Legislation as Logic Programs*

Robert A. Kowalski

Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ, UK

January 1991
Revised June 1992

Abstract. The linguistic style in which legislation is normally written has many similarities with the language of logic programming. However, examples of legal language taken from the British Nationality Act 1981, the University of Michigan lease termination clause, and the London Underground emergency notice suggest several ways in which the basic model of logic programming could usefully be extended. These extensions include the introduction of types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning and procedural notation.

In addition to the resemblance between legislation and programs, the law has other important similarities with computing. It needs for example to validate legislation against social and political specifications, and it needs to organise, develop, maintain and reuse large and complex bodies of legal codes and procedures. Such parallels between computing and law suggest that it might be possible to transfer useful results and techniques in both directions between these different fields. One possibility explored in this paper is that the linguistic structures of an appropriately extended logic programming language might indicate ways in which the language of legislation itself could be made simpler and clearer.

1 Introduction

The characteristic feature of the language of legislation is that it uses natural language to express general rules, in order to regulate human affairs. To be effective for this purpose, it needs to be more precise than ordinary language and, as much as possible, it needs to be understood by different people in the same way. In this respect legislation can be viewed as programs expressed in human language to be executed by humans rather than by computers.

Thus the language of legislation might also serve as a model for computing, suggesting ways in which programming languages might be made more like human languages, while still remaining machine executable. In this paper I shall focus on a comparison between the language of legislation and the language of logic programming. I shall argue that, although logic programming fares well in this comparison, it needs to be improved by incorporating such extensions as types, relative clauses, both ordinary negation and negation by failure, integrity constraints,

---

*

metalevel reasoning, and procedural notation. I shall also argue that in some cases legislation itself can be improved by re-expressing it in a style more closely resembling such an extended logic programming form.

I shall investigate three examples. The first consists of several sections from the British Nationality Act 1981; the second is the University of Michigan lease termination clause; and the third is the London underground emergency notice. The first example was investigated earlier by the author and his colleagues [24] as an illustration of the use of logic programming for representing legislation. The second was investigated by Allen and Saxon [1] as an example of the use of logic to eliminate ambiguities in the formulation of a legal contract. The third was identified by the author [13] as an example of a public notice which is meant not only to be precise but also to be as clear as possible to ordinary people.

In our earlier investigation of the British Nationality Act 1981 [10] we emphasized both the prospects of using logic programming to build legal applications as well as the problems of attempting to use logic programming for knowledge representation. In this paper I am concerned only with the second of these matters, but more specifically with investigating linguistic similarities and differences between logic programming and legislation, and more generally with exploring other parallels between computing and the law.

2 The British Nationality Act 1981

The following four examples from the British Nationality Act illustrate some of the complexity and precision of legal language. They also illustrate the treatment of time, default reasoning, negative conclusions and reasoning about belief.

2.1 Acquisition by Birth

The first subsection of the British Nationality Act deals with the case of acquisition of citizenship by virtue of birth in the United Kingdom after commencement (1 January 1983, the date on which the Act took affect).

> 1.-(1) A person born in the United Kingdom after commencement shall be a British citizen if at the time of the birth his father or mother is -
> (a) a British citizen; or
> (b) settled in the United Kingdom.

The English of this clause is already close to logic programming form, even to the extent of expressing the conclusion before (most of) the conditions. Using infix notation for predicates and upper case letters for variables, 1.1 can be paraphrased in logic programming form by:

> X acquires british citizenship by section 1.1
> if X is born in the uk at T
> and T is after commencement
> and Y is parent if X
> and Y is a british citizen at T or
> Y is settled in the uk at T

This has the propositional form

A if [B and C and D and [E or F]]

which is equivalent to two rules

A if B and C and D and E
A if B and C and D and F

in normal logic programming form.

In this paper I shall use the term logic program to refer to any set of sentences which are equivalent to a set of universally quantified implications in the normal logic programming form

A if B1 and ... and Bn

where A is an atomic formula, Bi for $0 < i < n$ is an atomic formula or the negation of an atomic formula, and all variables, e.g. X1, ..., Xm occurring in the implication are assumed to be universally quantified, i.e.

for all X1, ..., Xm [A if B1 and ... and Bn].

The logic programming representation of 1.1 can be made more like the English, while remaining formal, by introducing types and various forms of syntactic sugar. For example:

a person who is born in the uk at a time
which is after commencement
acquires british citizenship by section 1.1
if a parent of the person is a british citizen at the time,
or a parent of the person is settled in the uk at the time.

Here "person" and "time" are type identifiers; "a person" is the first occurrence of a variable of type "person"; "a time" is the first occurrence of a variable of type "time"; "the person" and "the time" stand for later occurrences of the same variables. The relative pronouns "who" and "which" also stand for additional occurrences of the variables they follow. "who" stands for an occurrence of type "person", whereas "which" stands for an occurrence of any type of variable. Relative clauses in expressions of the form

... V which P ***

for example, are syntactic sugar for

... V *** if V P

where "V" is a variable, and "P" is a predicate which applies to "V".
Similarly an expression of the form

... a R of T P ***

is syntactic sugar for

... V P *** if V R of T

where "R of" is a binary predicate, "T" is a term, and "V" is any variable not occurring elsewhere in the sentence.

Notice that the two transformations described above need to be combined with the simplication of formulae of the form

(A if B) if C

to the form

A if B and C

This kind of typing and syntactic sugar can be defined more precisely and can be extended to deal with several variables of the same type, pronouns, and more flexible kinds of relative clauses. In this way English can serve as a model to improve the naturalness of logic programming without sacrificing its precision.

I shall argue elsewhere in this paper that, conversely, the use of conclusion-conditions form, which characterises the syntax of logic programming, can sometimes improve the clarity of natural languages such as English.

2.2 Representation of Time

In the representation of 1.1 time has been represented by an explicit parameter of type "time". The expression

... after ***

is interpreted as short-hand for

... at a time which is after ***

i.e.

... at T if T is after ***.

This explicit representation of time contrasts with modal representations, where temporal relationships are represented by modal operators, and time itself is implicit rather than explicit.

As mentioned for example in [11], to reason about time, an explicit axiom of persistence can be formulated to express that

a property holds at a time which is after another time
if an event occurs at the other time
and the event initiates the property

and it is not the case that
another event occurs at yet another time
which is between the time and the other time
and the other event terminates the property.

"a person acquires british citizenship by section 1.1" initiates
"the person is a british citizen".

Perhaps this is an example where symbolic notation with explicit representation of variables might be easier for some people to follow. Here "a time", "another time", and "yet another time" introduce different variables of the same type "time". Notice that the English suggests that the variables refer to distinct individuals, whereas the usual logical convention is that different variables of the same type can refer to the same individual. This is one of several discrepancies which would need to be attended to in a more systematic study of the correspondence between logic and a precise style of English.

Notice also in the two axioms above how events and properties are treated metalogically as names of sentences.

2.3 Abandoned Children and Default Reasoning

The second subsection of the British Nationality Act is conceptually one of the most complex sentences of the Act.

1.-(2) A new-born infant who, after commencement, is found abandoned in the United Kingdom shall, unless the contrary is shown, be deemed for the purposes of subsection (1)-

(a) to have been born in the United Kingdom after commencement; and

(b) to have been born to a parent who at the time of the birth was a British citizen or settled in the United Kingdom.

Under the procedural interpretation of logic programs, conclusions of sentences are interpreted as goals and conditions as subgoals. According to this interpretation, the conclusion of a sentence identifies its purpose. Thus we can interpret the phrase "the purposes of subsection (1)" as a metalevel reference to the logical conclusion of 1.1, namely to acquire British citizenship. Moreover the object level phrases 1.2.a and 1.2.b are exactly the logical conditions of 1.1. Thus we can regard the entire sentence 1.2 as a mixed object level and metalevel sentence which expresses that

the conditions of 1.1 shall be assumed to hold for a person
if the person is found newborn abandoned in the uk
at a time which is after commencement.
and the contrary of the conditions of 1.1 are not shown

This can be reformulated at the object level alone by replacing the metalevel descriptions by their object level counterparts:

a person who is found newborn abandoned in the uk at a time
which is after commencement
acquires british citizenship by section 1.2
if it is not shown that it is not the case that
the person is born in the uk at a time
which is after commencement
and either it is not shown that it is not the case that
a parent of the person is a british citizen at the time of birth
or it is not shown that it is not the case that
a parent of the person is settled in the uk
at the time of birth

This seems to be a case where the mixed object-level meta-level expression may be easier to understand than the purely object level representation.

Conditions of the form

it is not shown that it is not the case that P

in the object level sentence above, can be interpreted as combining negation as failure "not" and ordinary negation "¬", i.e.

not ¬ P.

Thus, for another example, the statements

A bird flies if it is not shown that it is not the case that the bird flies.
It is not the case that an ostrich flies.

can be formalised by

a bird flies if not ¬ the bird flies
¬ an ostrich flies

Just such an extension of logic programming to include both negation by failure and ordinary negation has been investigated by Gelfond and Lifschitz [8] and by Kowalski and Sadri [14].

Negation by failure is a form of default reasoning and is non-monotonic. Thus a person who acquires citizenship by 1.2 might non-monotonically have citizenship withdrawn in the light of new information. It is unlikely, however, that parliament intended that citizenship be withdrawn in this way. Both such an intention and the opposite intention can be catered for by introducing an extra layer of time concerned with the time for which beliefs are held in addition to the historical time for which properties hold true in the world. A logic programming approach to such a joint representation of belief time and historical time has been developed by Sripada [25].

It is important to emphasize that when formalising legislation as (extended) logic programs we do not attempt to define concepts which occur in conditions of the legislation but are not defined in the legislation itself. Thus, for example, we do not attempt to define the concept "new born infant" which occurs in the conditions of 1.2.

This means, as a consequence, that a formalisation of the British Nationality Act has only limited applicability by itself. To be used in a particular case it would need to be supplemented, if not by a set of definitions of such vague terms, at least by a set of facts or assumptions which express judgements about whether or not such terms apply to the case in hand.

2.4 Deprivation of Citizenship and Negative Conclusions

Except for its occurrence in conditions of the form

not ¬ P

ordinary negation ¬ seems to be needed only in the conclusions of rules. In such cases, a negative conclusion typically expresses an exception to a general rule, as in the example

It is not the case that an ostrich flies.

which expresses an exception to the general rule that all birds fly.

Exceptions, expressed by sentences with negative conclusions, are common in legislation [12]. The provisions for depriving British citizens of their citizenship exemplify this use of negation:

> 40.-(1) Subject to the provisions of this section, the Secretary of State may by order deprive any British citizen to whom this subsection applies of his British citizenship if the Secretary of State is satisfied that the registration or certificate of naturalisation by virtue of which he is such a citizen was obtained by means of fraud, false representation or the concealment of any material fact.
>
> 40.-(5) The Secretary of State -
> (a) shall not deprive a person of British citizenship under this section unless he is satisfied that it is not conducive to the public good that that person should continue to be a British citizen; ...

40.1 has the logical form

P if Q

whereas 40.5 has the form

¬ P if not R

If both conditions Q and not R hold, then by ordinary logic it would be possible to

deduce a contradiction

P and ¬ P.

But this is not the intention of the legislation, which is rather that the exception should override the rule, or equivalently that the rule should be understood as having an extra, implicit condition.

P if Q and not ¬P.

In fact, the metalevel phrase "subject to the provisions of this section" at the beginning of 40.1 can be regarded as a caution that the meaning of 40.1 cannot be understood in isolation of the rest of the section as a whole.

The extension of logic programming to allow negative conclusions, for the purpose of representing exceptions, has been investigated by Kowalski and Sadri [14]. They also show that such extended logic programs can be transformed into normal logic programs. In particular a rule with a single exception

P if Q
¬ P if not R

can be transformed into the simpler rule

P if Q and R.

Both representations can be useful for different purposes. A representation in terms of rules and exceptions is often easier to develop and to maintain. However, the simpler representation as normal logic programs is usually clearer and easier to understand. The first representation, accordingly, might be preferred by a draftsman, who codifies the law; the second might be preferred by an administrator who executes the law.

In this discussion of the provisions for deprivation of citizenship we have considered only the propositional structure of the English sentences. We have not considered the meaning of such conditions as

"he is satisfied that it is not conducive
to the public good that that person
should continue to be a British citizen".

This is partly because it would be very difficult to do so; but also because we have restricted our attention to representing formally only what is defined explicitly in the legislation itself. Nonetheless, reasoning about reasoning can, at least to some extent, be formalised by metalogic or by logics of knowledge and belief.

2.5 Naturalisation and the Representation of Belief

Like the provisions for deprivation of citizenship, the provisions for naturalisation contain conditions concerning the Secretary of State's beliefs. In addition, however, they also contain rules governing the subject matter of those beliefs. This leads us to consider whether we can establish a logical connection between the two.

Section 6.1 contains the main provision for naturalisation:

6.-(1) If, on an application for naturalisation as a British

citizen made by a person of full age and capacity, the Secretary of State is satisfied that the applicant fulfills the requirements of Schedule 1 for naturalisation as such a citizen under this sub-section, he may, if he thinks fit, grant to him a certificate of naturalisation as such a citizen.

At the propositional level this is equivalent to a sentence in conclusion-conditions form:

the secretary of state may grant a certificate of naturalisation to a person by section 6.1
- if the person applies for naturalisation
- and the person is of full age and capacity
- and the secretary of state is satisfied that the person fulfills the requirements of schedule 1 for naturalisation by 6.1
- and the secretary of state thinks fit to grant the person a certificate of naturalisation.

The last two conditions vest considerable powers of discretion in the Secretary of State. The last condition is totally inscrutable and can only be given as part of the input for a given case. But the meaning of the penultimate condition ought at least to be constrained by the meaning of Schedule 1. This schedule is quite long, and it is convenient therefore to summarise its contents:

a person fulfills the requirements of schedule 1 for naturalisation by 6.1
- if either the person fulfills residency requirements specified in subparagraph 1.1.2
  - or the person fulfills crown service requirements specified in subparagraph 1.1.3
- and the person is of good character
- and the person has sufficient knowledge of english, welsh, or scottish gaelic
- and either the person intends to reside in the uk in the event of being granted naturalisation
  - or the person intends to enter or continue in crown service in the event of being granted naturalisation.

To understand the connection between 6.1 and Schedule 1, it is necessary to understand the connection between meeting the requirements for naturalisation specified in Schedule 1 and satisfying the Secretary of State that those requirements are met. Fortunately, this can be done, at least in part, by regarding satisfaction as a kind of belief. The appropriate rules of belief can be formalised in both modal logic and metalogic. The following formalisation in metalogic has the form of a metainterpreter.

a person is satisfied that P
- if the person is satisfied that $P \rightarrow Q$
- and the person is satisfied that Q

a person is satisfied that P ⌉ Q
    if the person is satisfied that P
    and the person is satisfied that Q

a person is satisfied that P v Q
    if the person is satisfied that P
    or the person is satisfied that Q

Here "→", "⌉", and "v" are infix function symbols naming implication, conjunction, and disjunction respectively.

We may safely assume that

the secretary of state is satisfied that P
    if P is a representation of the meaning
        of a provision of the british nationality act 1981

Thus the Secretary of State is satisfied in particular that the implication which represents the meaning of Schedule 1 holds. This assumption and the metainterpreters above are all we need to establish a logical connection between 6.1 and Schedule 1. This connection can be made more explicit, however, if we transform the metainterpreter using the technique of partial evaluation [7, 26]:

the secretary of state is satisfied that a person fulfills
the requirements for naturalisation by 6.1
if either the secretary of state is satisfied that
        the person fulfills residency requirements specified in
        paragraph 1.1.2
    or the secretary of state is satisfied that
        the person fulfills crown service requirements specified in
        paragraph 1.1.3
and the secretary of state is satisfied that
    the person is of good character
and the secretary of state is satisfied that
    the person has sufficient knowledge of english, welsh, or scottish
    gaelic
and either the secretary of state is satisfied that
        the person intends to reside in the uk in
        the event of being granted naturalisation
    or the secretary of state is satisfied that
        the person intends to enter or continue in
        crown service in the event of being granted naturalisation.

The result is an explicit, though somewhat tedious, statement of what it means to satisfy the Secretary of State concerning the requirements for naturalisation. Clearly the statement could be made a little less tedious if we used a pronoun, "he" or "she" for all references to the Secretary of State after the first.

The language of the British Nationality Act 1981 is for the most part extraordinarily precise. It is also very complex. Most of this complexity is inherent in the meaning of the Act. However, some of the complexity can be reduced by the explicit use of conclusion-conditions form and by the use of meaning-preserving transformations of the kind illustrated in the last two examples.

By comparison with ordinary language and even with legal language in general, the Act is also surprisingly unambiguous. However, as we have already seen, it does contain vague terms and undefined concepts. Such vagueness is often confused with ambiguity. Although, like genuine ambiguity, vagueness causes problems of interpretation, it is also useful, because it allows the law to evolve and adapt to changing circumstances.

Genuine ambiguity, on the other hand, generally serves no useful purpose. Moreover, whereas logic can easily accommodate vagueness, it cannot tolerate ambiguity.

The University of Michigan lease termination clause, presented in the next section, was originally investigated by Allen and Saxon [1] to illustrate the use of propositional logic to formulate a precise interpretation of an ambiguous legal text. I shall argue that the use of logic programming conclusion-conditions form has the further advantage of rendering many of the possible interpretations logically implausible.

3 The University of Michigan Lease Termination Clause

The clause consists of a single, long sentence which has the underlying, logically ambiguous form

> A if A1 and A2 or A3 or A4 or A5 or A6 or A7
> unless B1 or B2 or B3 or B4 or B5 in which cases B.

Different ways of introducing parentheses produce different interpretations. Some of these are logically equivalent because of the associativity of "or", for example. After accounting for these equivalences, Allen and Saxon identify approximately 80 questions that might need to be asked in order to distinguish between the different parenthesizations. As a result of this analysis they identify one intended interpretation which has the form

> ((A if (A1 and(A2 or A3)) or A4 or A5 or A6 or A7)
> if not (B1 or B2 or B3 or B4 or B5)) and
> (if (B1 or B2 or B3 or B4 or B5) then B)

where "unless" has been translated as "if not". It is interesting that this interpretation has a logic programming form.

The logic programming representation can be simplified if, as Allen and Saxon maintain, conditions B1-B5 are the only ones under which conclusion B holds. In that case the conditions not(B1 or B2 or B3 or B4 or B5) can be replaced by not B. Thus the intended interpretation can be represented by the simplified, normal logic program:

A if A1 and A2 and not B
A if A1 and A3 and not B
A if A4 and not B
A if A5 and not B
A if A6 and not B
A if A7 and not B
B if B1
B if B2
B if B3
B if B4
B if B5

This logical analysis of the propositional structure of the sentence should be compared with the English text of the sentence:

> "The University may terminate this lease when the Lessee, having made application and executed this lease in advance of enrollment, is not eligible to enroll or fails to enroll in the University or leaves the University at any time prior to the expiration of this lease, or for violation of any provisions of this lease, or for violation of any University regulation relative to Resident Halls, or for health reasons, by providing the student with written notice of this termination 30 days prior to the effective time of termination; unless life, limb, or property would be jeapordized, the Lessee engages in the sales or purchase of controlled substances in violation of federal, state or local law, or the Lessee is no longer enrolled as a student, or the Lessee engages in the use or possession of firearms, explosives, inflammable liquids, fireworks, or other dangerous weapons within the building, or turns in a false alarm, in which cases a maximum of 24 hours notice would be sufficient".

Notice how the conclusion A of the first half of the sentence is split into two parts by the insertion of the conditions A1-A7. Notice also that the language of the sentence is so complicated and so confused that the drafters mistakenly wrote "maximum of 24 hours" when they must have meant "minimum of 24 hours".

In fact I have slightly misrepresented Allen and Saxon's analysis of the sentence. In addition to identifying the intended placement of parentheses, they analyse for each of the three occurrences of "if" in the apparent meaning of the sentence whether or not "if and only if" is really intended. They conclude that in the first two cases (of the words "when" and "unless") it is not intended, whereas, in the third case (of the words "in which cases") it is. Thus their real analysis of the intended interpretation has the form

((A if (A1 and (A2 or A3)) or A4 or A5 or A6 or A7)
if not (B1 or B2 or B3 or B4 or B5)) and
(if (B1 or B2 or B3 or B4 or B5) then B) and
(if not (B1 or B2 or B3 or B4 or B5) then not B).

In contrast, with this change of representation using ordinary logic, the logic programming representation is not affected by this change of interpretation. In the logic program there is no difference between the representation of "if" and the

representation of "if and only if". The difference between the two interpretations depends upon whether or not the "closed world assumption" [6] is applied. The closed world assumption for a predicate P is the assumption that all the implications

P if Q1
P if Q2
:
P if Qn

with conclusion P in a program represent all the conditions under which conclusion P holds. It is this assumption that justifies the negation as failure rule:

not P holds if P fails to hold, i.e.
not P holds if all ways of trying to show P result in failure.

Thus, in the example of the lease termination clause, in the case of the word "when", the interpretation "if and only if" is not intended because there are other situations referred to elsewhere in the lease under which the University may terminate the lease with 30 days written notice. But in the case of the words "in which case", the interpretation "if and only if" is intended because there are no other cases under which the University may terminate the lease with 24 hours notice. In the case of the word "unless", the question is not relevant because in the context in which it occurs the closed world assumption is not applicable.

Allen and Saxon argue that the logical representation of the lease termination clause does not express what the drafters must have actually intended. After all the ambiguities have been resolved, the English text expresses that for the University to be able to terminate the lease with 30 days written notice, not only must one of the conditions

(A1 and (A2 or A3)) or A4 or A5 or A6 or A7

hold but none of the conditions

B1 or B2 or B3 or B4 or B5,

under which it may terminate the lease with 24 hours notice, may hold. But these extra negative conditions play no useful role. They serve only to make the conditions under which conclusion holds exclusive of the conditions under which conclusion B holds.

The simpler rules

A if ((A1 and (A2 or A3) or A4 or A5 or A6 or A7)
B if (B1 or B2 or B3 or B4 or B5)

are more flexible. Compared with the original rules they give the university the extra option of giving students 30 days notice in cases where they would otherwise be forced to give 24 hour notice.

Using indentation, and the expressions "both ... and", and "either ... or" in place of parentheses, this new interpretation can be written in a form which arguably has both the precision and simplicity of logic programming and the naturalness of English:

> The university may terminate this lease by providing the lessee with written notice of the termination 30 days prior to the effective time of termination
>
> if both the lessee has applied for and executed
> this lease in advance of enrollment
> and either the lessee is not eligible to enroll
> or the lessee fails to enroll
> or the lessee leaves the university at any
> time prior to the expiration of this lease
> or the lessee violates any provisions of this lease
> or the lessee violates any university regulations
> relative to residence halls
> or there are health reasons for terminating this lease.
>
> The university may terminate this lease by providing the lessee with notice of the termination a minimum of 24 hours prior to the effective time of termination
>
> if life, limb or property would be jeopardized by
> continuation of the lease
> or the lessee engages in the sale or purchase of
> controlled substances in violation of federal, state or local law
> or the lessee is no longer enrolled as a student
> or the lessee engages in the use or possession of
> firearms, explosives, inflammable liquids,
> fireworks, or other dangerous weapons within the building
> or the lessee turns in a false fire alarm.

The University of Michigan lease termination clause is not a good illustration of our thesis that legal language can be a good guide for improving computer languages. If anything, it seems to suggest the converse, that some computer languages might be a useful guide for improving the language of the law.

In fact, few legal documents are written to the standards of precision found in the acts of parliament; and hardly any legal documents at all are written not only to be precise but also to be clear and easy to understand. However, public notices, which are meant to be understood by ordinary people, are for the most part an important exception to this rule. The London underground emergency notice is a good example of such an exception.

4 The London Underground Emergency Notice

The notice has many characteristics of a logic program, but with some interesting differences:

> **EMERGENCIES**
>
> Press the alarm signal button
> to alert the driver.

The driver will stop immediately
if any part of the train is in a station.

If not, the train will continue to the next station,
where help can more easily be given.

There is a £50 penalty
for improper use.

From a knowledge representation point of view, the first sentence is probably the most interesting. Expressed in a procedural style, it shows that a procedural form of expression can sometimes be more appropriate than an "equivalent" statement in declarative style:

You alert the driver
if You press the alarm signal button.

Notice, however, that the procedural form can be regarded as a compiled version of the procedural interpretation of the declarative form. Like most compiled representations of knowledge, it requires less work on the part of the recipient to put the knowledge into effect.

This example and others like it suggest that logic programming could be made more like natural language if it allowed both declarative and procedural syntax. Under the procedural interpretation of logic programming, both the declarative syntax

A if B and C

and the procedural syntax

to do A do B and do C

would be equivalent. In fact both styles of expression would have the same declarative meaning

A if B and C

and the same procedural meaning

to do A do B and do C.

A procedural syntax for logic programs would not, however, include arbitrary imperative programming language constructs. It would not, for example, without further extension, include such purely imperative statements as

press the alarm signal button.

All imperative statements in a logic programming language would have to be imbedded in a procedure, which contains an expression of its purpose. I shall discuss

the possible extension of logic programs to include purposeless procedures, viewed as integrity constraints, in sections 5.1 and 5.2.

To simplify the discussion of the emergency notice, I have ignored and, for the most part, will continue to ignore the temporal relationships between the different actions and situations referred to in the notice. We should note however, that to be accurate the title of the notice should be incorporated into the conclusion of the sentence:

> press the alarm signal button,
> to alert the driver to an emergency.

The second sentence of the notice is explicitly expressed in a logic programming form. However, even allowing for the fact that the phrase

> the driver will stop immediately

is shorthand for

> the driver will stop the train immediately,

the sentence fails to express its intended meaning, because it is missing an entire condition. The meaning of the sentence can be made explicit, by supplying the missing condition from the conclusion of the previous sentence:

> the driver will stop the train immediately
> if You alert the driver to an emergency
> and any part of the train is in a station.

Certainly this precise expression of the meaning of the sentence is more cumbersome that the English. However, it is hard to see how the logic programming representation could be simplified so that it more closely resembles the English, without loosing its precision.

The third sentence begins with an allusion to the explicitly stated condition of the previous sentence. Ignoring for the moment, the comment at the end, the sentence with all its conditions made fully explicit has the logical form:

> the train will continue to the next station
> if You alert the driver to an emergency
> and not any part of the train is in a station.

But this alone cannot be all that it is intended by the English, because the train will generally continue to the next station whether or not the driver is alerted to an emergency. Surely, what is meant is that the train will stop at the next station and that help will be given there. This is part of the meaning of the phrase

> where help can more easily be given.

Moreover, presumably help will be given at a station whether it is the next station or not. Thus we can obtain a better approximation to the intended meaning of the third sentence with the two sentences:

the train will stop at the next station
if You alert the driver to an emergency
and not any part of the train is in a station.

help will be given in an emergency
if You alert the driver to the emergency
and the train is stopped in a station.

This second sentence of the revised formulation of the sentence captures part of the meaning of the comment at the end of the sentence. Presumably the rest of its meaning could be expressed by the meta statement that this procedure for getting help is better than the alternative procedure of stopping the train when it is not in a station.

The last sentence of the notice has a simple formulation in conclusion-conditions form:

there is a £50 penalty
if You use the alarm signal button improperly.

This contrasts with a purely imperative statement, which expresses a prohibition without expressing a purpose:

do not use the alarm signal button improperly.

In contrast with the purely imperative statement of prohibition, the procedural interpretation of the English sentence contains a clear expression of purpose:

if You want a £50 penalty,
then press the alarm signal button improperly!

Notice, by the way, how different the procedural syntax of a sentence can be from its declarative meaning. The English procedural sentence

if You want A, then do B

actually has the underlying declarative meaning

A if B.

Although the English of the London underground notice can be improved, it is undoubtably clear and easy to understand. I believe its clarity is due to at least three characteristics

- the explicit use of conclusion-conditions form
- the appropriate use of procedural form, and
- the use of ellipsis to avoid unnecessarily stating the obvious.

The first two characteristics can usefully be applied to the design and improvement of computer languages today. The third characteristic is harder to achieve, although some progress along these lines might be possible in the future.

5 Other Computing Paradigms

The preceding examples illustrate some of the typical characteristics of legal language and its relationship to logic programming form. It is also possible, however, to find indications of other computing paradigms.

5.1 Condition-Action Production Rules

Condition-action rules were developed by Newell and Simon [19] as a model of human psychology and have been used to implement expert systems [27]. They can also be found in the language of public notices. For example, the following notice is displayed in the carriages of the London underground

> Please give up this seat
> if an elderly or handicapped person needs it

This is a distinct improvement over the earlier, ambiguous, and potentially disturbing notice

> please give up this seat
> to an elderly or handicapped person.

But even with the explicit use of the word "if", the sentence falls short of logic programming form, because the apparent conclusion

> please give up this seat

is imperative rather than declarative. Moreover the sentence does not express a purpose.

The condition-action form in which the rule is expressed can be converted into logic programming form by making the purpose, e.g.

> to do a good deed

explicit rather than implicit. The resulting statement can be expressed procedurally

> to do a good deed
> give up this seat
> if an elderly or handicapped person needs it

or declaratively

> You do a good deed
> if You give up Your seat to a person
> who needs Your seat and
> who is elderly or handicapped.

The claim that every command has an explicit or implicit purpose is an important theory in legal philosophy. The use of logic programming form, which forces purposes to be made explicit, is in the spirit of this theory. Associating explicit purposes with commands makes it possible to reason about the relative merits of conflicting commands and even to reason whether a command is appropriate in a given context.

Nonetheless, natural language does allow the expression of commands without purpose, and there even seems to be a logic programming analogue of this in the form of integrity constraints.

5.3 Integrity Constraints

For many years the London underground displayed the following notice above the automatic doors of its carriages

> Obstructing the doors causes
> delay and can be dangerous.

In other words

> there will be a delay
> if You obstruct the doors.
>
> there can be danger
> if You obstruct the doors.

As long as delay and danger are regarded as undesirable, a thinking person will conclude that obstructing the doors is undesirable too.

But the London underground authorities have recently changed the wording of the notice on some of its trains. The new sign reads

> Do not obstruct the doors.

A sad reflection of our changing times. Either delay and danger are no longer regarded as undesirable, or the public cannot be relied upon to reason about the consequences of its behaviour.

But for a logic programmer the new notice is worrying, not only because it indicates the possibly deteriorating state of British underground society, but also because it represents a move away from a logic programming style of communication to a more imperative style. But on closer consideration, the change of wording is reminiscent of recent efforts to extend logic programming by the inclusion of integrity constraints.

This extension is motivated by database applications of logic programming. For these applications, a number of studies [5, 17, 21, 22] have investigated the nature of integrity constraints in logic programming and the development of efficient integrity checking methods. In all of these approaches integrity constraints are viewed as properties which a database or program must satisfy as it changes over the course of

time. To the extent that the contents of a database describe states of affairs in the world, commands, which impose obligations or prohibitions on states of the world, can be interpreted as integrity constraints on states of the database.

An integrity constraint can be expressed in the form of any sentence of first-order logic including a denial. Thus the command

do not obstruct the doors

might be represented by a denial

not You obstruct the doors

which expresses an integrity constraint on descriptions of events which take place in the world.

Similarly the condition-action rule

please give up this seat
if an elderly or handicapped person needs it

might be interpreted as an integrity constraint which has the form of an implication

You give up a seat to a person
if You are sitting in the seat
and the person needs Your seat
and the person is elderly or handicapped.

Thus, given a database that records events that take place in the world, the integrity of the database will be violated if the database records that a person is sitting in a seat which an elderly or handicapped person needs and the database does not contain a record of that person giving up the seat to the elderly or handicapped person. It is another problem, if integrity has been violated, to decide how integrity should be restored. Perhaps this is where "purpose" or "sanctions" might play a useful role.

Thus commands without purpose seem to be compatible with logic programs extended by the inclusion of integrity constraints. Moreover, there is even a transformation between integrity constraints and logic program rules, which is analogous to a transformation between commands without purpose and procedures with purpose:

Given an integrity constraint expressed as a first-order sentence

C

introduce a new predicate S and convert the constraint to the rule

S if not C

together with the new constraint

not S.

The new predicate S can be interpreted as a "sanction" which applies if the original constraint is violated. This transformation has been used in the literature on integrity constraints in deductive databases to convert arbitrary first-order integrity constraints into denial form.

The analogy between this transformation and the legal doctrine of sanctions suggest the possibility of adapting legal techniques for dealing with violations of commands to the problem of restoring integrity in deductive databases. This is an intriguing possibility that merits closer investigation.

5.3 Object-Oriented Programming

The paradigm of object-oriented programming has become increasingly important in computing in recent years. It is interesting to investigate, therefore, to what extent it has analogues in natural language and in legislative language more particularly.

We have already seen some characteristics of object-orientation in English when we saw the use of common nouns such as "person", "time" and "lessee" as a kind of object-oriented typing of variables. Other manifestations of object-orientation seem to be more difficult to find in the actual language of legislation, but easier to find both in descriptions of individual cases and in the organisation of law as a whole.

In natural language descriptions, it is common to group sentences together around a single topic placed at the beginning of each of the sentences. Such topics help to organise communication similar to the way in which objects can be used to organise knowledge in computing.

Compare, for example, the pair of sentences

> The Prime Minister stepped out of the plane.
> Journalists immediately surrounded her.

with the pair

> The Prime Minister stepped out of the plane
> She was immediately surrounded by journalists.

Psycho-linguists have found that the second pair of sentences is easier to understand than the first, despite the fact that the second pair uses the passive rather than the active voice. The two sentences in the more comprehensible pair have the same topic, whereas the two sentences in the other pair have different topics. Such examples suggest that organising knowledge around objects makes the knowledge more coherent and easier for humans to understand.

In the domain of law, it is common to organise the different areas of law into hierarchies, which are similar to hierarchies of objects. Thus a country might have one statute governing criminal law in general, another statute covering behaviour in

public places, and yet another dealing with behaviour in public buildings. Assault and battery, for example, might be prohibited everywhere, whether in public places or not. Going about naked, however, might be prohibited only in public places, but be allowed in the privacy of one's own home. Smoking, on the other hand, might be prohibited only in public buildings but be allowed everywhere else.

Thus natural language seems to support two notions of objects: objects in the small, which are used like types and topics to organise descriptions of individuals; and objects in the large, which are used in hierarchies to organise whole areas of knowledge. From this point of view, logic programming and object-orientation correspond to different aspects of natural language and are complementary.

However, the notion of object in computing has other characteristics, such as change of state, which do not have such obvious counterparts in natural language. These characteristics seem to be more closely associated with simulating the behaviour of objects in the world than with describing their behaviour.

There have been several attempts to apply object-orientation to legal reasoning. Some of these, like Gordon's Oblog [9], are based on a view of objects as types and topics, which is entirely compatible both with logic programming and with the representation of natural language meanings. Others, like the treatment of patent law by Nitta et al [20] are based on the use of objects to simulate behaviour.

The use of objects for simulation in the patent law example is especially interesting because of the way in which patent procedures, obligations and prohibitions are used to generate and filter changing states of the simulation of a patent application. It seems possible that, if the changing states of the simulation are viewed as database states, then the obligations and prohibitions expressed in the patent law might be viewed as integrity constraints. This possibility would establish an interesting link between imperative statements in object-oriented programming and integrity constraints in deductive databases and logic programming.

No matter what the outcome of a more detailed investigation of these possibilities, there can be little doubt that legislation provides a rich domain outside computing science itself within which relationships between different computing paradigms can be studied. These studies need not be confined to programming languages alone, but could usefully be extended to many other aspects of computing.

6 Other Relationships Between Computing and Law

To the extent that we can truly regard legislation as programs to be executed by people, we can also expect to find analogues in the law of such other computing matters as program specification and software management.

6.1 An Analogy Between Specifications and Policies

In the same way that programs are written to meet specifications, laws are drafted to achieve policies, which are social or political objectives. The purpose of the British Nationality Act 1981, for example, was "to make fresh provisions about citizenship and nationality, and to amend the Immigration Act 1971 as regards the right of abode in the United Kingdom", and in particular to restrict immigration to the United

Kingdom by residents of the former British colonies. The purposes of the University of Michigan lease termination clause presumably include such goals as discouraging unsociable behaviour in the halls of residence, restricting residency to legitimate students, and not causing undue hardship for individuals who are obliged to terminate their residence. The rules for dealing with London Underground emergencies, on the other hand, are designed to facilitate the provision of help as effectively and quickly as possible in the case of genuine emergencies and to avoid inconvenience and unnecessary trouble in the case of false alarms.

Program specifications have many characteristics in common with the policies of legal documents. In the same way, for example, that the primary obligation of a program might be to meet its specification, the primary duty of a legal document should be to achieve its social and political objectives. In both cases, moreover, specifications and policies are often ill-defined, inconsistent, or the result of compromise between conflicting demands.

The formal methods developed in computing to verify that programs meet their specifications are much more advanced than any corresponding methods developed for the law. A pilot study of the possibility of adapting formal methods of logic-based software verification to the problem of verifying social security regulations has been made by Bench-Capon [2].

Thus the transfer of techniques for program verification is one area in which the field of law might be able to benefit from its similarities with computing. In other areas, such as software management, the benefits might apply more equally to both fields.

6.2 An Analogy Between Software Maintenance and Maintenance of the Law

In the same way that programs need to be modified to meet changing specifications, legislation needs to be modified to meet changing social and political needs. But programs are both difficult to construct and difficult to change. So much so in fact, that programs are often still in use long after they have outlived their specifications.

The situation is not much better in the law, where legislation often lags far behind social and political changes. Obsolete and incorrect legislation is enforced simply for the sake of "law and order".

But the drafters of legislation have developed some ingenious devices for adapting, modifying and revising old legislation. The liberal use of vague and undefined terms such as "good character", "life, limb or property would be jeopardized" and "improper use" greatly contribute to the flexibility of legislation and to its ability to adapt to change. Such use of vague terms is reminiscent of the use of data abstraction and encapsulation in computer programming, which allow the lower levels of a program to change, while leaving the higher levels intact.

Much legislation is explicitly concerned with the repeal or amendment of earlier legislation. The British Nationality Act 1981, for example, repeals the British Nationality Acts 1948 to 1965 and amends the Immigration Act 1971. Amendments in particular are typically expressed by metalevel statements which describe how an old piece of text should be edited to create a new text. Metalevel statements are also used to create a new provision from a similar provision in the same act.

Section 6.2 of the British Nationality Act 1981, for example, makes special provision for naturalisation of people who are married to British citizens. The requirements are similar to those for people who apply under section 6.1, but include shorter residency requirements, omit the requirement of having sufficient knowledge of English, Welsh, or Scottish Gaelic, and include

> "the requirement specified in paragraph 1(1)(b)".

This metalevel reference to 1(1)(b) is in fact a reference to the requirement

> "that he is of good character".

This particular use of metalanguage is rather unusual in that the English expression of the metalinguistic form is actually longer than the equivalent object level expression. Usually the metalinguistic formulation is more concise than the object level formulation.

Thus the source code of legislation often mixes object level statements about the domain of discourse with metalevel statements about the text of other legislation or other provisions in the same legislation. The principle objective of using such metalevel statements in preference to equivalent object level statements is to make explicit the relationship between different but similar texts.

The language of legislation also employs remarkable techniques for reusing previous legislation. In the British Nationality Act 1981, for example, it states that one of the conditions for being a British citizen by descent under the 1981 Act is to be a person who

> under any provision of the British Nationality Acts 1948 to 1965,
> was deemed for the purposes of the proviso to section 5(1) of the 1948 Act to be a citizen of the United Kingdom and Colonies by descent only,
> or would have been so deemed if male.

The last phrase is an example of a counterfactual condition. A metalogical interpretation of such counterfactuals has been proposed by Bench-Capon [3]. It is possible to imagine how metaprogramming might be used to implement such counterfactual reuse of software in a logic programming environment.

6.3 The Relationship Between Case-Based and Rule-Based Reasoning

In artificial intelligence a contrast is sometimes made between case-based and rule-based reasoning, and a conflict is often held to exist between these two kinds of reasoning [23]. People, it is argued, reason by means of analogies between different cases rather than by means of the deductive application of rules.

The distinction between these two kinds of reasoning also lies at the heart of law. To some extent it is even reflected among the distinguishing features of the two main western legal traditions. Common law systems, such as those in England and the United States, place greater emphasis on reasoning by means of cases. Civil law systems, such as those on the continent of Europe, place greater emphasis on

reasoning by means of codified rules. In fact, in both systems of law the two kinds of reasoning interact and complement one another.

In rule-based legislation, for example, case-based reasoning plays a fundamental role in determining the meaning of vague concepts. Previous cases of a concept serve as precedents for new cases.

On the other hand, in case-based legal argumentation, the justification for a decision in a precedent setting case is often expressed in general terms and appeals to general principles. Moreover, authorative restatements of case law effectively reformulate the precedents set in individual cases into general, rule-based form, even though such case-based rules do not have the same binding force as rules in legislation. Indeed it can be argued that there is a natural evolution in the law from reasoning by means of cases to reasoning by means of rules.

7. Conclusion

The similarities between computing and the law seem to cover all areas of computing software. Moreover, the linguistic style in which legislation is drafted combines in one language the expressive power of computer languages for such diverse areas as programming, program specification, database description and query, integrity constraints, and knowledge representation in artificial intelligence. This linguistic style might be a good guide therefore to how these different areas of computing might be unified in the future.

The similarities between computing and law go beyond those of linguistic style. They extend also to the problems that the two fields share of developing, maintaining and reusing large and complex bodies of linguistic texts. Here too, it may be possible to transfer useful techniques between the two fields.

In this paper I have concentrated on similarities between logic programming and legislation. I have indicated several ways in which the language of legislation suggests that the basic model of logic programming can usefully be extended, to include types, relative clauses, both ordinary negation and negation by failure, integrity constraints, metalevel reasoning, and procedural notation. I believe that with the aid of such extensions logic programming can provide the foundations for a future, single computer language that will be suitable for all areas of computing in the same way that natural language is suitable for all areas of law.

Acknowledgement

This work was supported initially by the Science and Engineering Research Council and more recently by the ESPRIT Basic Research Action, "Computational Logic". I am especially indebted to my colleagues, Trevor Bench-Capon, Fariba Sadri and Marek Sergot, whose work on legislation and logic programming has provided much of the background for this paper.

References

[1] Allen, L. E., and Saxon, C.S. [1984] "Computer Aided Normalizing and Unpacking: Some Interesting Machine-Processable Transformation of Legal Rules",

Computing Power and Legal Reasoning (C. Walter, ed.) West Publishing Company, pp. 495-572.

[2] Bench-Capon, T.J.M. [1987]: "Support for policy makers: formulating legislation with the aid of logical models", Proc. of the First International Conference on AI and Law, ACM Press, pp. 181-189.

[3] Bench-Capon, T. [1989] "Representing Counterfactual Conditionals". Proceedings of Artificial Intelligence and the Simulation of Behaviour (A. Cohn, Ed.) Pitman Publishing Co.

[4] Bowen, K. A. and Kowalski, R. A. [1982]: "Amalgamating Language and Metalanguage in Logic Programming", in Logic Programming (Clark, K.L. and Tärnlund, S.-Å., editors), Academic Press, pp. 153-173.

[5] Bry, F., Decker, H., and Manthey, R. [1988] "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases", Proceedings of Extending Database Technology, pp. 488-505.

[6] Clark, K. L. [1978]: "negation by failure", in "Logic and databases", Gallaire, H. and Minker, J. [eds], Plenum Press, pp. 293-322.

[7] Gallagher, J. [1986] "Transforming Logic Programs by Specializing Interpreters", Proc. of 7th European Conference on Artificial Intelligence, pp. 109-122.

[8] Gelfond, M. and Lifschitz, V. [1990]: "Logic programs with classical negation", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 579-597.

[9] Gordon, T. F. [1987] "Oblog-2 a Hybrid Knowledge Representation System for Defeasible Reasoning" Proc. First International Conference on Artificial Intelligence and Law. ACM Press, pp. 231-239.

[10] H.M.S.O. [1981]: "British Nationality Act 1981", Her Majesty's Stationery Office, London.

[11] Kowalski, R. A. and Sergot, M. J. [1986]: "A logic-based calculus of events", New Generation Computing, Vol. 4, No. 1, pp. 67-95.

[12] Kowalski, R. A. [1989]: "The treatment of negation in logic programs for representing legislation", Proceedings of the Second International Conference on Artificial Intelligence and Law, pp. 11-15.

[13] Kowalski [1990] "English as a Logic Programming Language", New Generation Computing, Volume 8, pp. 91-93.

[14] Kowalski, R. A. and Sadri, F. [1990], "Logic programs with exceptions", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 598-613.

[15] Kowalski, R. A., Sergot, M. J. [1990]: "The use of logical models in legal problem solving", Ratio Juris, Vol. 3, No. 2, pp. 201-218.

[16] Lloyd, J. W. and Topor, R. W. [1984]: "Making Prolog more expressive", Journal of Logic Programming, Vol. 3, No. 1, pp. 225-240.

[17] Lloyd, J. W. and Topor, R. W. [1985] "A Basis for Deductive Database Systems", J. Logic Programming, Volume 2, Number 2, pp. 93-109.

[18] Mitchell, T. M., Keller, R. M. and Kedar-Cabelli [1986] "Explanation-based Generalization: A Unifying View" Machine Learning, Volume 1, pp. 47-80.

[19] Newell, A. and Simon, H. A. [1972] "Human problem solving", Prentice-Hall.

[20] Nitta, K., Nagao, J., and Mizutori, T., [1988] "A Knowledge Representation and Inference System for Procedural Law", New Generation Computing, pp. 319-359.

[21] Reiter, R. [1990]: "On asking what a database knows", Proc. Symposium on Computational Logic, Springer-Verlag.

[22] Sadri, F. and Kowalski, R. A. [1987]: "A theorem proving approach to database integrity", In Foundations of deductive databases and logic programming (J. Minker, editor), Morgan Kaufmann, pp. 313-362.

[23] Schank, R. C. [1983] "The current state of AI: One man's opinion", AI Magazine, Volume 4, No. 1, pp. 1-8.

[24] Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P. and Cory, H. T. [1986]: " The British Nationality Act as a logic program", CACM, Vol. 29, No. 5, pp. 370-386.

[25] Sripada, S. M. [1991] "Temporal Reasoning in Deductive Databases". Department of Computing, Imperial College, London.

[26] Takeuchi, A. and Furukawa, K. [1986] "Partial evaluation of PROLOG programs and its application to metaprogramming", Proc. of IFIP 86, North-Holland, pp. 415-420.

[27] Waterman, D. A. and Hayes-Roth [1978] "Pattern-directed Inference Systems", Academic Press, New York.