Legal knowledge based systems
JURIX 93
Intelligent Tools for Drafting Legislation, Computer - Supported Comparison of Law

The Foundation for Legal Knowledge Systems
Editors:
J.S. Svensson
J.G.J. Wassink
B. van Buggenhout

Cees groendijk and Henning Herrestad, An incremental approach to legal drafting support, in: J.S. Svensson, J.G.J. Wassink, B. van Buggenhout (eds.), Legal knowledge based systems JURIX 95: Intelligent Tools for Drafting Legislation, Computer - Supported Comparison of Law, Lelystad: Koninklijke Vermande, pp. 31-42, 1993 ISBN 90 5458 089 5.

More information about the JURIX foundation and its activities can be obtained by contacting the JURIX secretariat:

Mr. C.N.J. de Vey Mestdagh
University of Groningen, Faculty of Law
Oude Kijk in 't Jatstraat 26
P.O. Box 716
9700 AS Groningen
Tel: +31 50 3635790/5433
Fax: +31 50 3635603
Email: sesam@rechten.rug.nl

 http://jurix.bsk.utwente.nl/

AN INCREMENTAL APPROACH TO LEGAL DRAFTING SUPPORT

CEES GROENDIJK AND HENNING HERRESTAD

Summary

We present an approach to the development of intelligent drafting tools where the drafter is offered a ladder of incremental steps in formalization and application of LKBS techniques, each step enabling a new kind of check on the draft. We describe what benefits the drafter will gain at each step of the ladder and we describe some of the possibilities of simulation within reach at the top of the ladder.

1. Introduction

Various methods to support legal drafting with automated tools have been suggested. A common approach is to develop 'intelligent document assembly tools' to support drafters in creating legal documents from ready-made pieces of text. Although a success in the restricted legal domain of repetitive legal document drafting [Staudt, 1993], this approach is not suitable if we are to make a more general, context-free tool to enhance legal drafting. The point of departure in our approach is closer to the approach known as 'normalization' [Allen & Saxon, 1985][Allen & Saxon, 1988]. Normalized drafts are supposed to be less ambiguous and more readable than conventional drafts by making explicit the use of propositional logical connectives. However, normalization attempts became entangled in the double ambitions of becoming accepted as a typographical standard by the legislative bureaucracy and of making legal expert systems [Gray, 1985][Gray, 1988]. Moreover, the attempts of [Allen & Saxon, 1986] at further developing normalization were rendered impractical since the new answers they propose to distinguish between different sentence interpretations were too fine-grained to bother with. Yet another way to enhance legal drafting is to simulate some of the effects of the proposed legislation. In the Seventies non-generalizable simulation programs were tailor-made to simulate effects of particular legal drafts, while in the early Eighties attention turned to expert systems [Bing, 1991]. Anticipating a revived interest following the successes of [Svensson, 1993] and [denHaan & Breuker, 1991], we hereby present some ideas concerning how to use knowledge-based-system techniques to support the process of drafting legislation and to do simulation.

One of the main problems in using LKBS techniques to support drafting legislation is the fact that the drafter has to do substantial formalization. We have doubts about the willingness of a drafter to accept such inconveniences and about the more fundamental issue of whether a general drafting tool can provide a formal language capable of representing all the important concepts of newly drafted legislation. Perhaps we should be pessimistic. Drafting law is a highly creative process, requiring the freedom of unrestricted use of natural language if the drafter is to give the most ample expression to her thought. However, we might be more optimistic. Even if different legislation looks different on the surface, it is generally agreed by legal philosophers that most legislation has a common underlying structure which can be formalised (e.g. [Susskind, 1987]).

Whether and to what extent drafting legislation can be supported, depends on many factors such as the characteristics of the legal domain the new legislation refers to. Our approach is to offer the drafter a set of tools to support the drafting process in a step-by-step manner. We have called this an 'incremental' approach because, with each step, the enhancement of the draft is taken a little further. With each step, the drafter is free to

J.S. Svensson, J.G.J. Wassink and B. van Buggenhout (eds.) (1993). Legal Knowledge Based Systems: Jurix '93: Intelligent Tools for Drafting Legislation, Computer-Supported Comparison of Law

decide whether proceeding is worth the effort. The first step consists of adding propositional logical connectives to the text. This step is easy and involves only elementary assumptions about the underlying structure of the draft. Each additional step is a little more ambitious, requiring more effort and assuming more about the underlying structure of the draft. However, with each step there is added benefit.

We have implemented a Legal Knowledge Based System (LKBS) that we have named Prodeon. Prodeon is capable of reasoning with propositions, predicates and deontic operators (section 2). It is presented here because it has served as a point of departure and a source of inspiration for our ideas how the process of legislation can be supported with LKBS techniques. In its current state it is a LKBS and not a tool for drafting. As a result, most of our suggestions would require some modification of the Prodeon framework.
Section 2 provides a brief overview of the Prodeon framework. Sections 3 to 9 describe each of the seven steps we suggest for improving the drafting process.

2. Prodeon

Prodeon is designed for representing fragments of law, using rules, propositions, predicates and deontic operators. Prodeon can use forward chaining to deduce all consequences of a given fact situation, or use backward chaining to try to reach a user-provided goal. One of the major guidelines for the creation of Prodeon was that its knowledge representation language should be easy to use. Therefore, the language allows one to use simple constructs first, and then gradually to extend the language to a more complex and rich knowledge representation.

As a starting point, one can use propositions as elementary building blocks of the knowledge representation. Syntactically, a proposition must be a sentence enclosed with brackets. Inspired by the language of normalization [Allen & Saxon, 1985], we will call such a proposition a *constituent sentence* (cs). A constituent sentence can be negated explicitly by preceding it with *NOT*.

(1) [This is a constituent sentence]
(2) NOT [The legislator applies the law]

In Prodeon, legal rules are represented as IF-THEN rules. These IF-THEN rules are simple production rules; contrapositive inferences (modus tollens) are not made. Antecedents and consequents may consist of combinations of constituent sentences connected by AND or OR. Prodeon uses a prefix notation for these connectives. E.g:

(3) (IF [AND [person is unemployed][available for a job]] THEN [person receives social benefit])

Prodeon allows the use of variables and constants in constituent sentences. Variables are denoted as :<variable>; literal constants are simply written out and preceded by a colon. Hence, the basic building block of the knowledge representation may also be a predicate. We will refer to both propositions and predicates as constituent sentences.

(4) (IF [:<person> is caught in the act] THEN [:<person> is guilty])
(5) [:John claims copyright of :book]

Both on the conceptual and on the implementational level, Prodeon distinguishes between two types of constituent sentences. A constituent sentence is of type *sein* if it refers to the world as it is and of type *sollen* if it refers to what ought to be the case. Syntactically, a constituent sentence is of type *sollen* if it is preceded by O (O representing 'obligatory'). E.g.:

(6) O [:John pays :Mary]

As indicated earlier, a constituent sentence of type *sein* may be negated by putting the word NOT in front of the constituent sentence. With constituent sentences of type *sollen*, the situation is a bit different. The norm may be negated in two manners: before and after the deontic operator O. Thus NOT O[*cs*], O NOT [*cs*] and NOT O NOT [*cs*] are all valid expressions. Following Standard Deontic Logic [Follesdal & Hilpinen, 1971], O NOT [*cs*] can also be written as F[*cs*] and NOT O NOT [*cs*] can be written as P[*cs*] (F and P representing 'forbidden' and 'permitted'). In fact, Prodeon will accept any combination of negations and one of the operators O, F, or P. Internally, it will be rewritten to its equivalent form using O (e.g. NOT P[*cs*] is rewritten to O NOT [*cs*]).

Prodeon may either run forward or backward. To this end, a general (i.e. non-instantiated) version of each constituent sentence is stored with four lists associated to it: seinforward, seinbackward, sollenforward and sollenbackward. The forward lists associates each constituent sentence to rules in which the constituent sentence is used in the antecedent; the backward lists associates constituent sentences to rules in which the constituent sentence is used as a consequent. To determine whether a rule is applicable, Prodeon uses unification. Because *sein* and *sollen* is strictly separated, the rule selection scheme ensures that Prodeon never attempts to unify a constituent sentence of type *sein* with a constituent sentence of type *sollen*. With respect to constituent sentences of type *sein*, if the systems attempts to unify two constituent sentences, it proceeds if both constituent sentences are either negated or not negated and fails immediately otherwise. With respect to constituent sentences of type *sollen*, the scheme is slightly more complicated. Different combinations of negations fail immediately with two exceptions. Firstly, a query consisting of a permitted constituent sentence (i.e. NOT O NOT [*cs*]) is attempted to unify with an obligation (O[*cs*]) to express the rule that everything that is obligatory is also permitted. Secondly, a query consisting of a negated obligation (not O[*cs*]) is attempted to unify with a forbidden constituent sentence (O not [*cs*]) to express the rule that everything that is forbidden cannot be obligatory. The difference between *sein* and *sollen* also has consequences for the way conjunctions and disjunctions are handled but we will not discuss these differences here.

In the following we shall present a number of ideas as to how the Prodeon framework may be adapted to the drafting of legislation and how the draft may be adapted to fit the Prodeon formalism. These ideas, however, have not been implemented. Sections 3 to 6 deal with formalization: propositional logical connectives, variables, deontic operators, action operators and normative positions. Sections 7 to 9 deal with finding inconsistencies, finding imperfect law and simulating some of the effect of the draft. Most of the steps, but not all, require previous steps to be performed. However, each step is performed for its own sake and benefit. The aim of the first four steps is to improve the structure, readability and clearness of the draft by letting the drafter formalise certain very general aspects of new legislation. The drafter has a choice between two methods to do the formalization. Firstly, the drafter may choose to introduce the formal elements from a menu while writing. The advantage of this method is to have more certainty that a correct formalization is made. Secondly, she may choose to write freely and make the program attempt to introduce the formal elements afterwards by searching for words denoting concepts that may be formalised. The advantage of this method is to allow the drafter to write unfettered. But this advantage is bought at the cost of having to steady and correct the machine in making the formalization.

3. Step 1: Making propositional logical connectives explicit

As a first step, introducing propositional logical connectives may help to eradicate sentence ambiguity. Before sentence ambiguity can be settled, the drafter needs to mark what the machine should treat as a separate sentence, or, in the language of normalization, a *constituent sentence* (cs). A drafted rule may for instance state that:

(7) "If a person is aged 18, and is mentally sane, or is supported by a guardian, then he may enter into contracts".

The constituent sentences may be marked by square brackets in the following way:

(8) If [a person is aged 18,] and [is mentally sane,] or [is supported by a guardian,] then [he may enter into contracts].

To the machine the rule, which at this stage is still ambiguous, will now be represented as:

(9) If [cs-1] and [cs-2] or [cs-3] then [cs-4].

How could the program have discovered where to place the brackets? We suggest that the program is made to search for words like *if*, *then, and, or*, bracketing everything between them interpreting the commas as suppressed occurrences of *and*, or *or*. There may also be long pieces of text where no such words are found. This text will simply be put in one bracket pair, as it is only set aside by the machine as a separate proposition. We do not pretend this to be a foolproof method, and we would demand the drafter to check the result before proceeding. Actually, to the program it is the connectives that may be formalised. The machine will capitalise these words, regarding them as the connectives of propositional logic, or, the drafter may choose the connectives from a menu, and the program will automatically bracket what is in between. Experience with Prodeon has taught us that prefix notation for *and*, and *or* makes the representation more readable than infix notation, especially if the conjuncts or disjuncts consist of more than two elements. After the connective is introduced, the program therefore rearranges the sentences to make the first part of the rule look as follows:

(10)
```
(IF
        [AND
                [a person is aged 18,]
                [is mentally sane,]
        ]
)
```

Additional constituent sentences connected by AND would just be added underneath. However adding OR creates the ambiguity of whether we mean ((a AND b) OR c) or (a AND (b OR c). In the first meaning the conjunction is nested within a disjunction, in the second the disjunction is nested within the conjunction. The program has to ask the user to clarify which meaning is intended. Here, choosing the latter meaning, the rule is written:

(11)
```
(IF
        [AND
                [a person is aged 18,]
                [OR
                        [is mentally sane,]
                        [is supported by a guardian,]
                ]
        ]
 THEN
        [he may enter into contracts.]
)
```

The brackets and indentation effectively marks at which level of nesting we find the connected constituent sentences, disambiguating the meaning of the full sentence. If the drafter chooses to let the machine generate this construct automatically, we encounter the problem that the natural language words *and* and *or* sometimes change meaning. The

program therefore has to ask for each occurrence whether the intended meaning is that of conjunction or disjunction.

More structure may be added to the draft by making negation explicit. The easiest way is to choose NOT from the menu while writing and then write out the constituent sentence to be negated. E.g.:

(12) NOT [he may enter into contracts]

The second method is to let the machine search for occurrences of 'not' and 'no' within constituent sentences, and, if such an occurrence is found, ask the drafter whether the negation should be made explicit, i.e. lifted out of the constituent sentence and put in front of it. Obviously, this method will not find all constituent sentences that indicate a negation since there are many ways of expressing negation in natural language.

4. Step 2: Equalising propositions and adding variables

So far, a constituent sentence was simply a sentence used in the draft. However, the content of a constituent sentence may have many properties which are of interest if only the drafter is willing to point them out.
Firstly, a drafter will often use different wordings to express identical meanings. For a formalization it is important that constituent sentences that are identical or very close in meaning are represented as equal. The program might support the drafter in the following way. The program provides a list of all the constituent sentences used sofar in the formalization. The drafter may then see that the two constituent sentences are actually equivalent for the purposes of the formalization and decide that one of the two should be replaced by the other, making them all the same. The method can be enhanced by using text retrieval methods to order the list in such a way that constituent sentences with a high number of identical words (stop words not included) appear next to each other because these constituent sentences are often close in meaning.
Secondly, the list of constituent sentences may also be shortened by recognising that two constituent sentences are actually identical if some part of the constituent sentences is replaced by a variable. Suppose, for instance, that we have the following two constituent sentences in the list:

(13) [The buyer has possession of the goods]
(14) [Claimant has possession of the goods]

Probably, if some part of these constituent sentences is replaced with a variable (cf. section 2), these two can be replaced by:

(15) [:<person> has possession of the goods]

Subsequently, the program should present the drafter all the rules using (15) and ask what other characteristics will limit the instantiation of the variable for each rule.

5. Step 3: Making deontic operators explicit

Legal rules often state that in certain factual situations certain normative consequences may be drawn. Consequences are normative when they state what may or ought to be the case rather than what is or will be the case. Words like may and ought may be treated as modal qualifiers, qualifying the subsequent constituent sentence as descriptions of something ideal or optimal; and as implicitly stating that the actual circumstances may deviate from the ideal. I.e. in expressing something as obligatory, one implicitly recognises the possibility of transgression (cf. [Jones & Pörn, 1985] [Jones & Sergot, 1991]).

To support a further formalization of the draft, the program should offer the drafter the possibility to mark a constituent sentence with one of the deontic modal qualifiers *Obligatory*, *Forbidden*, and, *Permitted* (abbreviated to *O, F* and *P*) in any combination with negation (i.e. NOT may be placed before and after the qualifier). Using standard deontic logic in defining F[*cs*] = O NOT [*cs*] and P[*cs*] = NOT O NOT [*cs*] [Follesdal & Hilpinen, 1971] the program will translate each combination to what is probably the most readable form (i.e O[*cs*], F[*cs*], P[*cs*] and NOT O[*cs*]). We do not think it is possible to design a method to let the program point out the deontic modalities by itself; there are simply too many ways to express deontic modalities in natural language.
The aim of this step is primarily to make the deontic modalities explicit and to let the formalization be more equal to what Prodeon can accept. But also some inconsistencies may be detected during this stage. For instance, the program may detect that it is inconsistent to create the following two rules:

(16) (IF [:<person> is unemployed] THEN O[:<person> search for a job])
(17) (IF [:<person> is unemployed] THEN P NOT [:<person> search for a job])

A further aim is to prepare the formalization for the steps to come.

6 . Step 4: Clarifying action operators and normative positions

Inside the normatively qualified constituent sentences there are certain elements which are sufficiently general to merit another step up the ladder of formalization. Generally, normative sentences refer to an agent and an action being performed by the agent (cf.[Kanger, 1972], [Lindahl, 1977] and [Jones & Sergot, 1992]). The action may be marked by a DO operator to be inserted between the deontic operator and the constituent sentence (action descriptions are so varied and deeply submerged in natural language that we do not see any possibility for the program discovering them itself). The agent is represented by a variable or a constant as described earlier, and the program will secure that an agent variable or constant is declared as attached to the DO operator whenever the operator is used. We may thus construct expressions such as:

(20) P DO :John [:John has possession of :car]

to represent "It is permitted for John to bring it about that John has possession of the car". Hence, the DO operator may intuitively be read as "<x> brings it about that [*cs*] is true". The DO operator does not distinguish between intentional and non-intentional action, or active and passive actions (e.g. opening the door and keeping the door open), nor does it apply to laudable but unsuccessful attempts. The only property of the DO operator is to state that what has been brought about is a fact. This makes it inconsistent to state the following:

(21) DO :<x> [*cs*] AND NOT [*cs*],

which also makes it inconsistent to state the following:

(22) DO :<x> [*cs*] AND DO :<x> NOT [*cs*]

If the drafter finds that some action description in her draft confirms to the representation of the DO operator, she may use it. Even if the DO operator abstracts away most of the subtleties of our various concepts of action, the operator is still useful. It allows us to distinguish between "x brings it about that [*cs*] is true", and "x brings it about that the negation of [*cs*] is true":

(23) DO :<x> [*cs*]
(24) DO :<x> NOT[*cs*]

The statement: "x is passive, or not responsible, for the truth or falsity of [*cs*]", is an intuitive interpretation of:

(25) NOT DO :<x> [*cs*] AND NOT DO :<x> NOT [*cs*]

As (25) is a quite cumbersome statement, yet with great expressive potential, we shall give it a special operator. We may represent statements like "x is obliged to be passive (not responsible, unrelated) to the truth or falsity of [*cs*]" as:

(26) O PASS :<x> [*cs*]

With the introduction of PASS the possible precision of expressions of permission is increased. With the use of the deontic operator as described in section 5, freedom of choice could be expressed as: P[*cs*] AND P NOT[*cs*]. Now, freedom of choice may be expressed as:

(27) P DO :<x> [*cs*] AND P DO :<x> NOT[*cs*] AND P PASS :<x> [*cs*]

Furthermore, there are three additional expressions of forms of 'restricted' freedom of choice. Restricted because one of the conjuncts is forbidden:

(28) P DO :<x> [*cs*] AND P DO :<x> NOT[*cs*] AND F PASS :<x> [*cs*]
(29) P DO :<x> [*cs*] AND F DO :<x> NOT[*cs*] AND P PASS :<x> [*cs*]
(30) F DO :<x> [*cs*] AND P DO :<x> NOT[*cs*] AND P PASS :<x> [*cs*]

The program may try to make the drafter disambiguate her expressions of permission by asking which of the above statements is closest to the drafter's intention. However, the drafter may feel unable to make a categorical decision about this. The statements may be too specific for her draft. We do not want to force a decision. The point is only to show her the available options. She may use these options or abstain from them as she likes.

If freedom is restricted further, by forbidding two of the conjuncts, we have the equivalent of respectively:

(31) O DO :<x> [*cs*]
(32) O DO :<x> NOT[*cs*]
(33) O PASS :<x> [*cs*]

Together, the statements 27 to 33 are exhaustive in describing the seven possible normative positions of a single agent. The disjunction of the seven positions is a tautology, and the conjunction of any of the positions is a contradiction. Hence, there is an advantage of specifying the normative conclusion of a rule as a particular normative position. This is the possibility of identifying as inconsistent any set of normative statements expressing that <x> has more than one normative position with respect to the same proposition.

This only briefly illustrates what may be achieved when we start to expand the expressiveness of our formalism by adding an action operator to the deontic operators. Further increases in expressibility, adding relations between agents or iterations of operators, are described in [Lindahl, 1977]. [Jones & Sergot, 1992] describe how the whole approach may be generalised to take into account a whole range of other possible operators. However, the resulting sets of nuances of expression very quickly becomes very large and each formal statement very complicated. We fear that, by presenting the drafter with several thousand possible nuances of interpretations even for simple sentences, we may ridicule the whole approach. We therefore believe that development of the tool in the direction of an expanded formalism ought to be guided by more experience

with legal drafting, in order to be certain that the nuances of interpretation are of practical value.

7. Step 5: Finding inconsistencies

As a general rule, the application of legislation should not lead to contradictions. For instance, if the following rules

(34) (IF [:<X> is married] THEN [:<X> receives tax reduction])
(35) (IF [:<X> receives unemployment benefit] THEN NOT [:<X> receives tax reduction])

are applied to the following test case

(36) [:John is married]
(37) [:John receives unemployment benefit]

then Prodeon is able to derive

(38) [:John receives tax reduction]
(39) NOT [:John receives tax reduction]

which is an inconsistency that should be pointed out to the drafter.

Basically, the approach to be taken to find such inconsistencies is to create a set of test cases, apply the formalised version of the new legislation to these test cases one by one and check the results for inconsistencies. An obstacle to this approach is that one has to find or create a set of test cases. In some domains, one may be lucky enough to have a set of test cases available beforehand which can be adapted to the formalization used. [Svensson, 1993] describes various methods to adapt the set of test cases to a LKBS. However, such a set of test cases will usually not be available. Hence, we will first discuss some methods to create a set of test cases.

A first method is to create the set of test cases by hand. The drafter, probably, knows to what cases the new legislation should apply. Prodeon can only apply the rules to a test case if the case is expressed as a set of constituent sentences which are also used in the formalised version of the new legislation. This observation leads to two remarks. Firstly, the program can be used to support the creation of test cases by presenting all the constituent sentences used in the formalised rules and offering the facility to make selections, store cases etc. The second point is of a different nature: if the drafter feels that the constituent sentences offered do not allow her to express the crucial points of a test case she has in mind, then the rules are most certainly not detailed enough. Hence, creating test cases can also be seen in itself as a check on the drafting process.

A second method to obtain a set of test cases is to generate the cases automatically. For instance, one might suggest to create all possible test cases by creating all possible combinations of constituent sentences exhaustively (cf. [Alchourrón & Bulygin, 1971]). There are, however, two major problems with this approach. Firstly, such an approach would usually lead to more cases than one can handle [Nieuwenhuis, 1989], [Breuker, 1991]. For instance, even in a well structured domain such as traffic regulations, combining relevant traffic characteristics to obtain all possible traffic situations (types of roads, types of actions etc.) may easily lead to a combinatorial explosion [Breuker, 1991]. A second problem is that if constituent sentences are put together blindly, many of the resulting cases would not make sense as a description of a real-word situation and, hence, these cases are useless as with regard to finding inconsistencies. Depending on the domain, both problems can be tackled more or less successfully by using world knowledge to constrain the case generation in such a way that an acceptable number of

valid cases is produced. For instance, [Nieuwenhuis, 1989] uses in its Rule Analysing Tool common-sense knowledge to filter out the test cases with mutually exclusive facts (e.g. male and pregnancy).

If a set of test cases is available or made available, Prodeon is able to find inconsistencies in the formalised version of the draft, by taking the test cases one by one and applying all applicable rules to constituent sentences of the test case forwardly until no addition constituent sentences can be deduced. Then all the constituent sentences are checked for inconsistencies, i.e. Prodeon searches for similar constituent sentences with incompatible prefixes. Obviously, the following

(40) [*cs*] & NOT[*cs*]

constitute an inconsistency. If deontic operators are used in the formalised draft, the following (and equivalent) combinations are inconsistencies

(41) O[*cs*] & NOT O[*cs*]
(42) O NOT [*cs*] & NOT O NOT [*cs*]

The latter may also be read as (F[*cs*] & NOT F[*cs*]) or (NOT P[*cs*] & P[*cs*]). Prodeon may also discover inconsistencies in the form of a normative conflict:

(43) O[*cs*] & O NOT [*cs*]

which can also be read as (F NOT[*cs*] & F[*cs*]). The two combinations:

(44) O[*cs*] & NOT O NOT [*cs*]
(45) O NOT [*cs*] & NOT O[*cs*]

which can also be read as (O[*cs*] & P[*cs*]) and (F[*cs*] & NOT O[*cs*]) are not considered inconsistent because in Prodeon permission only means negation of prohibition; permission is entailed by obligation (cf. section 2) and thus also compatible with obligation. If, however, permission is interpreted as a permission of choice, i.e. a permission entails the stronger statement P[*cs*] AND P NOT [*cs*], then (44) and (45) should also be considered inconsistencies.

Additional inconsistencies may be found if the formalization uses action operators and normative positions. These are already described in section 6.

Even if the program succeeds in finding inconsistencies, there may be other properties of the text (i.e. hierarchical structures or other relations of preference) that makes the drafter decide that the inconsistency is only apparent or not a problem. Such relations have been the matter of intensive study [Prakken, 1993], and may possibly be explicitly represented and reasoned within some future version of our program. Our present ambition is only to point out the inconsistencies to the drafter, who may decide whether they are real, unintended contradictions or only appear to be.

8. Step 6: Finding lege imperfecta

With new legislation, new obligations and prohibitions are created. But what to do if people break their obligations or perform acts that are forbidden? Ideally, the draft should give an answer to this question. To put it differently, the draft should provide, what can be called, *repair* rules for the following two combinations:

(46) O[*cs*] & NOT [*cs*]
(47) O not [*cs*] & [*cs*]

(the latter being equivalent to F[*cs*] & [*cs*]). One way to be sure that a repair rule is present for each new obligation or prohibition, is to ask the drafter to point these rules out. To support this, the program could make a list of rules out of the formalised version of the draft that are potential repair rules and present them to the drafter. For instance, in case of a new prohibition F[*cs*], the top of the list of potential repair rules should consist of the rules with both F[*cs*] and [*cs*] in the antecedent. Since repair rules often do not repeat the prohibition F[*cs*] in the antecedent, the list should further include rules which refer to the prohibited behaviour [*cs*] in the antecedent without referring to F[*cs*].

Having repair rules is one thing. They should also be applicable in cases where they are needed. Applicability is not certain, because repair rules may often have more preconditions than the violation they try to cure. For instance, there may be several repair rules for the same violation, each referring to different situations in which the violation takes place. The method to check that at least one repair rule is applicable is somewhat similar to the procedure presented in the previous section using a set of test cases (section 7). The formalised version of the new legislation is applied to each of the test cases and Prodeon checks whether one of the combinations (46) or (47) occurs. If so, it is verified that at least one repair rule, pointed out by the drafter, has indeed been applied.

Repair rules often create new obligations or prohibitions when applied. Hence, the requirement that obligations and prohibitions should be accompanied by a repair rule cannot be made too rigid because, otherwise, this requirement could lead to an unending regress of repair rules, each repair rule creating an obligation or prohibition which may be violated again.

9. Step 7: Simulating draft application

The previous two sections described the application of the formalised version of the draft to test cases with the aim to evaluate certain technical aspects of the draft. When the draft has reached a stage of near completion and a high level of technical soundness, one might be more interested in the question whether the new legislation will indeed have the results the legislator wishes to achieve by bringing the legislation into operation. In this respect we may distinguish between two levels of evaluation: the micro and the macro level (cf.[Svensson, 1993]). On the micro level, an evaluation is made of the effects the new legislation will have when applied to individual cases. A macro level evaluation tries to obtain detailed data of some of the effects the new legislation will have on the society.

With respect to the micro level, the approach is straightforward. Apply the formalised version of the draft to the test cases one by one and check whether the results are in accordance with what one might expect. The test cases already created to find inconsistencies and lege imperfecta will do as a starting point. Several researchers have had success with this approach. For instance, [denHaan, 1992] used a LKBS to validate the new Traffic Regulations (RVV90) and found various flaws (e.g. just to name a hilarious result: the new regulation prescribes that in some situations, trams should be on the road instead of on the railway). Svensson has used the technique quite extensively to evaluate the Revision of the General Social Security Act [Svensson, 1993]. His study shows that results of automated rule application should be evaluated very carefully. Most of the unexpected results originated from errors in the test cases or errors in the formalization of the draft. Hence, if results of the simulation are not to the satisfaction of the drafter, she should not immediately return to the drafting table, but should first examine the rules in the rule-based system as well as the definition of the case that produced the unexpected result. However, Svenssons evaluation also shows that errors may be found in a draft. His evaluation exhibited that the Revision of the General Social Security Act contained an erroneous method to determine the *living cost supplement*.

In certain domains, one might attempt to extend the approach presented above to make a macro level evaluation of the draft. Basically, the idea is to apply the formalised version

of the draft to a large number of test cases which preferably should be representative for the population of subjects the new legislation will apply to. Instead of studying each case separately, the results are integrated (e.g. by statistical means). For instance, such an evaluation should be able to answer questions like: *How many people will receive less social benefit after the Revision of the General Social Security Act has been made operational*. [Svensson, 1993] provides a detailed study and discussion of this idea.

A serious limitation of this approach is that it only measures so-called (cf.[Svensson, 1993]) first-order effects. The approach assumes that everything remains the same except for the new legislation. In many domains this is an unrealistic assumption. People do not simply wait for the new legislation to be applied to them, they change their behaviour or change the situation they are in to profit from the most favourable rules and to avoid others. In fact, behavioural change is often the main purpose of newly issued legislation. Hence, in many domains this approach may not be useful.

However, in some domains, first order effects are interesting and are not disturbed by higher order effects. Also in some domains it may be useful to measure first-order effects in order to obtain insight in higher order effects. For instance, a government may be planning to issue an amendment to the Tax Law in order to increase certain taxes for unemployed people with the idea that, as a result, unemployed people will be more motivated to apply for badly paid jobs and thus unemployment is reduced. Obviously, without actually bringing the amendment in operation, it will be very hard to forecast whether unemployment will actually reduce. It may, however, be within reach to apply the simulation technique [Svensson, 1993] describes in order to ascertain that the amendment will actually lead to a reduced income of unemployed people. It might, for instance, well be that social security regulations overcompensate the increased tax.

10. Conclusion

We have presented a ladder of incremental steps in formalization and application of LKBS techniques to enhance the process of drafting legislation, each step enabling a new kind of check on the draft. In the first step, a rudimentary logical structure is added to the draft in order to improve the structure and to eradicate sentence ambiguity. Additional steps offer a different perspective for examining the draft and improving the formalization. If the drafter has taken the effort to climb the ladder so far, we offer some ideas for as to how the drafter may make use of the formalism to check some of the properties of the draft, verify the legal consequences and some macro level effects. The formalism that is proposed may be made more sophisticated, but in its current form, it may be sufficient for the checks we are suggesting. If these checks prove to be of practical benefit for drafters, we shall happily go along extending the ladder. However, we believe in the need for the strategy of tempting the drafters to climb the ladder by presenting some extra benefit for every step they climb, rather than demanding them to accept an all or nothing package of formal tools.

References

[Alchourrón & Bulygin, 1971] Alchourrón, C.E. & Bulygin, E. (1971) *Normative Systems*, New York, Springler Verlag.

[Allen & Saxon, 1985] Allen, Layman & Saxon, Charles (1985), Computer aided normalizing and unpacking: Some interesting machine-processable transformations of legal rules, in Walter, Charles (ed.), *Computing Power and Legal Reasoning*, West Publishing Company, St. Paul, Minn.

[Allen & Saxon, 1986] Allen, Layman & Saxon, Charles (1986), "Analysis of the logical structure of legal rules by a modernized and formalized version of Hohfeld fundamental legal conceptions", in Martino, A.A. & Socci Natali, F. (eds.), *Automated Analysis of Legal Texts -Logic-Informatics-Law*, Elsevier Science Publishers B.V. (North-Holland)

[Allen & Saxon, 1988] Allen, Layman & Saxon, Charles (1988), Exploring Computer-Aided Generation of Questions for Normalizing Legal Rules, in Walter, Charles (ed.), *Computing Power and Legal Language*, Quorum books, New York

[Bing, 1991] Bing, Jon (1991), *Improving regulatory management: The use of information systems,* NORIS(96)II - NRCCL Oslo.

[Breuker, 1991] Breuker, J.A. (1991) Towards a workbench for the legal practitioner. In: Noortwijk, C. van, A.H.J. Schmidt & R.G.F. Winkels (eds), *Legal Knowledge bases systems; Aims for research and development,* Lelystad, Koninklijke Vermande BV

[denHaan & Breuker, 1991] denHaan, N. & Breuker, J.A. (1991) A tractable Juridical KBS for applying and teaching traffic regulations. In: J.A Breuker, R.V. de Mulder, J.C. Hage (eds), *Legal Knowledge Based Systems: Model-based legal reasoning,* JURIX'91, Koninklijke Vermande, Lelystad, NL, 1991

[denHaan, 1992] N. den Haan (1992) TRACS: A Support Tool for Drafting and Testing Law. In: Grütters, C.A.F.M., J.A.P.J. Breuker, H.J. van den Herik, A.H.J. Schmidt and C.N.J. de Vey Mestdagh (eds), *Legal Knowledge Based Systems: Information Technology & Law, JURIX'92,* Koninklijke Vermande, Lelystad, NL, 1992

[Follesdal & Hilpinen, 1971] Follesdal, D and R. Hilpinen (1971) Deontic Logic: An Introduction, in: R. Hilpinen (ed.) *Deontic Logic: Introductory and Systematic Readings,* D. Redider Publishing Company, Dordrecht Holland

[Gray, 1985] Gray, Grayfred F. (1985), Statutes enacted in normalized form: The legislative experience in Tennessee, in Walter, Charles (ed.), *Computing Power and Legal Reasoning,* West Publishing Company, St. Paul, Minn.

[Gray, 1988] Gray, Grayfred F. (1988), *An Experiment with Normalized Statutes in an Emycin Expert System*, in Walter, Charles (ed.), Computing Power and Legal Language, Quorum books, New York.

[Jones & Pörn, 1985] Jones, A.J.I. and Porn, I. (1985) *Ideality, sub-ideality and deontic logic,* Synthese 65.

[Jones & Sergot, 1991] Jones, A.J.I. & Sergot, M. (1991) *On the Role of Deontic Logic in the Characterization of Normative Systems,* In: Ch. Meyer and R.J. Wieringa (eds.), Proceedings of the First International Workshop on Deontic Logic in Computer Science, Amsterdam, the Netherlands.

[Jones & Sergot, 1992] Jones, Andrew J.I. & Sergot, Marek (1992), *Formal Specification of Security Requirements using the Theory of Normative Positions,* Proceedings of ESORICS-92, Toulouse.

[Kanger, 1972] Kanger, Stig (1972) *Law and Logic,* Theoria, vol. 38, pp. 105-132.

[Lindahl, 1977] Lindahl, Lars (1977), *Position and Change - A Study in Law and Logic,* D. Reidel Publishing Company, Dordrecht Holland.

[Lindahl, 1991] Lindahl, Lars (1991), *Stig Kanger's Theory of Rights,* 9th Int. Congress of Logic, Methodology and Philosophy of Science, Uppsala Sverige.

[Nieuwenhuis, 1989] Nieuwenhuis, M.A. (1989) *Tessec: een expertsysteem voor de Algemene Bijstandswet,* Deventer, Kluwer

[Prakken, 1993] Prakken, H. (1993) *Logical tools for modelling legal argument,* Thesis, January 14, Vrije Universiteit, Amsterdam

[Staudt, 1993] Staudt, Ronald W. (1993), *Does the grandmother come with it? Teaching and practising law in the 21st century,* Prepublishing version june 1993, Chicago Kent Law School.

[Susskind, 1987] Susskind, Richard E. (1987), *Expert Systems in Law - A Jurisprudential Inquiry,* Clarendon Press - Oxford.

[Svensson, 1993] Svensson, J.S. (1993) *Kennisgebaseerde microsimulatie, een nieuwe methode voor het bepalen van sociaal-economische gevolgen van wet- en regelgeving in de sociale zekerheid.* Thesis, Faculteit der Bestuurskunde, Universiteit Twente, Enschede.