Constructing Arguments with a Computational Model of an Argumentation Scheme for Legal Rules

Interpreting Legal Rules as Reasoning Policies

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

ABSTRACT

A knowledge representation language for defeasible legal rules is defined, whose semantics is purely procedural, based on Walton's theory of argumentation and Loui's break with the relational tradition in 'Process and Policy'. Legal rules are interpreted as reasoning policies, by mapping them in the semantics to argumentation schemes. The reasoning process is regulated by argumentation protocols. Reasoning with legal rules is viewed as applying schemes for arguments from rules to construct arguments to be put forward in dialogues.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence—*Knowledge Representation Formalisms and Methods*; J.5 [**Computer Applications**]: Administrative Data Processing

Keywords

Legal Knowledge-Based Systems, Computational Models of Legal Reasoning and Argumentation

1. INTRODUCTION

Legal rules express public policy. These are not only policies about how to act, but also policies about how to reason when planning actions or determining the legal consequences of actions after they have been performed. For example, the definition of murder as the "unlawful killing of a human being with malice aforethought" expresses both the policy against the intentional killing of another human being and the reasoning policy to presume that a murder has been committed if it has been proven that a human being has been killed intentionally. Such presumptions are not sufficient for making legally correct decisions. Having proven that the defendant has killed someone intentionally is not sufficient for proving guilt. Rather, a guilty verdict would be legally correct only at the end of a proper and fair legal procedure, i.e. via "due process", in which the parties are given a fair opportunity to produce evidence and make arguments and these are properly taken into consideration in the justification of the decision. If during this procedure the defendant is able to produce evidence of the killing having been done in self-defense, for example, a guilty verdict would be correct only if the prosecution succeeds in addition to meet its burden of persuading the court that the killing was in fact not done in self-defense.

Thus, the semantics of the knowledge-representation formalism for legal rules presented here is based on the dialectical and argumentation-theoretic approach articulated by Ron Loui in his landmark article "Process and Policy: Resource-Bounded Non-Demonstrative Reasoning" [4]. Essentially, legal rules are interpreted as policies for reasoning in resource-limited, decision-making processes. In argumentation theory such reasoning policies are viewed as inference rules for presumptive reasoning, called *argumentation schemes* [9]. Arguments are instances of argumentation schemes, constructed by substituting variables of a scheme with terms of the object language. An *argument graph* is a proof constructed from a set of arguments. A set of argumentation schemes defines a search space over argument graphs. Reasoning with argumentation schemes can be viewed as heuristic search in this space, looking for argument graphs in which some disputed claim is acceptable or not given the proof represented by the argument graph. In dialogues, the parties take turns searching this space, looking for counterarguments. Turn-taking, termination conditions, resource limitations and other procedural parameters are determined by the rules of the legal proceeding, i.e. by the argumentation protocol for the particular type of proceeding.

The rule language developed here is much like the one the author developed for the Pleadings Game [1]: rules are reified and subject to exceptions; conflicts between rules can be resolved using other rules about rule priorities; the applicability of rules can be reasoned about and excluded by other rules; and the validity of rules can be questioned. The rule language of the Pleadings Game is similar in some ways to Reason-Based Logic [3], developed independently at about the same time. The Pleadings Game is cited in Prakken and Sartor's 1996 article [6], in which they introduce their language for defeasible rules, now known as PRATOR, but it is not clear to what extent the rule language of the Pleadings Game influenced the design of PRATOR. All three of these systems viewed reasoning with legal rules as argumentation,

ICAIL 2007 Palo Alto, California,USA

but none of them interpreted legal rules as argumentation schemes. Rather, in all of these systems legal rules were either represented as sentences in a nonmonotonic logic or, as in the Pleadings Game, compiled to a set of such sentences.[1] Verheij was the first to explicitly discuss the modeling of legal rules using argumentation schemes [8] but like the Pleadings Game interprets rules as abstractions of sets of formulas in a nonmonotonic logic, rather than interpreting rules as abstractions of arguments, i.e as argumentation schemes in Walton's sense [9]. With the exception of the Pleadings Game, these prior systems model argumentation as deduction, i.e. as a defeasible consequence *relation*. In the Pleadings Game, argumentation was viewed procedurally, as dialogues regulated by protocols, but this was accomplished by building a procedural layer on top of a nonmonotonic logic. The system presented here abandons the relational interpretation of argumentation entirely, in favor of a purely procedural view, and is thus more in line with modern argumentation theory in philosophy. We are not suggesting that logic no longer has an important role to play, but only that argumentation cannot be reduced to logic.

The rest of this paper is organized as follows. First, we provide an informal introduction and overview of a language for legal rules, including some examples. This is followed by the formal definition of the syntax of this rule language. Then we define the semantics for the rule language, by mapping rules to argumentation schemes, using our Carneades model of argument [2]. The final section recapitulates the main results and discusses future work.

2. INFORMAL OVERVIEW

For simplicity and readability, we will be using a concrete syntax based on Lisp *s-expressions* to represent rules. Variables will be represented as symbols beginning with a question mark, e.g. `?x` or `?y`. Other symbols, as well as numbers and strings, represent constants, e.g. `contract`, `23.1`, or `"Jane Doe"`.

An *atomic sentence* is a simple declarative sentence, containing no logical operators, such as negation, conjunction or disjunction. The sentence "The mother of Caroline is Ines." can be represented as `(mother Caroline Ines)`, which has the form `(<predicate> <subject> <object>)`.

If P is an atomic sentence, then `(not P)` is a *negated atomic sentence*. Sentences which are either atomic sentences or negated atomic sentences are called *literals*. The *complement* of a literal `Q` is `P`, if `Q` equals `(not P)`, or `(not P)`, if `Q` equals `P`.

Rules are *reified* in this language, with an identifier and a set of properties, enabling any kind of meta-data about rules to be represented, such as a rule's date of enactment, issuing governmental authority, legal source text, or its period of validity. We do not define these properties here. Our focus is on defining the syntax and semantics of these rules.

Rules have a *body* and a *head*. The terms 'head' and 'body' are from logic programming, where they mean the conclusions and antecedents of a rule, respectively, interpreted as Horn clauses. Unlike Horn clause logic, a rule in our system may have more than one conclusion, including, as will be explained shortly, negated conclusions.

Here is first example, a simplified reconstruction of a rule from the Article Nine World of the Pleadings Game [1], meaning that all movable things except money are goods.

```
(rule §-9-105-h
    (if (and (movable ?c)
             (unless (money ?c)))
        (goods ?c)))
```

'`§-9-105-h`' is an identifier, naming the rule, which may be used as a term denoting the rule in other rules.

We use the term *condition* to cover both literals and the forms `(unless P)`, called *exceptions*, and `(assuming P)`, called *assumptions*, where `P` is a literal. The head of a rule consists of a list of *literals*. Notice that, unlike Horn clause logic, rules may have negative conclusions. Negated atomic sentences may also be used in the body of a rule, also in exceptions and assumptions. Exceptions and assumptions are allowed only within the body of rules. The example rule above illustrates the use of an exception.

Legal rules are defeasible generalizations. Showing that some exception applies is one way to defeat a rule, by *undercutting* [5] it. A rule applies if its conditions are acceptable, *unless* some exception applies. A party who wants to apply this rule need not show that the exception does not apply. The burden of producing evidence that the exception does apply is on the other party. Assumptions on the other hand, as their name suggests, are assumed to hold until they are questioned.

Another source of defeasibility is conflicting rules. Two rules conflict if one can be used to derive `P` and another `(not P)`. To resolve these conflicts, we need to be able to reason (i.e. argue) about which rule has priority. To support reasoning about rule priorities, the rule language includes a built-in predicate over rules, `prior`, where `(prior r1 r2)` means that rule `r1` has priority over rule `r2`. If two rules conflict, the arguments constructed using these rules are said to *rebut* each other, following Pollock [5].

The priority relationship on rules is not defined by the system. Rather, priority is a substantive issue to be reasoned (argued) about just like any other issue. One way to construct arguments about rule priorities is to apply the argumentation scheme for arguments from legal rules to meta-level rules, i.e. rules about rules, using information about properties of rules, such as their legal authority or date of enactment. The reification of rules and the built-in priority predicate make this possible. In knowledge bases for particular legal domains, represented using this rule language, rules can be prioritized both *extensionally*, by asserting facts about which rules have priority over which other rules, and *intensionally*, using meta-rules about priorities.

For example, assuming metadata about the enactment dates of rules has been modeled, the legal principle that later rules have priority of earlier rules, *lex posterior*, can be represented as:

```
(rule lex-posterior
    (if (and (enacted ?r1 ?d1)
             (enacted ?r2 ?d2)
             (later ?d2 ?d1))
        (prior ?r2 ?r1)))
```

[1]Technically speaking, the rules in PRATOR also may be viewed as domain-dependent inference rules, since they may not be used contrapositively, but nonetheless they are formulated as sentences in the object language.

Rules can be defeated in two other ways: by challenging their validity or by showing that some exclusionary conditions apply. These are modeled with rules about validity and exclusion, using two further built-in predicates: `(valid <rule>)` and `(excluded <rule> <literal>)`, where `<rule>` is a constant naming the rule, not its definition. The second argument of the `excluded` predicate is a compound term representing a literal. Thus, literals can also be reified in this system.

The `valid` and `excluded` relations, like the `prior` relation, are to be defined in domain models. Rules can be used for this purpose. For example, the exception in the previous example about money not being goods, even though money is movable, could have been represented as an exclusionary rule as follows:

```
(rule §-9-105-h-i
    (if (money ?c)
        (excluded §-9-105-h (goods ?c))))
```

To illustrate the use of the validity property of rules, imagine a rule which states that rules which have been repealed are no longer valid:

```
(rule repeal
    (if (repealed ?r1)
        (not (valid ?r1))))
```

Notice the use of negation in the conclusion (head) of this rule.

3. SYNTAX

This section presents a formal definition of an s-expression syntax for rules, in Extended Backus-Naur Form (EBNF)[2]. This syntax is inspired by the Common Logic Interchange Format (CLIF) for first-order predicate logic, which is part of the draft ISO Common Logic standard.[3] While inspired by CLIF, no attempt is made to make this rule language conform to Common Logic standard.[4]

The syntax uses the Unicode character set. White space, delimiters, characters, symbols, quoted strings, Boolean values and numbers are lexical classes, not formally defined here. For simplicity and to facilitate the development of a prototype inference engine using the Scheme programming language, we will use Scheme's lexical structure, as defined in the R5RS standard, extended to support the Unicode character set.[5]

Variable and Constant Symbols

```
variable ::= symbol
constant-symbol ::= symbol
```

Variable and constant symbols are disjunct. A variable begins with a question mark character. Symbols are case-sensitive. Constant symbols may include a *prefix* denoting a *namespace*. Some mechanism for binding prefixes to namespaces is presumed in this report, rather than being defined here. The prefix of a constant symbol is the part of the constant symbol up to the first colon. The part of the constant symbol after the colon is the local identifier, within this namespace.

Here are some example variable and constant symbols:

```
?x
?agreement
contract-1
lkif:permission
event-calculus:event
```

Term

A term is either a constant or a compound term. A constant is either a variable, constant symbol, string, number, or Boolean value. A compound term consists of a constant symbol and a list of terms.

```
constant ::= variable | constant-symbol
           | string | number | Boolean
term ::= constant | | ’´’ term |
       ’(’ constant-symbol term* ’)’
```

Quoted terms are used, as in Lisp, to denote lists. Here are some example terms:

```
?x
contract-1
"Falkensee, Germany"
12.345
#t
(father-of John)
’(red green)
```

Literal

Literals are atomic sentences or negations of atomic sentences.

```
atom ::= constant-symbol
      | ’(’ constant-symbol term* ’)’
literal ::= atom | ’(’ ’not’ atom ’)’
```

Notice that constant symbols can be used as atomic sentences. This provides a convenient syntax for a kind of propositional logic.

The following are examples of literals:

```
liable
(initiates event1 (possesses ?p ?o))
(holds (perfected ?s ?c) ?p)
(children Ines ’(Dustin Caroline))
(not (children Tom ’(Sally Susan)))
(applies UCC-§-306-1 (proceeds ?s ?p))
```

Rule

This rule language generalize the syntax of Horn clause logic in the following ways:

1. Rules are reified with names.
2. Rules may have multiple conclusions.

[2]EBNF is specified in the ISO/IEC 14977 standard.

[3]http://philebus.tamu.edu/cl/

[4]Common Logic is a family of concrete syntaxes for first-order predicate logic, with its model-theoretic semantics and classical, monotonic entailment relation. These semantics are sufficiently different as to not make it useful to attempt to make the syntax of our rule language fully compatible with CLIF.

[5]http://www.schemers.org/Documents/Standards/R5RS/

3. Negated atoms are permitted in both the body and head of rules.

4. Rule bodies may include exceptions and assumptions.

5. Negated atoms are allowed in exceptions and assumptions, e.g. `(unless (not p))` or `(assuming (not p))`.

Here is the formal definition of the syntax of rules:

```
rule ::= '(' 'rule' constant-symbol
             '(' 'if' body head ')' ')'
       | '(' 'rule' constant-symbol
             literal literal* ')'

head :== literal
       | '(' 'and' literal literal+ ')'

body :== condition
       | '(' 'and' condition condition+ ')'

condition :== literal
            | '(' 'unless' literal ')'
            | '(' 'assuming' literal ')'
```

The second rule form is convenient for rules with empty bodies. These should not be confused with Prolog 'facts', since they are also defeasible. Conditions which are not assumptions or exceptions are called *ordinary conditions*.

Here are a few examples of rules and facts, reconstructed from the Article Nine World of the Pleadings Game [1]:

```
(rule §-9-306-1
    (if (and (goods ?s ?c)
             (consideration ?s ?p)
             (collateral ?si ?c)
             (collateral ?si ?p)
             (holds (perfected ?si ?c) ?e)
             (unless (applies §-9-306-3-2
                        (perfected ?si ?p))))
        (holds (perfected ?si ?c) ?e)))

(rule §-9-306-2a
    (if (and (goods ?t ?c)
             (collateral ?s ?c))
        (not (terminates ?t
                (security-interest ?s)))))

(rule F1 (not (terminates T1
                  (security-interest S1))))
(rule F2 (collateral S1 C1))
```

Reserved Symbols

The following predicate symbols have special meaning in the semantics, as explained in Section 4, and are thus reserved: `prior`, `excluded`, `valid`, and `applies`.

4. SEMANTICS

We now proceed to define the semantics of this rule language. Due to space limitations, knowledge of the Carneades model of argument [2] is presumed. A rule denotes a *set* of argumentation schemes, one for each conclusion of the rule, all of which are subclasses of a scheme for arguments from legal rules.[6] Applying a rule is a matter of instantiating one of these argumentation schemes to produce a particular argument. Reasoning with rules is viewed as a process of applying these schemes to produce arguments to put forward in dialogues.

The scheme for arguments from legal rules is based on the rule language we developed for the Pleadings Game [1], but has also been influenced by Verheij's reconstruction of Reason-Based Logic in terms of argumentation schemes [8]. The scheme can be defined informally as follows:

Premises

1. r is a legal rule with ordinary conditions $a_1, \ldots, a_n$ and conclusion c.
2. Each a_i in $a_1 \ldots a_n$ is presumably true.

Conclusion. c is presumably true.

Critical Questions.

1. Does some exception of r apply?
2. Is some assumption of r not met?
3. Is r a valid legal rule?
4. Does some rule excluding r apply in this case?

Our task now is use this scheme to define the semantics of the formal language of Section 3, by mapping rules in the language to schemes for arguments in Carneades. We begin by mapping rule conditions to argument premises.

Definition 1 (Condition to Premise) *Let p be a function mapping conditions of rules to argument premises, defined as follows:*

$$p(c) = \begin{cases} c & \text{if } c \text{ is a literal} \\ \bullet s & \text{if } c \text{ is } \texttt{(assuming } s\texttt{)} \\ \circ s & \text{if } c \text{ is } \texttt{(unless } s\texttt{)} \end{cases}$$

If a conclusion of a rule is an atomic sentence, s, then the rule is mapped to a scheme for arguments *pro* s. If a conclusion of the rule is a negated atomic sentence, `(not` s`)`, then the rule is mapped to a scheme for arguments *con* s.

Definition 2 (Scheme for Arguments from Rules) *Let r be a rule, with conditions $a_1 \ldots a_n$ and conclusions $c_1 \ldots c_n$. Two premises, implicit in each rule, are made explicit here. The first, $\bullet v$, where $v =$* `(valid` r`)`*, makes the assumption that r is a valid legal rule explicit. The second, $\circ e$, where $e =$* `(excluded` r c_i`)`*, expresses the exception that r is* excluded *with respect to c_i.*

For each c_i in $c_1 \ldots c_n$ of r, r denotes an argumentation scheme of the following form, where d is 'pro' if c_i is an atomic sentence and 'con' if c_i is a negated atomic sentence:

$$\frac{p(a_1) \ldots p(a_n), \bullet v, \circ e}{d\ c_i}$$

[6] We do not claim that argumentation schemes can be modeled as or reduced to defeasible generalizations. Here we go in the other direction: each legal rule is interpreted as a defeasible generalization and mapped to a set of argumentation schemes.

To construct an argument from one of these argumentation schemes, the variables in the scheme need to be systematically renamed and then instantiated using a *substitution environment*, i.e. a mapping from variables to terms, constructed by unifying the conclusion of the argumentation scheme with some goal atomic statement, as in logic programming.

The `valid` and `excluded` relations used in the argumentation scheme are to be defined in the models of legal domains, as explained in Section 2.

Conflicts between competing pro and con arguments are resolved using a priority relation over rules. In Carneades, such a relation is assumed as part of the context. Rules can be used to define the priority relation, as in the Pleadings Game [1] and PRATOR [6]. Legal principles for resolving rule conflicts, such as *lex posterior*, can be modeled in this way, as illustrated in Section 2.

The `applies` predicate is a 'built-in', meta-level relation which cannot be defined directly in rules. It is defined as follows:

Definition 3 (Applies) *Let σ be a substitution environment and G be an argument graph. Let r be a rule and S be the set of argumentation schemes for r, with all of the variables in these schemes systematically renamed. There are two cases, for atomic sentences and negated atomic sentences. The rule r* applies *to a sentence in σ and G if there exists a* pro *argumentation scheme s in S, if the sentence is atomic, p, or a* con *argumentation scheme, if the sentence is negated,* `(not p)`*, such that the conclusion of s is unifiable with p in σ, and every premise of s, with its variables substituted by their values in the σ,* holds *in G.*

Given a set of rules and an argument graph, this definition of the `applies` predicate enables some meta-level reasoning. It allows one to identify the rules which can be used to generate defensible pro and con arguments for some goal statement or to check whether a particular rule can be used to generate a defensible pro or con argument for some statement.

The semantics of negation is dialectical, not classical negation or negation-as-failure. Exceptions also do not have the semantics of negation-as-failure. The closed-world assumption is not made. In Carneades, a negated sentence, `(not p)`, is acceptable just when the complement of the proof standard assigned to `p` is satisfied, where the complement of a proof standard is constructed by reversing the roles of pro and con arguments in the standard. See [2] for details.

5. CONCLUSION

The rule language presented here is syntactically similar to the rule languages of the Pleadings Game [1] and the PRATOR system [6]. Our main original contribution is the particular argumentation-theoretic semantics we have given these rules, by mapping them to argumentation schemes using the Carneades model of argument. This approach has at least two advantages:

1. The system can be extended with comparable models of other argumentation schemes, e.g. for reasoning with evidence, ontologies or precedent cases. Argumentation schemes provide a unifying framework for building hybrid reasoners.
2. The semantics is purely procedural, in line with Loui's 'Process and Policy' work, Rawl's concept of procedural justice [7] and the modern procedural perspective of argumentation in philosophy [9]. Despite the expressiveness of the rule language, which would result in an undecidable logic using the relational approach, argumentation protocols can be defined for using these rules in legal proceedings which are guaranteed to terminate with legally correct conclusions.

A prototype inference engine for this rule language has already been fully implemented in Scheme, a dialect of Lisp. Due to lack of space, a description of the implementation as well further examples must more await a more complete future publication. Our work in the near future, together with our colleagues in the European ESTRELLA project, will focus on extending this system with computational models of schemes for arguments from ontologies, case law and evidence, especially testimonial evidence, and validating the prototype in pilot applications, probably in the domain of tax law.

6. ACKNOWLEDGMENTS

The work reported here was conducted as part of the European ESTRELLA project (IST-4-027655). I would like to thank Alexander Boer, Trevor Bench-Capon, Tom van Engers, Jonas Pattberg, Henry Prakken, Doug Walton, and Adam Wyner for fruitful discussions about topics related to this paper. I would also like to thank the anonymous reviewers for inviting me to be succinct.

7. REFERENCES

[1] 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.

[2] T. F. Gordon, H. Prakken, and D. Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 2007. In Press.

[3] J. C. Hage. Monological reason-based logic. a low level integration of rule-based reasoning and case-based reasoning. In *Proceedings of the Fourth International Conference on Artificial Intelligence and Law*, pages 30–39, New York, 1993. ACM.

[4] R. P. Loui. Process and policy: resource-bounded non-demonstrative reasoning. *Computational Intelligence*, 14:1–38, 1998.

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

[6] 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.

[7] J. Rawls. *A Theory of Justice*. Belknap Press of Harvard University Press, 1971.

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

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