Reference:

Verheij, B. (1995). Reason Based Logic in Law. *Verso un sistema esperto giuridico integrale* (eds. C. Ciampi, F. Socci Natali and G. Taddei Elmi), pp. 681-693. CEDAM, Padova. Also published as report SKBS/B3.A/93-18.

Reason Based Logic in Law

H.B. Verheij

University of Limburg
P.O. Box 616
6200 MD Maastricht
+31 43 883048
email: bart.verheij@metajur.rulimburg.nl

ABSTRACT

Because legal rules are defeasible in nature, and can be contradictory, special features are required for the modelling of legal reasoning. The idea proposed in this paper to deal with these aspects is to consider a rule as a reason-generator. The condition of a rule is merely regarded as a possible reason for its conclusion. Other reasons considering that conclusion, originating from other rules, can influence the actual derivation. Knowledge on the relative weights of the relevant reasons will be needed to derive a conclusion.

This idea is supplemented by the notion of rule applicability. In the context of this paper the applicability of a rule means that its condition becomes a reason for its conclusion. In this way reasons can be excluded in the presence of certain knowledge that makes the rule inapplicable.

The resulting formal framework is called Reason Based Logic. Several examples show how the ideas can be applied in the field of law.

1 MODELLING LEGAL REASONING

Research on legal reasoning, and more specifically on legal knowledge based systems, has shown that classical logic is not fully satisfactory as a model of legal reasoning.[1] Regarding the modelling of legal reasoning, we can distinguish the following issues that are characteristic for the domain of law.

Legal rules have in general a defeasible character. Rules can be inapplicable in exceptional situations. These are the so-called 'undercutting' exceptions.[2] In legislation these exceptions are often separated from the rules themselves. Furthermore, the literal application of a rule can be unwanted, because it is against the purpose of the rule, or a legal principle.

Legal rules have often implicit scope restrictions. For example, legal rules are mostly valid only in a specific country.

Legal regulations are sometimes in conflict. An example of this are the 'rebutting' exceptions to a rule, which have an opposite or conflicting conclusion. In the field of law several ways of conflict resolution are used. Apart from explicit priority clauses for pairs of rules, or classes of

---

1 Prakken (1993a) gives an extensive survey of this. Prakken (1993b) and Sartor (1993) also pay attention to the relation of logic and legal reasoning.

2 Prakken (1993a) discusses this type of exception, and more.

rules, there are the generally applicable, implicit principles Lex Superior, Lex Posterior, and Lex Specialis.

Note that all these techniques only resolve conflicts between pairs of rules. In matters of classification however, as for example in case law, there are often more considerations involved, some of which plead for, some against a conclusion. Their relative weight then leads to the final conclusion.

In this paper a formalization is given of a logic that uses reasons and explicit knowledge on their relative weight. It turns out that the resulting logical framework can deal flexibly and intuitively with the requirements for the modelling of legal reasoning listed above. The formalization of these ideas is referred to as Reason Based Logic.[3]

2 REASONING WITH REASONS

The reasoning model presented in this paper defines what conclusions can be drawn given certain knowledge. We distinguish three types of knowledge, namely facts, rules and weighing results. This distinction is not made to stress intrinsic differences, but to allow for a clear presentation and a simple formalization of the main ideas.[4]

In Reason Based Logic the basis for derivations is standard deduction of First Order Predicate Logic. An extra derivation mechanism uses rules and weighing results. In short this type of derivation goes as follows.

To derive a statement first all reasons for and against it are collected. These reasons originate from rules. The following step is to check the relative weight of the reasons and conclude accordingly. If the reasons for a statement outweigh the reasons against it, the statement can be derived. If the balance is the other way around, its negation can be derived. It is also possible that the weighing knowledge is not sufficient to generate a conclusion. Then neither the statement nor its negation can be derived.

In Reason Based Logic a rule is just a reason-generating object. It consists of a condition and a conclusion. Only the condition of an applicable rule actually is a reason for its conclusion. Whether a rule is applicable or not, has to be derived itself. The default reason pleading for the applicability of a rule is the satisfaction of its condition. The denial of its condition is by default a reason against its applicability. In addition other rules can give rise to reasons for or against the applicability of a rule, if they are applicable themselves. Note that the applicability of a rule only means that its condition becomes a reason for its conclusion. It does *not* mean that its conclusion actually follows, as in other formalisms.

The basic weighing knowledge is simple. If all reasons point in one direction, i.e., all the reasons plead for a statement, it can be derived. If all reasons plead against it, its negation can be derived. In all other cases no conclusion follows and nothing can be derived.

Note that as a result the resolution of a conflict between rules can be divided in two parts. First, priority knowledge can simplify a conflict by making rules inapplicable. Second, the conflict can be solved by knowledge on the relative weight of the reasons originating from the remaining applicable rules.

---

3 The main ideas of Reason Based Logic have been explained earlier by Hage (1991, 1993).

4 In fact, Hage (1993) treats the three types on an equal basis.

The term 'rule' as used in this paper can lead to confusion. Because a rule is considered to be only a reason-generator, its conclusion does not even automatically follow, if it is applicable. This can be the case if there are other conflicting reasons involved. Because of this the phrase 'The rule fires' is meaningless. Only the group of rules with a statement or its opposite as conclusion *together* can fire, if the necessary weighing knowledge is available.

It is however the case that under normal circumstances the conclusion of a rule follows, if its condition is satisfied. These normal circumstances are that there are no rules with a contradicting conclusion, and no rules that can generate reasons against the applicability of the rule.

In table 1 an overview of the derivation mechanism, that characterizes Reason Based Logic is given.

The derivation of a statement consists of three parts:
1 Find all reasons for and against the statement.
2 Use weighing knowledge to weigh the reasons found in step 1.
3 Derive the statement or its opposite (or neither one) according to the result at step 2.
Part 1 comes down to:
1.1 Find rules that have a conclusion matching the statement.
1.2 Derive the applicability, or non-applicability, of the rules found in step 1.1 (again applying steps 1, 2 and 3).
1.3 The conditions of applicable rules (found in step 1.2) constitute reasons for or against the statement.

Table 1. An informal overview of reasoning in Reason Based Logic

3 A FORMALIZATION OF REASON BASED LOGIC

3.1 Theories: facts, rules, weighing results

Reason Based Logic (RBL) is an extension of First Order Predicate Logic (FOPL). An RBL-theory is divided in three parts: facts, rules and weighing results.

Definition 1. The language L_{RBL} of Reason Based Logic is the language L_{FOPL} of First Order Predicate Logic, plus the unary predicate symbols applicable, condition_satisfied and condition_denied, and the unary function symbols cs and cd. An *RBL-formula* is a formula of L_{RBL}. An *RBL-proposition* is an RBL-formula without free variables.

Definition 2. An *RBL-rule* has the form $r : \varphi \Rightarrow \lambda$, where r is a function symbol, φ an RBL-formula, and λ an *RBL-literal*, i.e., an RBL-formula that is also a literal. A rule is called *closed*, if neither φ, nor λ contain free variables; otherwise *open*. If $\lambda = \alpha$ for an atom α, we call φ a *possible reason for* α, if $\lambda = \neg\alpha$ a *possible reason against* α. The term $r(x)$, where x stands for the free variables in the rule, is called the *name* of the rule.

The symbol $\Rightarrow$ that is part of a rule should not be confused with the implication in First Order Predicate Logic which is denoted $\rightarrow$. Rules are the reason-constituting objects in Reason Based Logic. If a rule is applicable its condition is a reason for its conclusion. This constitution of a reason is called the application of the rule. Whether the conclusion of a rule follows, is not solely determined by its application. A conclusion follows by knowledge on the relative weight

of the reasons that arise through the application of all relevant and applicable rules (see definition 7 in section 3.2).

Definition 3. A *weighing result* is a tuple (α, P, C, v), where α is an *RBL-atom* (that is an RBL-formula that is also an atom), P and C are finite sets of formulas, and v equals *pos* or *neg*. A weighing result is called *closed*, if neither α, nor any formula in P or C contains free variables; otherwise *open*. Let R be a set of rules. We call a set of weighing results W_R a *weighing relation for* R, if it has the following properties:

(1) For all $(\alpha, P, C, v) \in W_R$: $P \subseteq P_R(\alpha)$ and $C \subseteq C_R(\alpha)$. Here $P_R(\alpha)$ and $C_R(\alpha)$ are the set of possible reasons for α (stemming from rules in R), and the set of possible reasons against α, respectively.

(2) For all non-empty sets $P \subseteq P_R(\alpha)$ and $C \subseteq C_R(\alpha)$, $(\alpha, P, \emptyset, pos) \in W_R$ and $(\alpha, \emptyset, C, neg) \in W_R$.

The *minimal weighing relation for* R, denoted W°_R, is the intersection of all weighing relations for R.

Weighing results represent the knowledge on the relative weight of reasons. The weighing result (α, P, C, v) denotes that weighing the reasons in P, pleading for α, and those in C, pleading against α has the outcome v. Note that property (2) describes the weighing knowledge in case all reasons point in one direction. In the minimal weighing relation this is the only weighing information.

Definition 4. Consider a triple $\boldsymbol{T} = (F, R, W_R)$, where F is a set of RBL-propositions, R a set of RBL-rules (with names not of the form cs(r) or cd(r)), and W_R a weighing relation for R. Let $Atom_{\boldsymbol{T}}$ be the set of atoms that occur in a proposition in F, or in a rule in R, and $Names_{\boldsymbol{T}}$ the set of rule names that occur in an atom in $Atom_{\boldsymbol{T}}$. $\boldsymbol{T}$ is an *RBL-theory*, if $Names_{\boldsymbol{T}}$ is well-founded, in the following sense:

(1) If cs(r) $\in Names_{\boldsymbol{T}}$, then $r \in Names_{\boldsymbol{T}}$.

(2) If cd(r) $\in Names_{\boldsymbol{T}}$, then $r \in Names_{\boldsymbol{T}}$.

(3) If $r \in Names_{\boldsymbol{T}}$, and $r \neq$ cs(s), $r \neq$ cd(s), then $r \in R$.

A theory is called *closed*, if both R and W_R are closed; otherwise *open*.

Definition 5. Let $\boldsymbol{T} = (F, R, W_R)$ be a (closed) RBL-theory. $\boldsymbol{T}^* = (F^*, R^*, W_R^*)$, the *associated theory of* $\boldsymbol{T}$, is defined as follows:

$$F^* := F \cup \{ \varphi \leftrightarrow \text{applicable(cs}(r)) \mid r\colon \varphi \Rightarrow \alpha \in R, \text{cs}(r) \notin Names_{\boldsymbol{T}}, \text{cd}(r) \notin Names_{\boldsymbol{T}} \}$$
$$\cup \{ \neg\varphi \leftrightarrow \text{applicable(cd}(r)) \mid r\colon \varphi \Rightarrow \alpha \in R, \text{cs}(r) \notin Names_{\boldsymbol{T}}, \text{cd}(r) \notin Names_{\boldsymbol{T}} \}$$
$$R^* := R \cup \{ \text{cs}(r)\colon \text{condition\_satisfied}(r) \Rightarrow \text{applicable}(r) \mid r \in Names_{\boldsymbol{T}} \}$$
$$\cup \{ \text{cd}(r)\colon \text{condition\_denied}(r) \Rightarrow \neg\text{applicable}(r) \mid r \in Names_{\boldsymbol{T}} \}$$
$$W_R^* := W^\circ_{R^*} \cup W_R.$$

The rules with names of the form cs(r) and cd(r) of definition 4 are defined in definition 5, and play a special role. The requisite that the set of names of a theory should be well-founded means informally that each reference to a rule (either direct, or indirect) should be to a rule in R.

In definition 5 certain axioms are added to a theory. For each rule in a theory two rules, with names cs(r) and cd(r), are added. They define default reasons for applicability and non-applicability of a rule, namely the satisfaction and the denial of its condition.

Recall that the applicability of a rule just means that it constitutes a reason for its conclusion. Thus, if $r\colon \varphi \Rightarrow \alpha$ is a rule, $\varphi \leftrightarrow$ applicable(cs(r)) simply means that condition_satisfied(r) is a reason for applicable(r), if and only if the condition φ of r holds.

3.2 An extra rule of inference

Theories are implicitly assumed to be closed in the following definitions to avoid unnecessary attention to technicalities. Measures involving Skolemized versions of a theory should be taken to deal with theories containing open rules and weighing results.

Definition 6. Let $\boldsymbol{T} = (F, R, W_R)$ be a (closed) RBL-theory, and α an RBL-atom. We define the following sets:

$R^+(\alpha) := \{ \rho \mid \rho = r: \varphi \Rightarrow \alpha \in R \}$

$R^-(\alpha) := \{ \rho \mid \rho = r: \varphi \Rightarrow \neg\alpha \in R \}$

$R(\alpha) := R^+(\alpha) \cup R^-(\alpha)$

Definition 7. Let $\boldsymbol{T} = (F, R, W_R)$ be a (closed) RBL-theory. The relation $\vdash^\circ_{RBL}$ is the smallest relation, obeying the following three properties.

(1) For an RBL-proposition φ, $\boldsymbol{T} \vdash^\circ_{RBL} \varphi$, if $F \vdash_{FOPL} \varphi$. Here $\vdash_{FOPL}$ denotes the deduction relation in First Order Predicate Logic.

(2) Let φ and ψ be RBL-propositions. If $\boldsymbol{T} \vdash^\circ_{RBL} \varphi$, and $\boldsymbol{T} \vdash^\circ_{RBL} \varphi \rightarrow \psi$, then $\boldsymbol{T} \vdash^\circ_{RBL} \psi$.

(3) Let α be an RBL-atom. Suppose that we have

$\forall r: \varphi \Rightarrow \lambda \in R(\alpha)$: $\boldsymbol{T} \vdash^\circ_{RBL}$ applicable(r) $\vee$ $\boldsymbol{T} \vdash^\circ_{RBL} \neg$applicable($r$),

Define the sets

$P := \{ \varphi \mid \exists r: \varphi \Rightarrow \alpha \in R^+(\alpha)$: $\boldsymbol{T} \vdash^\circ_{RBL}$ applicable(r) $\}$,

$C := \{ \psi \mid \exists s: \psi \Rightarrow \neg\alpha \in R^-(\alpha)$: $\boldsymbol{T} \vdash^\circ_{RBL}$ applicable(s) $\}$.

Then:

(i) If $(\alpha, P, C, pos) \in W_R$, then $\boldsymbol{T} \vdash^\circ_{RBL} \alpha$.

(ii) If $(\alpha, P, C, neg) \in W_R$, then $\boldsymbol{T} \vdash^\circ_{RBL} \neg\alpha$.

In this situation the elements of P are called *reasons for* α, those of C *reasons against* α.

This is a preliminary derivability relation. The final version is given in definition 8. By (1) all tautologies of First Order Predicate Logic are RBL-derivable, hence property (2), Modus Ponens, suffices to include all FOPL-derivations in the derivability relation of Reason Based Logic. Property (3) is the essence of Reason Based Logic. It encompasses the main points, that applicable rules constitute reasons, and that a conclusion follows as a result of weighing reasons.

Definition 8. Let $\boldsymbol{T} = (F, R, W_R)$ be a (closed) RBL-theory, and φ an RBL-proposition. We define $\boldsymbol{T} \vdash_{RBL} \varphi$, if and only if $\boldsymbol{T}^* \vdash^\circ_{RBL} \varphi$.

3.3 Examples

Let's first consider the simplest case, and see how the conclusion of a rule follows from its condition, if no interfering rules are available. Because in Reason Based Logic this involves several steps, this first and basic example is worked out in detail.

We will consider the bird Tweety, and the rule that, if Tweety is a bird, she can fly. So, let the theory $\boldsymbol{T}_1 = (F_1, R_1, W_1)$ be defined by

F_1 := { bird(Tweety) },

R_1 := { f(Tweety): bird(Tweety) $\Rightarrow$ fly(Tweety) },

$W_1 := W^\circ_{R_1}$.

Then $\boldsymbol{T}_1^* = (F_1^*, R_1^*, W_1^*)$, the associated theory of $\boldsymbol{T}_1$, is

$F_1^* := F_1 \cup$ { bird(Tweety) $\leftrightarrow$ applicable(cs(f(Tweety))),

$\neg$bird(Tweety) $\leftrightarrow$ applicable(cd(f(Tweety))) },

$R_1^* := R_1 \cup \{$ cs(f(Tweety)): condition_satisfied(f(Tweety)) $\Rightarrow$ applicable(f(Tweety)),
cd(f(Tweety)): condition_denied(f(Tweety)) $\Rightarrow \neg$applicable(f(Tweety)) $\}$,
$W_1^* := W^{\circ}_{R_1^*}$.

We try to derive fly(Tweety). To apply (3) in definition 7 we must first derive applicable(f(Tweety)) or its negation. By (1) in definition 7 we have $\boldsymbol{T}_1^* \vdash^{\circ}_{RBL}$ bird(Tweety). Thus using bird(Tweety) $\leftrightarrow$ applicable(cs(f(Tweety))) $\in F_1^*$, and applying definition 5 and (2) in definition 7, we find $\boldsymbol{T}_1^* \vdash^{\circ}_{RBL}$ applicable(cs(f(Tweety))). By $\neg$bird(Tweety) $\leftrightarrow$ applicable(cd(f(Tweety))) $\in F_1^*$, we have $\boldsymbol{T}_1^* \vdash^{\circ}_{RBL} \neg$applicable(cd(f(Tweety))). Because cs(f(Tweety)), and cd(f(Tweety)) are the only rules in R_1^*(applicable(f(Tweety))), and only cs(f(Tweety)) is applicable, condition_satisfied(f(Tweety)) is the only reason concerning applicable(f(Tweety)). Applying (3) in definition 7 to applicable(f(Tweety)), using the weighing result

(applicable(f(Tweety)), { condition_satisfied(f(Tweety)) }, $\varnothing$, *pos*) $\in W_1^*$,

yields $\boldsymbol{T}_1^* \vdash^{\circ}_{RBL}$ applicable(f(Tweety)). There are no other rules concerning fly(Tweety), so this time applying (3) in definition 7 to fly(Tweety), and using

(fly(Tweety), { bird(Tweety) }, $\varnothing$, *pos*) $\in W_1 \subseteq W_1^*$,

we find $\boldsymbol{T}_1^* \vdash^{\circ}_{RBL}$ fly(Tweety). By definition 8 we finally arrive at $\boldsymbol{T}_1 \vdash_{RBL}$ fly(Tweety).

The next example is taken from the field of law, though imaginary. Pat, who is sixteen year old, is coming to trial for shoplifting. Normally she would be punished. But because it's her first offense, the judge does not convict her. The seventeen year old Bob, who attacked and injured somebody in a fight, got away with a warning, because it was his first offense. But John, sixteen years of age, was punished, even though it was his first encounter with the law: while stealing from a shop, he got involved in a fight and injured a customer, who was trying to stop him. This was considered too serious to let him get away with it.

In this case three rules are involved in a conflict. Pairwise priorities do suffice in Pat's and Bob's case (because in their cases only two rules are actually applicable), but in John's case the relative weight of the reasons provided by all three rules is needed to solve the conflict. This type of reasoning can be modeled easily in Reason Based Logic.

Define the theory $\boldsymbol{T}_2 = (F_2, R_2, W_2)$ by

$F_2 := \{$ steal(Pat) $\wedge \neg$injure(Pat) $\wedge$ minor(Pat),
$\neg$steal(Bob) $\wedge$ injure(Bob) $\wedge$ minor(Bob),
steal(John) $\wedge$ injure(John) $\wedge$ minor(John) $\}$,
$R_2 := \{ p_1$: steal(x) $\Rightarrow$ punish(x), p_2: injure(x) $\Rightarrow$ punish(x), p_3: minor(x) $\Rightarrow \neg$punish(x) $\}$,
$W_2 := W^{\circ}_{R_2} \cup \{$ (punish(x), { steal(x) }, { minor(x) }, *neg*),
(punish(x), { injure(x) }, { minor(x) }, *neg*),
(punish(x), { steal(x), injure(x) }, { minor(x) }, *pos*) $\}$.

In this definition the convention is broken that theories should be closed. Each element of R_2 and W_2 can be read as shorthand for its three instances, one for Pat, Bob and John. It is then straightforward to conclude the following:

$\boldsymbol{T}_2 \vdash_{RBL}$ applicable(p_1(Pat)) $\wedge \neg$applicable(p_2(Pat)) $\wedge$ applicable(p_3(Pat)),
$\boldsymbol{T}_2 \vdash_{RBL} \neg$applicable($p_1$(Bob)) $\wedge$ applicable(p_2(Bob)) $\wedge$ applicable(p_3(Bob)),
$\boldsymbol{T}_2 \vdash_{RBL}$ applicable(p_1(John)) $\wedge$ applicable(p_2(John)) $\wedge$ applicable(p_3(John)).

The denial of the conditions of rules p_2(Pat) and p_1(Bob) leads to their inapplicability. This is an important point: we need the knowledge that Pat did *not* injure someone and that Bob did *not* steal anything to get this far. The next section deals with this requirement of complete knowledge.

Using the weighing results

(punish(Pat), { steal(Pat) }, { minor(Pat) }, *neg*),

(punish(Bob), { injure(Bob) }, { minor(Bob) }, *neg*),

(punish(John), { steal(John), injure(John) }, { minor(John) }, *pos*),

we have the following outcome:

$\boldsymbol{T}_2 \vdash_{RBL} \neg$punish(Pat) $\wedge \neg$punish(Bob) $\wedge$ punish(John).

4 ADDING NEGATION AS FAILURE

4.1 Complete versus incomplete knowledge

A drawback of the formalization presented in section 3 is that it assumes complete knowledge on all relevant facts. In the second example above it was necessary to know that Pat did *not* injure anyone, and that Bob did *not* steal from a shop, to be able to conclude that they were not punished. This necessity stems from definition 7 (3). It is necessary to know of each rule in the set R(punish(x)) whether it is applicable or not.

This requirement can be relaxed. Because the applicability of a rule is itself derived on the basis of reasons, it seems natural to assume that it is *not* applicable in case it is not derivable that it is. Note that the satisfaction of the condition of a rule is a reason for its applicability, thus making a rule *by default* applicable if its condition is satisfied.

This idea is incorporated in the implementation as a Prolog program of an inference mechanism based on Reason Based Logic, described below. On the one hand it is simpler than the formalization of section 3, because the logical language and the derivability notion are restricted. On the other hand it is more flexible in dealing with incomplete knowledge, by using negation as failure for the applicability of rules. The code of the program is added in the appendix of this paper.

4.2 The Prolog program

The language used in the program is that of First Order Logic in an appropriate format. The following recursive definition is used:

1. Formula = Atom
2. Formula = [not, Formula]
3. Formula = [and, Formula, Formula+]
4. Formula = [or, Formula, Formula+]

where 'Atom' is an atomic formula written as a Prolog clause, and 'Formula+' stands for a sequence of one or more items 'Formula'. The only logical connectives that are used are 'not', 'and' and 'or'. No quantifiers are used.

Facts, rules and weighing results are represented in a knowledge base using the predicate given/1. A fact must be an atom or a negated atom, a rule has the form

rule(RuleId, Condition, Conclusion, ProOrCon)

Here 'Condition' and 'Conclusion' are respectively the condition and the conclusion of the rule. The conclusion must be an atom. The condition is a formula represented as a list as above. 'RuleId' is the identifier of the rule. 'ProOrCon' can have the values 'pro' or 'con', in accordance with whether the condition pleads for or against the conclusion. Weighing results have the following form:

weighing_result(Atom, ReasonsPro, ReasonsCon, Result)

where 'ReasonsPro' and 'ReasonsCon' are lists containing the reasons for and against the atom 'Atom', and 'Result' represents their relative weight.

The predicate reason/3 establishes which formulas are actually reasons:

reason(Conclusion, ProOrCon, Condition)

Here 'Condition' is a formula that is a reason for or against the statement 'Conclusion', according to the value of 'ProOrCon', that is 'pro' or 'con'.

It is defined as follows:

```
reason(Conclusion, ProOrCon, Condition):-
    given(rule(RuleId, Condition, ProOrCon, Conclusion)),
    derivable(applicable(RuleId)).
```

which means that a rule generates a reason precisely when its applicability can be derived.

The main predicate of the Prolog program is derivable/1. Its single variable must be an atom or a negated atom. If a formula 'Formula' is derivable from the knowledge base, the goal

derivable(Formula)

will succeed, otherwise it fails. If a formula is not given in the knowledge base, it can only be derived on the basis of reasons. For the exact definition the reader is referred to the appendix.

In the complete formalization of Reason Based Logic in section 3, a rule generates a reason, if it is derivable that it is applicable, and it does not generate a reason, if it is derivable that it is not applicable. This means that a derivation on the basis of reasons is blocked, if there is a rule of which neither the applicability, nor the inapplicability is derivable. In the Prolog program this requirement is relaxed using negation as failure. If it is derivable that a rule is applicable, it generates a reason. If it is not derivable, that it is applicable, it does not generate a reason.

As a result in the second example of section 3.3 it becomes derivable that Pat and Bob are not punished, even if it is not known whether or not Pat has injured anyone, and whether or not Bob has stolen anything.

5 LEGAL APPLICATIONS

In this paragraph some examples are given to show how Reason Based Logic can be applied to model legal knowledge and reasoning. The examples below follow the discussion in section 1.

Suppose we include the following (open) rule and weighing result in a theory.

exc: exception(r) $\Rightarrow \neg$applicable(r)

(applicable(r), { condition_satisfied(r) }, { exception(r) }, neg)

The meaning of the weighing result is that the reason for the applicability of a rule, that its condition is satisfied, is outweighed by the reason against, that there is an exception. If this scheme is included in a theory it becomes very simple to model rules and (undercutting) exceptions. For example, consider the general rule that an offense leads to punishment. An exception to this rule is made, if the offense has happened too long ago. But then again, there is an exception to this latter rule, which states that war crimes can never become outdated. The following rules will lead to the expected conclusions.

r_1: offense $\Rightarrow$ punish

r_2: out_of_date $\Rightarrow$ exception(r_1)

r_3: war_crime $\Rightarrow$ exception(r_2)

Another kind of exceptions are the 'rebutters' (see section 1). This involves two rules with opposite conclusions. In Reason Based Logic this can be modelled by including a suitable weighing result in the theory that solves the conflict.

Another scheme can be used to extend the application of a rule to a case that is not covered by its condition, but is within its purpose.

pur: purpose(r) $\Rightarrow$ applicable(r)

(applicable(r), { purpose(r) }, { condition_denied(r) }, pos)

For example, suppose it is forbidden to sleep in a railway station. At a certain moment, a judge might broaden the interpretation of this rule to the apparent intention of sleeping in a station, appealing to the purpose of the rule. This can be formalized as follows.

r_1: sleep_at_station $\Rightarrow$ fine

r_2: intention_of_sleeping $\Rightarrow$ purpose(r_1)

The implicit scope restrictions of legal rules are structurally modelled in much the same way as exceptions.

sc: $\neg$scope(r) $\Rightarrow$ $\neg$applicable(r)

(applicable(r), { condition_satisfied(r) }, { $\neg$scope(r) }, neg)

Suppose for example the rule that it is forbidden to drive faster than 120 km/h. An implicit scope restriction to this rule is that it holds in Holland, and not in Germany. Formally this can be represented by the following two rules.

r_1: speed>120 $\Rightarrow$ fine

r_2: $\neg$in_Holland $\Rightarrow$ $\neg$scope(r_1)

The basic scheme to model priorities between rules is this.

pr: priority(r, s) $\wedge$ applicable(r) $\Rightarrow$ $\neg$applicable(s)

(applicable(s), { condition_satisfied(s) }, { priority(r, s) $\wedge$ applicable(r) }, neg)

A priority relation is now simply represented as a set of facts of the form priority(r, s).

It is interesting to model the general principles Lex Superior, Posterior, and Specialis explicitly as rules.

lsup: lex_superior(r, s) $\Rightarrow$ priority(r, s)

lpos: lex_posterior(r, s) $\Rightarrow$ priority(r, s)

lspe: lex_specialis(r, s) $\Rightarrow$ priority(r, s)

Because each principle now just leads to a reason concerning the priority of a pair of rules, a conflict between these principles will only lead to a conclusion, if knowledge on the *priority of these principles* is explicitly available.

An example of a conflict involving more than two reasons was already given in section 3.3 as theory $\boldsymbol{T}_2$.

6 FINAL REMARKS

We have seen how Reason Based Logic can be used to model specific aspects of legal knowledge and reasoning. A strong point is the explicitness of the methods to resolve conflicts, only on the basis of knowledge. No magically unloaded guns (as in the Yale shooting problem) can occur. In Default Logic (Reiter, 1980) it can, because merely preserving consistency can be the cause not to apply a rule.

A formalization of the Yale shooting problem is the theory $\boldsymbol{T}_3 = (F_3, R_3, W_3)$ defined by

F_3 := { result(s_0, load, s_1), result(s_1, wait, s_2), result(s_2, shoot, s_3), holds(alive, s_0) },

R_3 := { y_1: result(x, load, y) $\Rightarrow$ holds(loaded, y),
y_2: result(x, shoot, y) $\wedge$ holds(loaded, x) $\Rightarrow$ $\neg$holds(alive, y),
y_3: result(w, x, y) $\wedge$ holds(z, w) $\Rightarrow$ holds(z, y) },
W_3 := W $^{\circ}_{R_3}$ $\cup$ { (holds(alive, y), { result(x, shoot, y) $\wedge$ holds(alive, x) },
{ result(x, shoot, y) $\wedge$ holds(loaded, x) }, *neg*) }.

The extra weighing result means that one does not stay alive by the frame axiom y_3 if one is shot at with a loaded gun. We have $\boldsymbol{T}_3 \vdash_{RBL} \neg$holds(alive, s_3). Without the extra weighing information neither holds(alive, s_3), nor $\neg$holds(alive, s_3) would be derivable, indicating that there is an unsolved conflict.

A disadvantage of the present formalization is that no means have been offered to handle reasoning with rules and weighing results. An elegant extension would be to relativize the validity of rules and weighing results so that those can be the result of weighing reasons themselves, as proposed by Hage (1993).

ACKNOWLEDGEMENTS

This research was partly financed by the Foundation for Knowledge-based Systems (SKBS) as part of the B3.A project. SKBS is a foundation with the goal to improve the level of expertise in the Netherlands in the field of knowledge-based systems and to promote the transfer of knowledge in this field between universities and business companies.

I would like to thank the other members of the ARCHIMEDES project for providing a stimulating environment, and especially my supervisor J.C. Hage for his fruitful ideas, and our lively discussions on the subject.

REFERENCES

Clark K.L., Negation as failure, in: Gallaire H., Minker J. (eds.) "Logic and data bases", New York, Plenum Press, 1978, p. 293-322.
Hage J., Monological reason based reasoning, in Breuker J.A. et al. "Legal knowledge based systems, Model-based legal reasoning", Lelystad, Vermande, 1991, p 77-91.
Hage J.C., Monological Reason-Based Logic. A low level integration of rule-based reasoning and case-based reasoning, in: "The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference", New York, ACM, 1993, p. 30-39.
Ginsberg M.L. (ed.), Readings in nonmonotonic reasoning, Los Altos, Morgan Kaufmann, 1987.
Lukaszewicz W., Non-monotonic reasoning. Formalization of commonsense reasoning, New York, Ellis Horwood, 1990.
Prakken H., Logical tools for modelling legal argument (PhD thesis), Amsterdam, 1993.
Prakken H., A logical framework for modelling legal argument, in: "The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference", New York, ACM, 1993, p. 1-9.
Reiter R., A logic for default reasoning, Artificial Intelligence 13, 1980, pp. 81-132.
Sartor G., A simple computational model for non-monotonic and adversarial legal reasoning, in: "The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference", ACM, New York, 1993, p. 192-201.

APPENDIX: THE PROLOG PROGRAM

```
% reason(Conclusion, ProOrCon, Reason)
reason(Conclusion, ProOrCon, Condition):-
```

```
        given(rule(RuleId, Condition, ProOrCon, Conclusion)),
        derivable(applicable(RuleId)).

% derivable(AtomOrNegatedAtom)
derivable(AtomOrNegatedAtom):-
        given(AtomOrNegatedAtom).
derivable(condition_satisfied(RuleId)):-
        given(rule(RuleId, Condition, _, Conclusion)),
        satisfied(Condition).
derivable(applicable(cs(RuleId))):-
        derivable(condition_satisfied(RuleId)).
derivable(Atom):-
        not(is_list(Atom)),
        collect_reasons(Atom, Reasons),
        reason_sort(Reasons, ReasonsPro, ReasonsCon),
        weigh_reasons(Atom, ReasonsPro, ReasonsCon, pos).
derivable([not, Atom]):-
        not(is_list(Atom)),
        collect_reasons(Atom, Reasons),
        reason_sort(Reasons, ReasonsPro, ReasonsCon),
        weigh_reasons(Atom, ReasonsPro, ReasonsCon, neg).

% collect_reasons(Atom, Reasons),
collect_reasons(Atom, Reasons):-
        collect(reason(Atom, ProOrCon, Reason), reason(Atom, ProOrCon, Reason), Reasons),
        match(Atom, Reasons).
match(Atom, []).
match(Atom, [reason(Atom, ProOrCon, Reason)|Rsns]):-
        match(Atom,Rsns).

% weigh_reasons(Atom, ReasonsPro, ReasonsCon, Result)
weigh_reasons(_, RsnsPro, [], pos) :-
        RsnsPro \== [],
        !.
weigh_reasons(_, [], RsnsCon, neg) :-
        RsnsCon \== [],
        !.
weigh_reasons(Atom, ReasonsPro, ReasonsCon, Result):-
        given(weighing_result(Atom, ReasonsPro, ReasonsCon, Result)).

% reason_sort(RsnLst, ProRsnLst, ConRsnLst)
reason_sort([], [], []).
reason_sort([reason(_, pro, ProRsn)|Rsns], [ProRsn|ProRsns], ConRsns) :-
        reason_sort(Rsns, ProRsns, ConRsns).
reason_sort([reason(_, con, ConRsn)|Rsns], ProRsns, [ConRsn|ConRsns]) :-
        reason_sort(Rsns, ProRsns, ConRsns).

% Definition 5, section 3.1
given(rule(cs(RuleId), condition_satisfied(RuleId), pro, applicable(RuleId))):-
        given(rule(RuleId,_,_,_)),
        not RuleId = cs(_).

% satisfied(Formula)
satisfied(Atom) :-
        not(is_list(Atom)),
        derivable(Atom).
satisfied([not, Atom]):-
        not(is_list(Atom)),
        derivable([not, Atom]).
satisfied([not, [and, Formula1, Formula2]]) :-
        satisfied([or, [not, Formula1], [not, Formula2]]).
satisfied([not, [and, Formula|Formulas]]) :-
        satisfied([or, [not, Formula], [not|Formulas]]).
satisfied([not, [or, Formula1, Formula2]]) :-
        satisfied([and, [not, Formula1], [not, Formula2]]).
satisfied([not, [or, Formula|Formulas]]) :-
        satisfied([and, [not, Formula], [not|Formulas]]).
satisfied([and, Formula1, Formula2]):-
        satisfied(Formula1),
        satisfied(Formula2).
```

```
satisfied([and, Formula|Formulas]):-
        satisfied(Formula),
        satisfied([and|Formulas]).
satisfied([or, Formula1, Formula2]):-
        satisfied(Formula1);
        satisfied(Formula2).
satisfied([or, Formula|Formulas]):-
        satisfied(Formula);
        satisfied([or|Formulas]).
```