,undue-hardship) since
      [never-had-parent-child-relationship,
       unless is-counterexample(Schmidt,Müller)])
```

This argument, while correct, is more concrete than we would like, since the exception asks only if the Schmidt precedent is a counterexample to the cited case, Müller, rather than asking whether any case in the casebase is a counterexample. Once again, this is due to the simplifying assumption restricting statements to ground formulas. In the argument returned by Carneades implementation, Schmidt would be replaced by a variable.

In fact, Schmidt is a counterexample to Müller. The counterexample scheme could be used to construct an argument for the exception, undercutting the argument above. But since Bauer is even more on-point than Schmidt, the counterexample scheme could be used again to undercut the argument from Schimdt.

6. ARGUMENT FROM TESTIMONIAL EVIDENCE

In court proceedings, a basic source of evidence about the facts of the case is witness testimony. Similarly, in administrative procedures of public agencies, such as procedures for determing tax obligations or rights to social benefits, citizens provide information about the facts, typically by completing forms. In both cases, the conclusions one may draw are only presumptively true. Witnesses do not always tell the truth or can be mistaken. Tax declarations are audited for good reasons. Thus the conventions of an agency, court or other organziation for drawing inferences from claims and testimony can be viewed as argumentation schemes.

Argument from testimonial evidence is a specialization of the following scheme for argument from position to know [36, p. 46]:

Argument from Position to Know

Major Premise. Source a is in a position to know about things in a certain subject domain s containing proposition p.

Minor Premise. a asserts that p is true (false).

Trustworthiness Exception. a is not trustworthy, reliable or honest.

Conclusion. p is true (false).

One way to implement a computational model of witness testimony is to use a database to store answers to questions. In Carneades, conceptually we use a database schema with the following four tables:

1. A *witness* table storing information about persons, such as their name and contact information.

2. A *question* table stores the information needed for asking questions of the form: Is it the case that predicate(subject,object)? For example: Is it that case that Gertrude is the mother of John, mother(John,Gertrude)? Readers familiar with the Resource Description Framework (RDF) will recognize such statements as *triples* [22]. Triples can represent both binary and unary relations and thus are sufficient for representing all description logic assertions. Unary relations can be modeled as in this example: isa(Joe,Person). The question table records the type of the object of each predicate (e.g. symbol, number, string, Boolean) and the cardinality of the predicate (one or many), along with a text to be used as a template for asking questions in natural language.

3. An *answer* table stores the answers to questions. For predicates with a cardinality greater than 1, it is also noted whether all values of the object of the predicate have been provided by the witness, or only some. If a witness asserts there are no further values, then this can be used by an argumentation scheme to construct arguments against claims of other values, as will be discussed in more detail below.

4. Finally, a *form* table stores a set of forms, where each form is a sequence of questions. This enables dialogues to be structured more coherently, by asking related questions at the same time. For example, when asking for a person's name, one could also ask for essential contact information.

This database is used to construct arguments for propositions at issue by first matching the proposition at issue to the questions in the question table. If a question can be found, we then check whether the question for this issue has already been asked. The question has been asked if there is an entry in the answer table for this question. If the witness was not able to provide any answers, the set of values will be empty. If the question has not been asked, or the witness when first answering the question indicated he knew further answers, the question is asked and the answers are both stored in the answer table and used to construct the arguments returned by the argumentation scheme. If the question has been previously asked and the witness had indicated that he had provided all the answers he was able to provide, arguments are constructed from these answers and returned, without modifying the answer table. While this is admittedly a very operational and procedural description of the process of constructing arguments using this scheme, rather than a declarative definition of a mathematical function, it should be remembered that argumentation schemes, in their role as argument generators, are *methods* for constructing arguments. While some of these methods can be defined functionally others are more naturally defined in procedural terms.

If the proposition at issue is p(s,o), the witness has testified that he had provided all the values of the p property of s, and o is not one of those values, then an argument is constructed *con* the proposition p(s,o) from this testimony. Such a con argument is reminiscent of *predicate completion*, which plays a role in the semantics of *negation as failure* (NAF) in logic programming [9], but is different in a few ways. First, it is restricted in scope to triples with particular predicates and subjects, whereas predicate completion as it is typically used in logic programming applies to all predicates. Second, such arguments are supported by the testimony of a witness who expressly stated that no further values exist, whereas predicate completion is based on the closed-world assumption, that all relevant facts are known and in the database. In our approach, the closed-world assumption is not made. Finally, con arguments constructed in this way can be rebutted or undercut by other arguments, also by testimony of other witnesses. Predicate completion has nothing comparable.

This model of a scheme for argument from witness testimony provides functionality similar to the way rule-based systems ask users for information when there are no rules for deriving some needed fact. But our model is more general as it can handle possibly conflicting testimony about the same issue from more than one witness.

Let us illustrate this data model with a few questions from the German family law example:

predicate	*type*	*cardinality*	*template*
mother	symbol	one	Who is _'s mother?
father	symbol	one	Who is _'s father?
child	symbol	many	Who is a child of _?
needy	boolean	one	Is _ needy?

Let testimony be the type of a database with the above tables and argument-from-testimony be a function of type testimony ⇒ (arguments: set[argument], assumptions: set[statement], issue: statement) ⇒ stream[argument], matching the protocol for argumentation schemes.

To illustrate this model of an scheme for argument from witness testimony, using our German family law domain, let testimony1 be a database of type testimony, with a record of Max's testimony. Suppose Max has yet to be asked any questions. We can construct an argument generator from his testimony as follows. Let argument-from-testimony1 be argument-from-testimony(testimony1). Now, to ask Max whether Gertrude is his mother, we can evaluate argument-from-testimony1(G,A,mother(Max,Gertrude)). Assuming he answers yes, this results in the following argument:

```
stream(mother(Max,Gertrude) since
        [unless not trustworthy(Max)])
```

In this argument, the major and minor premises of the scheme for argument from position to know have been left implicit.[6]

7. RELATED WORK

Our work builds on and was inspired by previous work in AI and Law on using argumentation schemes for legal reasoning [34, 6].

Most prior work on computational models of argumentation schemes has focused on their role as patterns for classifying arguments and revealing implicit premises, to support the process of argument reconstruction. For example, Aracuaria provides a way to define templates for argumentation schemes and to use these templates to classify arguments and their premises [30]. Recently, an OWL ontology of many of the second author's argumentation schemes has been developed, with the aim of being able to use description logic theorem provers to classify arguments [29]. Others have focused on the problem of how to model in a computational argumentation framework the critical questions of argumentation schemes and investigated how such critical questions affect the burden of proof when evaluating the acceptability of statements given a set of arguments [34, 17].

In artificial intelligence, the *blackboard architecture* for hybrid knowledge-based systems, as first implemented in the Hearsay-II speech understanding system [13], provides a way for multiple inference engines to work together on solving a problem. Each inference engine uses its own *knowledge source*, modeled in whatever way is appropriate for its particular reasoning methods. In blackboard systems, the inference engines collaborate by writing statements to a shared data structure, called the 'blackboard'. In Hearsay-II, the statements represent hypotheses about the utterances being interpreted. Inference engines use the statements on the blackboard as input to their reasoning methods, in a forward-chaining, data-driven way. Whenever sufficient data is on the blackboard for some reasoning method of an inference engine to be applicable, the inference engine announces this to a scheduler. If several inference engines have applicable methods, the scheduler decides in which order to apply the methods. The inference engines can derive conflicting conclusions. Hearsay-II provides some way to weigh or order inference engines to resolve these conflicts.

Later blackboard systems, such as Walker's Expander legal expert system [35], recorded not only statements on the blackboard, but also *justifications* for these statements, what we would now call arguments, using a reason-maintenance system [11, 10] to manage dependencies between statements. As the inference engines continue to work on problems and post further information to the blackboard, the reason maintenance system would update the status of the statements on the blackboard, labeling them 'in' or 'out'.

[6]In Carneades, all arguments are annotated with an identifier of the scheme used to construct the argument, so it is not necessary to use pattern matching to try to identify the scheme used, unlike when using schemes to reconstruct arguments in natural language texts.

Clearly there are similarities between our approach to hybrid reasoning using argumentation schemes and blackboard systems. The role of inference engines is played by argument generators. And the acceptable function, of type (arguments: set[argument], assumptions: set[proposition], issue: proposition) ⇒ Boolean can be viewed as providing reason-maintenance services. There are however significant differences. Firstly, argument generators are not problem solvers. They do not implement problem-solving methods with their own control strategies. Rather, a set of argument generators induces a space of sets of arguments which can be searched using a centralized search strategy. Secondly, whereas blackboard systems forward chain from the statements on the blackboard, our approach allows the space of arguments to be searched in a goal-directed way.

As Walker notes, any architecture for integrating hybrid reasoners requires a formal language for expressing statements which is "powerful enough to express the input to and output from any of the knowledge sources" [35, p. 76]. Our approach places few restrictions on the language used for expressing statements in argumentation schemes, requiring only equality and complement operators. The formalism currently used in Carneades allows some meta-level statements to be expressed. For example, it is possible to state that some rule is not applicable to some other statement. This formalism may need to be extended as further argument generators are added to the system.

The problem of translating between the languages used by different 'problem-solving methods' is the focus of a recent article by Henry Prakken [26], in which he develops a model of 'I/O transformers' between problem-solvers, and illustrates this method with transformers for first-order logic, Bayesian probability theory and two versions of default logic. We speculate that such I/O transformers can be reconstructed as argument generators in our framework, but this needs to be validated in future work.

Some research in the artificial intelligence and law field has addressed ways of integrating reasoning with rules and cases [14, 8, 32, 28] and ways to resolves conflicts among arguments, such as prefering arguments from cases to arguments from rules [14, 8, 32]. Our work aims to generalize these results by providing an open, extensible architecture for integrating models of any argumentation scheme.

8. CONCLUSION

Two functions of argumentation schemes can be distinguished, as argument patterns useful for reconstructing arguments from natural language texts, and as methods for generating arguments from argument sources, such as legislation or precedent cases. In many fields, such as the law, solving problems requires several forms of reasoning to be integrated. Our thesis is that argumentation schemes, in their capacity as argument generators, together with an argumentation framework such as Carneades, can provide the foundation for an open architecture for integrating multiple forms of reasoning. We have tested this thesis with models of several argumentation schemes, for argument from ontologies, rules, cases and testimonial evidence, together with an example from German family law, showing how these schemes can be used together to argue about the issues of a case.

In this architecture, there is a division of responsibility between the schemes and the argumentation framework. The schemes define a search space of argument sets or graphs. The argumentation framework is used to evaluate the acceptability of arguments or statements in each state of the search space. A party can use a system which implements this architecture as a tool for constructing arguments in support of or opposing some position by heuristically searching the space for a set of arguments in which the position is acceptable or not acceptable, respectively. After the arguments found are put forward in the dialogue, the opposing party can use the same or another implementation of the architecture, perhaps with other argumentation schemes, to search for counterarguments.

All the argumentation schemes presented have been implemented in Carneades, as part of the European ESTRELLA project, and used to build a number of demonstrators in the legal domain. Carneades is freely available on the Web, as Open Source software.

The demonstrators of the ESTRELLA project are prototypes of expert systems for helping citizens to apply legislation in order to assess their legal rights and obligations. Most deployed legal expert systems are currently built using rule-based systems technology from the 1980s. While such systems have proven their usefulness for supporting the processing of complex claims in public administration as well as the private sector, for example in the insurance industry, they are based on the simplifying assumption that the relevant laws and regulations can be adequately modeled as a logical theory. Claims assessment is viewed as deduction, in which a theory is applied to the facts of the case to deduce legal consequences. Lawyers have long understood that in general legal reasoning cannot be reduced to deduction in this way. Rather, legal reasoning generally involves the iterative construction and comparison of alternative theories of the facts and the law, interpreting both the evidence and the relevant legal sources, in an argumentative process. Our aim in modeling argumentation schemes is to develop tools which can help people to construct a wide variety of arguments, improving their ability to protect their interests in dialogues, especially in the legal domain.

Acknowledgments

The work reported here was conducted as part of the European ESTRELLA project (IST-4-027655). We would like to thank Alexander Boer, Trevor Bench-Capon, Joost Breuker, Tom van Engers, Rinke Hoekstra, Henry Prakken and Adam Wyner for fruitful discussions about topics related to this paper. An earlier version of this article was presented at a workshop on Computional Models of Natural Argument [16].

9. REFERENCES

[1] F. Abate and E. J. Jewell, editors. *New Oxford American Dictionary*. Oxford University Press, 2001.

[2] V. Aleven. *Teaching Case-Based Argumentation Through a Model and Examples*. Ph.d., University of Pittsburgh, 1997.

[3] K. D. Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals*. Artificial Intelligence and Legal Reasoning Series. MIT Press, Bradford Books, 1990.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic*

Handbook – Theory, Implementation and Applications. Cambridge University Press, 2003.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[6] F. Bex, H. Prakken, C. Reed, and D. Walton. Towards a formal account of reasoning with evidence: Argumentation schemes and generalizations. *Artificial Intelligence and Law*, 11(2-3):125–165, 2003.

[7] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1-2):63–101, 1997.

[8] L. K. Branting. *Reasoning with Rules and Precedents: A Computational Model of Legal Analysis.* Kluwer Academic Publishers, Dordrecht, 2000. Book version of 1991 PhD Thesis.

[9] K. Clark. Negation as failure. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 311–325. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[10] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[11] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[12] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[13] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Comput. Surv.*, 12(2):213–253, 1980.

[14] A. Gardner. *An Artificial Intelligence Approach to Legal Reasoning.* MIT Press, 1987.

[15] T. F. Gordon. *The Pleadings Game; An Artificial Intelligence Model of Procedural Justice.* Springer, New York, 1995. Book version of 1993 Ph.D. Thesis; University of Darmstadt.

[16] T. F. Gordon. Hybrid reasoning with argumentation schemes. In *Proceedings of the 8th Workshop on Computational Models of Natural Argument (CMNA 08)*, pages 16–25, Patras, Greece, July 2008. The 18th European Conference on Artificial Intelligence (ECAI 2008).

[17] T. F. Gordon, H. Prakken, and D. Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-11):875–896, 2007.

[18] G. Governatori, M. Maher, G. Antoniou, and D. Billington. Argumentation semantics for defeasible logic. *Journal of Logic and Computation*, 14:675–702, 2004.

[19] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings of the Twelth International World Wide Web Conference (WWW 2003)*, pages 48–57, Budapest, Hungary, May 2003. ACM.

[20] J. C. Hage. *Reasoning with Rules – An Essay on Legal Reasoning and its Underlying Logic.* Kluwer Academic Publishers, Dordrecht, 1997.

[21] J. D. Mackenzie. Question-begging in non-cumulative systems. *Journal of Philosophical Logic*, 8:117–133, 1979.

[22] F. Manola and E. Miller. RDF primer, 2004.

[23] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language overview. http://www.w3.org/TR/owl-features/, 2004.

[24] D. Nute. Defeasible logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 253–395. Clarendon Press, Oxford, 1994.

[25] J. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

[26] H. Prakken. Combining modes of reasoning: An application of abstract argumentation. In *Proceedings of The 11th European Conference on Logics in Artificial Intelligence (JELIA 2008)*, volume 5293 of *Springer Lecture Notes in AI*, pages 349–361, Berlin, April 2008. Springer Verlag.

[27] H. Prakken and G. Sartor. A dialectical model of assessing conflicting argument in legal reasoning. *Artificial Intelligence and Law*, 4(3-4):331–368, 1996.

[28] H. Prakken and G. Sartor. Modelling reasoning with precedents in a formal dialogue game. *Artificial Intelligence and Law*, 6(2-4):231–287, 1998.

[29] I. Rahwan and B. Banihashemi. Arguments in OWL: A progress report. In A. Hunter, editor, *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA)*, Amsterdam, The Netherlands, 2008. IOS Press.

[30] C. A. Reed and G. W. Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal of AI Tools*, 13(4):961–980, 2004.

[31] G. R. Simari and R. P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.

[32] D. B. Skalak and E. L. Rissland. Arguments and cases: An inevitable intertwining. *Aritificial Intelligence and Law*, 1(1):3–45, 1992.

[33] B. Verheij. *Rules, Reasons, Arguments. Formal Studies of Argumentation and Defeat.* Ph.d., Universiteit Maastricht, 1996.

[34] B. Verheij. Dialectical argumentation with argumentation schemes: An approach to legal logic. *Artificial Intelligence and Law*, 11(2-3):167–195, 2003.

[35] R. Walker. *An Expert System Architecture for Hetergeneous Domains – A Case Study in the Legal Field.* PhD thesis, Vrije Universiteit Amsterdam, 1992.

[36] D. Walton. *Legal argumentation and evidence.* Pennsylvania State University Press, University Park, PA, 2002.

[37] D. Walton. *Fundamentals of Critical Argumentation.* Cambridge University Press, Cambridge, UK, 2006.

[38] A. Wyner and T. Bench-Capon. Argument schemes for legal case-based reasoning. In *JURIX 2007: The Twentieth Annual Conference on Legal Knowledge and Information Systems*, 2007.