THE BRITISH NATIONALITY ACT AS A LOGIC PROGRAM

The formalization of legislation and the development of computer systems to assist with legal problem solving provide a rich domain for developing and testing artificial-intelligence technology.

M. J. SERGOT, F. SADRI, R. A. KOWALSKI, F. KRIWACZEK, P. HAMMOND, and H. T. CORY

There are essentially two kinds of law, *case law*, determined by earlier court decisions, and *statutes*, determined by legislation. Substantial amounts of statutory law are basically definitional in nature and attempt to define more or less precisely some legal relationship or concept. The British Nationality Act 1981 [20] defines British citizenship and is a good example of statutory law. The act embodies all the characteristics of statutes in general: syntactic complexity, vagueness, and reference to previously enacted legislation.

In the course of this article, we will describe how the text of a large part of the British Nationality Act 1981 was translated into a simple form of logic, and we will examine some possible applications of this translation.

The form of logic used is that on which the programming language Prolog is based. Later in the article, we will describe how our translation of the act can be executed as a program by an augmented Prolog system, so that consequences of the act can be determined mechanically.

Although Prolog logic is severely restricted, it proved to be sufficiently high level so that our implementation could resemble the style and structure of the actual text of the act. Such a resemblance is important because it helps increase confidence in the accuracy of the implementation and makes the implementation easier to maintain as the legislation changes and as case law evolves to augment the legislation.

Our implementation of the British Nationality Act 1981 was undertaken as an experiment to test the suitability of Prolog logic for expressing and applying legislation. The British Nationality Act 1981 was chosen for this experiment for a number of reasons. At the time it was first proposed, the act was a controversial piece of legislation that introduced several new classes of British citizenship. We hoped that formalization of the various definitions might illuminate some of the issues causing the controversy. More importantly, the British Nationality Act is relatively self-contained, and free, for the most part, of many complicating factors that make the problem of simulating legal reasoning so much more difficult. Furthermore, at the time of our original implementation (summer 1983) the act was free of the complicating influence of case law.

A complication that we anticipated was the presence of vagueness. The act contains such vague phrases as "being a good character," "having reasonable excuse," and "having sufficient knowledge of English." These concepts are not defined in the act and occur only at the lowest level of detail. At higher levels, the question of whether a person is a British citizen depends primarily on concrete, easily

This work was supported by the Science and Engineering Research Council.

understood concepts such as the person's time and place of birth, and the citizenship and place of settlement of the parents. In order to determine whether an individual is a British citizen, the higher level defined concepts (such as "place of settlement") are repeatedly replaced by lower level ones, until all defined concepts are eventually reduced to undefined ones. The applicability of undefined, concrete concepts can be established by obtaining data from the user or some other source. The simplest way to handle vagueness is to assume that the vague concepts always apply and to use this assumption to generate qualified answers. For example,

Peter is a citizen,
if he is of good character.

A more sophisticated approach might combine this with the use of rules of thumb that reduce vague concepts to concrete ones, but are not guaranteed to cover all cases. The rules of thumb arise from the analysis of previous cases. We deliberately avoided such complications and chose the simpler alternative in our implementation of the act. The treatment of vagueness and case law is the subject of current investigation in our group [2, 36].

In addition to vagueness, legislation is generally thought to contain both imprecision and ambiguity. We will later discuss some examples of imprecision (such as the lack of any reference to the time at which an individual actually becomes a British citizen) that came up during the course of our implementation. In fact, we found fewer such examples than we originally expected. In practice, where imprecision or ambiguity did exist, it was usually possible to identify the intended interpretation with little difficulty.

It was never our intention to develop the implementation of the act into a fully functional system. Consequently, we have subjected it to only a limited number of test applications, primarily to test the citizenship of various real and hypothetical individuals. We discovered, somewhat to our surprise, that our "declarative" representation of the act, which places primary emphasis on resemblance to the original form of the legislation and very little emphasis on efficiency, actually performed acceptably well in practice.

Data needed for an individual case are obtained interactively by using an expert-system shell, APES [18, 19], implemented in micro-Prolog [8]. APES also provides explanations when requested by the user. The quality of this interactive dialogue is sensitive to the order in which the different rules for acquiring citizenship are written, and to the order of the conditions within individual rules. Relatively little effort was put into adjusting these to improve the interaction.

In theory, mechanical theorem provers can derive arbitrary logical consequences of legislation expressed in logical form. In practice, sufficiently efficient theorem provers exist today only for certain restricted forms of logic, such as that incorporated in Prolog. Moreover, such theorem provers behave most effectively when they are restricted to determining consequences of the act for individual cases. Even with Prolog, however, we were able to derive a limited number of more general consequences of the act. This ability is potentially quite important. It means that an executable, logic-based representation of rules and regulations can be used not only to apply the rules, but to aid the process of drafting and redrafting the rules in the first place—a point that was made by Layman Allen [1] as long ago as 1957. A similar observation was brought to our attention when we first demonstrated our implementation in January 1984 to officials from the Home Office who were involved in drafting the act.

We believe that many of the potential advantages of representing rules and regulations in computer-executable logical form are independent of the actual use of computers. Representation in logical form helps to identify and eliminate unintended ambiguity and imprecision. It helps clarify and simplify the natural language statement of the rules themselves. It can also help to derive logical consequences of the rules and therefore test them before they are put into force—again, points that were also made by Allen [1].

We believe that the formalization of legislation and legal reasoning offers potential contributions to computing technology itself. It should help to discriminate, better than other applications, between different knowledge representation formalisms and problem-solving schemes in artificial intelligence. Moreover, the rules and regulations that govern the management of institutions and organizations have exactly the same character as legal provisions. This suggests, therefore, an unconventional approach to the construction of software for data-processing applications. A payroll system, for instance, could be based directly on tax and sick-pay legislation, could include a representation of the company pension scheme, the rules that govern holiday allocation, and promotion regulations. We have already constructed a number of experimental systems dealing with some of these topics.

Finally, we should stress once again that we have not addressed the broad and much more difficult problem of simulating legal reasoning. Rather, we have concentrated on the limited objective of implementing rules and regulations with the purpose of applying them mechanically to individual cases. The British Nationality Act 1981 was chosen because it best facilitated the accomplishment of this more limited objective. It is an application for which Prolog, because of its foundation in logic, proved to be particularly well suited. However, we do not wish to imply that no other computer-executable formalism would be capable of achieving similar results.

In the remainder of the article, we illustrate our approach by showing how Prolog might be used to represent the very first clause of the act, and then proceed to describe the general structure of the act. To describe the methodology by which the formalization was constructed, we examine the first three clauses of the act in greater detail, discuss some applications of the formalization, and detail some of the logical difficulties we encountered. Finally, we compare and contrast such formalization of legislation with expert systems and with more conventional software techniques, and indicate how our work relates to other approaches to the computer assistance of legal reasoning.

In writing this article, we have assumed no previous knowledge of law, logic, or Prolog. The next section introduces the necessary background to Prolog.

PROLOG

The key to our approach is the representation of knowledge by means of *definite Horn clauses,* which are rules of the form

A `if` B_1 `and` B_2 `and ...` B_n

Each such clause has exactly one conclusion A, but zero or more conditions B_i, each of which is an atomic relationship among individuals. Definite Horn clauses are invoked or queried by means of conjunctions of atomic relationships such as

B_1 `and` B_2 `and ...` B_n?

These can be regarded as degenerate "rules" that have no conclusion. (The terminology "Horn clauses" is used here to cover both definite Horn clauses and Horn clause queries.)

The Horn clause form of logic is the basis of the computational paradigm, logic programming, and of the logic programming language Prolog. Every set of definite Horn clauses is a Prolog program.

As an example of formalization using Horn clauses, consider the first clause of the British Nationality Act:

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

The act states that "after commencement" means after or on the date on which the act comes into force.

As a first approximation, 1.-(1)(a) can be represented by the rule

```
x is a British citizen
  if  x was born in the U.K.
  and x was born on date y
  and y is after or on commencement
  and z is a parent of x
  and z is a British citizen on date y
```

Here x, y, and z are *variables,* which can have any values. For example, the common knowledge that a parent is either a father or a mother can itself be expressed by two rules:

```
z is a parent of x if z is mother of x
z is a parent of x if z is father of x
```

It is important to note that variables in different rules are distinct even if they look the same.

Facts such as

```
Peter was born in the U.K.
William is father of Peter
```

can be regarded as *degenerate rules* that have a conclusion, but no conditions.

If we introduce the concept of having a parent who qualifies under 1.-(1), defined by the rules

```
x has a parent who qualifies
    under 1.1 on date y
  if  z is a parent of x
  and z is a British citizen on date y
x has a parent who qualifies
    under 1.1 on date y
  if  z is a parent of x
  and z is settled in the U.K. on date y
```

then we can combine rule 1.-1(a) with the corresponding rule 1.-1(b), to obtain a rule that represents all of subsection 1.-(1):

```
x is a British citizen
  if  x was born in the U.K.
  and x was born on date y
  and y is after or on commencement
  and x has a parent who qualifies
      under 1.1 on date y
```

We will see later that this formalization of 1.-(1) is inadequate; partly because of the need to determine

Part 1	British Citizen Acquisition; entitlement to register/naturalize; Renunciation and resumption
Part 2	British Dependent Territories Citizen Acquisition; entitlement to register/naturalize; Renunciation and resumption
Part 3	British Overseas Territories Citizen Acquisition; entitlement to register/renunciation
Part 4	British Subject Acquisition; entitlement to register/renunciation
Part 5	Interpretations and Miscellany
Schedules	

The British Nationality Act 1981 consists of five parts and nine supplementary schedules, summarized above. The first four parts define four categories of citizenship. The fifth part and the schedules include definitions needed for the other four parts. Each of the first four parts deals with automatic acquisition of citizenship (by birth, for example), entitlement to register, and provisions for renunciation of citizenship. Parts 1 and 2, in addition, define entitlement to naturalize and resume citizenship after renunciation.

FIGURE 1. The Structure of the British Nationality Act 1981

the date on which an individual acquires British citizenship, and partly because elsewhere in the act it is necessary to know the section by which an individual is deemed to be a British citizen. We will also see that Horn clause logic itself is not entirely adequate for representing legislation in a natural manner and that, in many cases, Horn clause logic must be extended to allow negated conditions in rules. The resulting fragment of predicate logic will be called *extended Horn clause logic.*

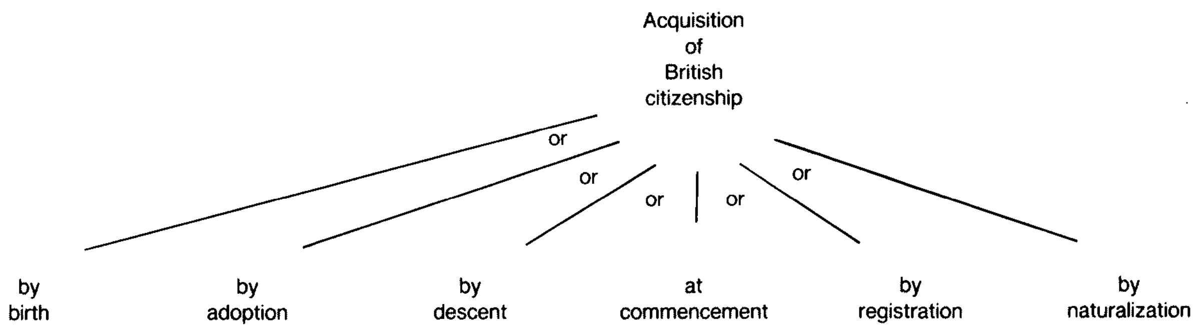
THE STRUCTURE OF THE ACT

The structure of the act is not apparent from the English text. The table of contents at the front of the text does give an abstract indication of its general contents, but not of the relationships among the different abstractions (see Figure 1).

We have an independent interest [26] in the relationship between logic programming and structured systems analysis [9]. The formalization of the act provided an opportunity for us to pursue this interest, and we attempted to use the data-flow diagrams of structured systems analysis. We were disappointed, however, to discover that data-flow diagrams were too procedural for our needs. It proved particularly impossible to identify a clear direction of data flow, and we eventually decided to use and/or graphs instead.

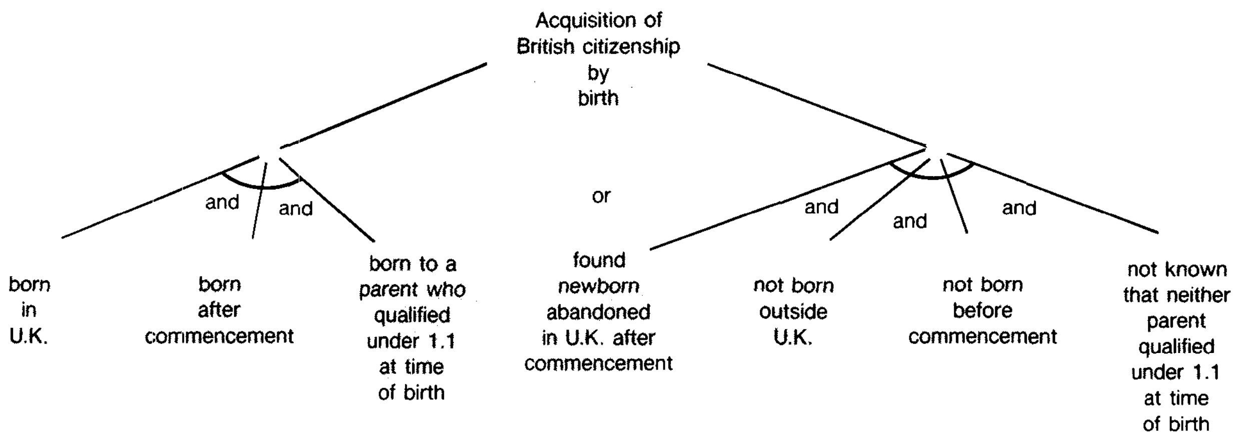
And/or graphs can be viewed as a graphical syntax for Horn clause logic [24]. Like data-flow diagrams, they encourage a structured, top-down view of information. But, unlike data-flow diagrams, they focus attention on logical structure rather than on data. In our use of and/or graphs, we actually ignored many of the data parameters altogether in order to avoid distracting detail (see Figures 2 and 3).

Acquisition by birth, displayed in Figure 3, divides into two cases depending on whether or not the in-



Shown is an and/or graph representation of the top level of acquisition of British citizenship in Part 1 of the act. Within this part of the act, it is possible to acquire British citizenship by six different routes.

FIGURE 2. Six Routes to Acquisition of British Citizenship


Acquisition of British citizenship by birth
or
and
and
born in U.K.
born after commencement
born to a parent who qualified under 1.1 at time of birth
found newborn abandoned in U.K. after commencement
and
and
and
not born outside U.K.
not born before commencement
not known that neither parent qualified under 1.1 at time of birth

An and/or graph representation of the top level of acquisition of British citizenship by birth. This section of the act is divided into two cases, depending on whether or not the individual was found abandoned as a newborn infant.

FIGURE 3. Acquisition of British Citizenship by Birth

dividual was found abandoned as a newborn infant. The left side of Figure 3 is dealt with by Section 1.-(1) of the act, as described above. The right side, which deals with the case of newborn abandoned infants, is a reformulation of Section 1.-(2):

> (2) A newborn 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.

This can be interpreted as expressing that the conditions of Subsection 1.-(1) will hold provided that it *cannot* be shown that the conditions of 1.-(1) do *not* hold. In these circumstances, the conclusion of 1.-(1)—that the newborn abandoned infant is a British citizen—will also hold.

As a first approximation, Subsection (2) can be expressed by the rule

```
x is a British citizen
  if  x was found as a newborn
      infant abandoned in the U.K.
  and x was found on date y
  and y is after or on commencement
  and not[x was not born in the U.K.
          after or on commencement]
  and not[x was not born to a parent
          who qualifies under 1.1 at
          time of birth]
```

The treatment of negation in the last two conditions above, however, is problematic and will be discussed later.

FORMALIZATION BY TRIAL AND ERROR

We have already described Sections 1.-(1) and 1.-(2) and a first approximation to their formalization. The next section of Part 1 of the act, 1.-(3), shows that our previous formalizations of Sections 1.-(1) and 1.-(2) were incomplete. It is insufficient to conclude only that an individual is a British citizen; it is also *necessary to determine the section under which citizenship is acquired:*

> (3) A person born in the United Kingdom after commencement who is not a British citizen by virtue of subsection (1) or (2) shall be entitled to be registered as a British citizen if, while he is a minor -
> (a) his father or mother becomes a British citizen or becomes settled in the United Kingdom; and
> (b) an application is made for his registration as a British citizen.

This section also shows that a more explicit treatment of time is necessary.

We can deal with both shortcomings at once by changing the conclusions of rules 1.-(1) and 1.-(2) to

```
"x acquires British citizenship on
     date y by sect. 1.1" and
"x acquires British citizenship on
     date y by sect. 1.2."
```

Sections 1.-(1) and 1.-(2) can then be represented by the rules

```
x acquires British citizenship on
    date y by sect. 1.1
  if  x was born in the U.K.
  and x was born on date y
  and y is after or on commencement
  and x has a parent who qualifies
      under 1.1 on date y
x acquires British citizenship on
    date y by sect. 1.2
  if  x was found as a newborn infant
      abandoned in the U.K.
  and x was found on date y
  and y is after or on commencement
  and not[x was not born in the U.K.
          after or on commencement]
  and not[x was not born to a parent
          who qualifies under 1.1
          at time of birth]
```

The possession of British citizenship can be related to its acquisition by the following rule:

```
x is a British citizen on
    date y by sect. z
  if  x is alive on y
  and x acquires British citizenship
      on date y1 by sect. z
  and y is after or on y1
  and not[x ceases to be a British
          citizen on date y2 and
          y2 is between y1 and y]
```

A person can cease to be a citizen by renunciation or by being deprived of citizenship. The condition

```
"x is alive on y"
```

ensures that a person ceases to be a citizen at death.

Notice that this formalization of Sections 1.-(1) and 1.-(2) makes the assumption, not explicitly stated in the act, that an individual who acquires citizenship by 1.-(1) does so at birth, and one who acquires citizenship by 1.-(2) does so at time of discovery. This assumption means that the formalization of 1.-(2), in particular, is only an approximation to the act. The act states that an abandoned infant, under the appropriate circumstances, should be treated as if he or she satisfied the conditions for 1.-(1). We should therefore conclude that such an abandoned infant becomes a British citizen at the time of his or her birth, since that is what we have assumed for Section 1.-(1).

This is, however, problematic—we cannot expect that the time of birth of an abandoned infant will be known exactly. It is much simpler to assume that an abandoned infant becomes a citizen at the time he or she is found. This is a reasonable approximation, particularly since an abandoned newborn infant could not be discovered very long after the time of his or her birth. On the other hand, it would also be very easy to change our formalization of 1.-(2) to conclude that the individual acquires citizenship at the time of birth.

The modification of the rules for 1.-(1) and 1.-(2) in the light of 1.-(3) illustrates the process of progressively refining the formalization by trial and error. In addition to adding extra parameters, as described above, the necessary modifications are sometimes achieved by adding more rules or conditions to the definition of a concept formalized earlier. For example, it turns out that the earlier formalization of

```
"x has a parent who qualifies
     under 1.1 on date y"
```

is an oversimplification. Section 48 of the act provides for the case of posthumous children. Under certain circumstances, a parent of a child qualifies for 1.1 even if that parent dies before the child is born. The effect of Section 48 is straightforwardly incorporated, by adding more rules to the earlier formalization to cover the case of posthumous children.

In certain cases, incorporating later sections of the act requires a more drastic restructuring of the formalization. For example, Section 50.-(9) specifies that a man is the "father" of only his legitimate children, whereas a woman is the "mother" of all her children, legitimate or not. Section 50.-(9) requires only a minor adjustment of the existing rules. Section 47, however, further complicates matters by allowing illegitimate children to become legitimate by the subsequent marriage of their parents. Section 47, therefore, suggests that what is important in the act is not that

```
"x is a parent of y"
```

but rather that

```
"x is a parent of y on date z."
```

This change in turn requires a modification of all rules that include the notion of "parent" in their conditions.

If we were writing a program, such formalization by trial and error would be regarded as bad programming methodology [10]. Good methodology consists of rigorously deriving correct programs from correct program specifications. It can be argued, however, that the formalization of legislation is closer to program specification or even to systems analysis than it is to programming. There is an analogy here with axiomatic systems in mathematics. The formalization of legislation results in an axiomatic theory that represents the legislation; the derivation of programs from specifications corresponds to the proof of theorems from axioms. Although we rightly demand that theorems be rigorously derived from axioms and similarly that programs be rigorously derived from specifications, there is no correspondingly rigorous way to justify the axioms themselves. The formulation of such axioms must inevitably be a trial-and-error process. The philosopher Imre Lakatos has referred to this as the "quasi-empirical" nature of mathematics [27].

APPLICATIONS OF THE FORMALIZATION

Because the formalization of the British Nationality Act is an axiomatic theory, any logical consequence of the axiomatization can, in theory, be derived by means of a complete mechanical theorem prover. Although Prolog cannot deal with arbitrary sentences of logic, it is a special-purpose theorem prover that is very efficient for proving certain kinds of simple theorems from axioms formulated as ex-

```
Is Peter a British citizen on date (16 Jan 1984) by sect. Z?
        Which X : Peter was born on date X ? (3 May 1983)
        Is it true that Peter died before (16 Jan 1984) ? no
        Is it true that Peter was born in the U.K. ? yes
        Which X : X is father of Peter ? why
---------------------------------------------------------------------
   if   X is father of Peter
   then X is a parent of Peter

   if   X is a parent of Peter
   and  X is a British citizen on date (3 May 1983)
   then Peter has a parent
        who qualifies under 1.1 on date (3 May 1983)

        Peter was born in the U.K.
        Peter was born on date (3 May 1983)
        (3 May 1983) is after or on commencement, so
   if   Peter has a parent
        who qualifies under 1.1 on date (3 May 1983)
   then Peter acquires British citizenship
        on date (3 May 1983) by sect. 1.1

        Peter is alive on (16 Jan 1984), so
   if   Peter acquires British citizenship
        on date (3 May 1983) by sect. 1.1
   and  (16 Jan 1984) is after or on (3 May 1983)
   and  not[Peter ceases to be a British citizen on date Y
            and Y is between (3 May 1983) and (16 Jan 1984)]
then Peter is a British citizen on date (16 Jan 1984) by sect 1.1
---------------------------------------------------------------------
     Which X : X is father of Peter ? William
     Which X : William was born on date X ? (1 March 1952)
   Is it true that William died before (3 May 1983) ? no
   Is it true that William was born in the U.K. ? yes
   Is it true that William was found as a newborn infant
                   abandoned in the U.K. ? no
   Is it true that William was adopted ? no
   Is it true that William was a citizen of the U.K. and
                   Colonies on date (31 Dec 1982) ? yes
```

FIGURE 4. Extract of an APES Dialogue

tended Horn clauses. Because of limitations imposed for the sake of efficiency, however, Prolog sometimes goes into infinite loops [25] and fails to prove theorems that are logically implied by the axioms. Nevertheless, it can still be used to prove a large class of theorems and can help test a set of axioms by assisting in the derivation of logical consequences.

An obvious application of the formalization of the act is to determine whether in a given circumstance a particular individual is or is not a British citizen. To do so, one must have access to facts concerning the individual as well as to other information not supplied by the act. A convenient way to provide such information is to supply it dynamically as it is required by the theorem prover. The augmented Prolog system, APES, itself implemented in Prolog and initially developed as an expert-system shell, generates the appropriate queries to the user. The answers it produces are logical consequences of the rules contained in the formalization of the act together with the information obtained from the user. In addition, APES can provide explanations for both *why* a given question is being asked and *how* a given solution has been obtained.

These features are illustrated in Figure 4, where user input is indicated by boldface type. To make the dialogue easier to follow, the complications of posthumous and illegitimate children are omitted. We have included in the program the information

```
    Is it true that on date (31 Dec 1982) William had the
                    right of abode in the U.K. under the Immigration Act 1971 as
                    then in force ? yes

                                                  .
                                                  .
                                                  .

    Is it true that Peter has ever renounced British citizenship ? no
    Is it true that Peter has ever been deprived of British
                    citizenship by section 40 ? no

 Yes, I can confirm that
  Peter is a British citizen on date (16 Jan 1984) by sect. 1.1 how

 I deduced
  Peter is a British citizen on date (16 Jan 1984) by sect. 1.1
 from the rule

       x is a British citizen on date y by sect. z
         if  x is alive on y
         and x acquires British citizenship on date y1 by sect. z
         and y is after or on y1
         and not[x ceases to be a British citizen on date y2
                 and y2 is between y1 and y]

 I can show
       1 Peter is alive on (16 Jan 1984)
       2 Peter acquires British citizenship
         on date (3 May 1983) by sect. 1.1
       3 (16 Jan 1984) is after or on (3 May 1983)
       4 not[Peter ceased to be a British citizen on date X
             and X is between (3 May 1983) and (16 Jan 1984)]
 how 2

 I deduced
       Peter acquires British citizenship
       on date (3 May 1983) by sect. 1.1
 from the rule

                                                  .
                                                  .
                                                  . etc
```

FIGURE 4. Extract of an APES Dialogue

that the act came into effect on January 1, 1983.

A consultation of the system can be invoked by asking, for example, whether a given individual is a British citizen by any section of the act on some given date.

Figure 4 shows how Prolog proves theorems by reasoning backwards from conclusion to conditions. Rules of the form

```
A if B and C and ...
```

are interpreted as *procedures*:

```
to show A,
   show B and C and ....
```

It is this procedural interpretation of rules that makes Prolog a programming language as well as a theorem prover.

Viewed as a programming language and compared with other programming languages, Prolog is especially well suited for the implementation of legislation because it is *nondeterministic*. Different ways of establishing the same conclusion can be represented by different rules; the implementation takes responsibility for systematically exploring the alternatives. In the case of Prolog, this exploration takes the form of a depth-first search determined by the order in which the rules are written.

As we have already argued, the main strength of Prolog for such applications is its foundation in logic. Statements in Prolog refer directly to the domain of discourse, not to the state of computer memory as they do in imperative programming languages. Moreover, individual statements can be understood independently of one another. This too contrasts with the situation in imperative languages, where because of side effects the meaning of individual program statements depends on the context in which they occur.

SOLVING SUBPROBLEMS

Prolog tackles the solution of subproblems, "*B* and *C* and ...," in the order in which they are written. Like the order of clauses, this too is under the programmer's control.

Conceptually, such subproblems can be solved in different ways:

- By means of other rules

  For example, the condition

  ```
  "z is settled in the U.K.
        on date y"
  ```

  can be established by using rules that formalize the definition of settlement, given in Section 50 of the act.

- By accessing data

  For example,

  ```
  "y is a Dependent Territory"
  ```

  can be determined by matching the condition against degenerate, conditionless rules that solve the problem without introducing further subproblems.

- By querying the user

  For example,

  ```
  "x was born on date y"
  ```

  can be determined by posing the problem to the user for solution. The answer can be stored in rule form, like any other information used by the system.

- By means of computation

  For example,

  ```
  "y is after commencement"
  ```

  can be computed by means of a program. Because of the procedural interpretation of rules, any such program can be expressed by means of Horn clauses. (The extension to allow negative conditions is not strictly necessary, but it is desirable.)

- By querying an expert

  For example,

  ```
  "throughout the period from date u
      to v, x had the right of abode
      in the U.K. under the 1971
      Immigration Act."
  ```

  The solution of this subproblem requires knowledge of previous legislation. It can be provided by a human expert or by a computerized formalization of the Immigration Act 1971.

Reasoning backwards from problems to subproblems not only facilitates cooperative man-machine problem solving, but it also encourages top-down, goal-directed knowledge representation—the definition of high-level concepts before lower level ones. This guarantees that, at every stage in the knowledge refinement process, the current state of knowledge is relevant and applicable to the class of problems to be solved. It also means that high-level definitions can be tested before the lower level ones have been defined, by querying the system designer for the solution to undefined subproblems.

This is a complete reversal of the normal approach to the development of axiomatic systems. The normal methodology starts with a primitive set of concepts and axioms. Higher level concepts are defined bottom up in terms of lower level ones that are primitive or have already been defined. A major problem with this approach is the difficulty of identifying an appropriate bottom level of primitive, undefined concepts. In the case of the British Nationality Act 1981, it would be difficult to decide how to treat the earlier Immigration Act 1971. Even worse, we would have to decide from the outset whether concepts such as

```
"x is a newborn infant"
```

and vague concepts in general would need to be treated as primitive or could be defined.

SOME DIFFICULTIES WITH THE FORMALIZATION OF NEGATION

Horn clause logic is that fragment of full first-order logic that allows sentences with at most one unnegated conclusion and any number of unnegated conditions. Extended Horn clause logic allows some or all of the conditions to be negated if necessary. This in turn, as observed by Lloyd and Topor [28], makes it possible to express arbitrary expressions of first-order logic in the conditions. Extended Horn clause logic, however, does not allow disjunctive or negative conclusions. We did not expect that an inability to express disjunctive conclusions would be a problem in formalizing the British Nationality Act. Legislation attempts to be definite, after all, and this expectation was confirmed. We did, however, expect to need negative conclusions.

In most places in the act where we needed to deal with negation, a straightforward interpretation of negation as failure was adequate (see below). This proved to be the case, for example, in the treatment of exceptions. Legislation is often drafted by a general rule, followed separately by a list of exceptions to it. Such rules are also common in the British Nationality Act.

Sections 11-(1) and 11-(2) of the act, for example, state that

> 11-(1) Subject to subsection (2), a person who immediately before commencement -
> (a) was a citizen of the United Kingdom and Colonies; and
> (b) had the right of abode in the United Kingdom under the Immigration Act 1971 as then in force,
> shall at commencement become a British citizen.
> (2) A person who ... [*P*] ... shall not become a British citizen under subsection (1) unless ... [*Q*] ...

By expressing the details of 11-(2) by means of a new predicate, viz.

```
x is prevented by 11.2 from acquiring
   British citizenship at commencement
 if ... [P] ...
 and not ... [Q] ...
```

Section 11-(1) can be expressed by the rule

```
x acquires British citizenship on date
   y by sect. 11.1
  if  commencement is on y
  and y1 is immediately before y
  and x was a citizen of the United
      Kingdom and Colonies on y1
  and on date y1, x had the right of
      abode in the U.K. under
      the Immigration Act 1971
      as then in force
  and not[x is prevented by 11.2 from
         acquiring British citizen-
         ship at commencement]
```

The negation in the last condition of the rule for 11-(1) is interpreted as negation as failure [7]:

not[*Q*] holds

when

all ways of showing *Q* fail.

The treatment of negation as failure is justified whenever we can make a "Closed World" Assumption:

Anything which is not *known* is assumed to be *false*.

With this assumption, negation as failure is consistent with ordinary, classical negation.

The interpretation of negation as failure is often appropriate for handling exceptions. If we cannot show that an individual is excepted, then it is natural to assume that he or she is *not* excepted.

Negation as failure can be implemented very easily and efficiently in a logic programming framework. It is, however, inappropriate in those circumstances where it is unreasonable to make a Closed World Assumption. We could not make an all-embracing Closed World Assumption, for example, if we had reason to believe that there is some other way of acquiring British citizenship that is not covered by the provisions of the British Nationality Act. It is notoriously difficult in law to determine all the legal provisions that might be relevant to deciding a particular case. In such circumstances, we would be forced to abandon negation as failure and instead employ theorem provers that can reason with ordinary negation. The need for such reasoning, however, potentially entails the need to reason with all of first-order logic. Theorem provers that can reason with all of first-order logic are substantially less efficient than those that are restricted to extended Horn clause form.

We would nevertheless suggest that there are many instances where the legislation explicitly specifies all the cases for which a given predicate is intended to hold, and where the interpretation of negation as failure can safely be made. This is not the case, unfortunately, with the use of negation in Section 1-(2) of the act, which states the following:

(2) A newborn 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.

Earlier in the article, we formalized this section as the rule

```
x acquires British citizenship on
    date y by sect. 1.2
  if  x was found as a newborn infant
      abandoned in the U.K.
  and x was found on date y
  and y is after or on commencement
  and not[x was not born in the U.K.
          after or on commencement]
  and not[x was not born to a parent
          who qualifies under 1.1 at
          time of birth]
```

The last two conditions of this rule contain double negations, which would cancel each other out if both were interpreted classically:

not[not *P*] is equivalent to *P*.

Obviously, this is not what is intended by the act. Intuitively, it seems clear that the first occurrence of "not" in both conditions should be interpreted as negation as failure. We argue, however, that the second occurrence, underlined above, should be interpreted as ordinary, classical negation.

Suppose, for example, we attempt to interpret the second "not" in the last condition also as negation as failure by adding the Prolog rule:

```
x was not born to a parent who
    qualifies under 1.1 at
    time of birth
  if not[x was born on date y and
         x has a parent who qualifies
         under 1.1 on date y]
```

Putting aside the problem of determining the exact date of birth of an abandoned infant, suppose that we know neither the father nor the mother of a particular abandoned infant. Then the condition

```
"x has a parent who qualifies
     under 1.1 on date y"
```

will *fail* for this infant (whether we know the date of birth or not). Consequently, the condition

```
"x was not born to a parent who
     qualifies under 1.1 at time
     of birth"
```

will *succeed*, and the negative condition

```
"not[x was not born to a parent
     who qualifies under 1.1 at time
     of birth]"
```

will *fail*. We fail to conclude that the abandoned infant acquires citizenship by Section 1-(2).

This conclusion is exactly the opposite of what is intended by the act. If both parents of the infant are unknown, then we cannot show that they do *not* satisfy the conditions specified, and we should conclude by Section 1-(2) that the infant is a British citizen.

The last two conditions of Section 1-(2) seem to involve default reasoning of the general form

infer *P* if fail to show "'not *P*"

combining both failure to prove and classical negation.

Negation as failure

infer "not *P*" if fail to show *P*

is obviously similar, but easier to implement because we can state precisely what "fail to show" means for positive conditions in the context of extended Horn clauses.

The similarity between the two default rules can often be exploited, however, to get the effect of the required default reasoning. In the present example, we can obtain the effect of the first default rule

infer *P* if fail to show "not *P*"

by using a rule of the form

infer *P* if fail to show *Q*

where the conditions *Q* are equivalent to "not *P*" in the context of the problem we are considering.

Thus, the first of the problematic conditions of Section 1-(2)

```
"x was not born in the U.K. after or
    on commencement"
```

can be replaced by

```
"x was not born in the U.K.
```

or

```
x was not born after or on
    commencement."
```

If we further assume that

```
"not born in the U.K."
```

is equivalent to

```
"born outside the U.K."
```

and that

```
"not born after or on commencement"
```

is equivalent to

```
"born before commencement"
```

then we can replace the original condition

```
"not[x was not born in the U.K. after
     or on commencement]"
```

in the rule for Section 1-(2) by the two conditions

```
"not[x was born outside the U.K.]"
```

and

```
"not[x was born before commencement]"
```

where all occurrences of "not" are now interpreted as negation by failure.

The other condition with a double negation

```
"not[x was not born to a parent
     who qualifies under 1.1 at time
     of birth]"
```

is more problematic. We can remove the second occurrence of negation from this condition by defining

```
"x was not born to a parent
     who qualifies under 1.1 at time
     of birth."
```

explicitly as a positive predicate:

```
x was not born to a parent
    who qualifies under 1.1
    at time of birth
  if  z1 is father of x
  and z2 is mother of x
  and z1 does not qualify under 1.1 at
      time of x's birth
  and z2 does not qualify under 1.1 at
      time of x's birth
```

This definition makes use of the assumption that an individual has two parents: a unique father and a unique mother. Notice that we now establish an abandoned infant's citizenship immediately if either or both of the parents are unknown.

We now have to consider the new predicate

```
"x does not qualify under 1.1 at
     time of y's birth"
```

We can define this by writing:

```
x does not qualify under 1.1 at
    time of y's birth
  if  x was not a British citizen at
      the time of y's birth
  and x was not settled in the U.K. at
      the time of y's birth
```

In principle, we could continue with this kind of analysis, reasoning through the provisions of the act to construct explicit definitions for the predicates

```
"x was not a British citizen at the
     time of y's birth"
"x was not settled in the U.K. at the
     time of y's birth".
```

In practice, however, we could not construct such definitions; the act is too large, and there are too many separate possibilities to consider for this to be a practical solution.

In the absence of a better general solution, we circumvented the problem in practice by treating the negative information

```
"x was not a British citizen at the
     time of y's birth"
"x was not settled in the U.K. at the
     time of y's birth"
```

as part of the input that is obtained by querying the user. This treatment is not entirely satisfactory, but it does provide a reasonable solution for most practical purposes. Notice that the user is asked such questions only after it is established that the newborn infant was found abandoned in the United Kingdom, that the discovery occurred after commencement, that the infant was not known to have been born outside the United Kingdom, that the infant was not known to have been born before commencement, and after both parents have been identified. Even then, users have the option of invoking subsidiary consultations of the system to help them answer questions about citizenship and settlement of the parents.

Before briefly discussing other knowledge representation problems in the British Nationality Act, we must make one final remark about negation. The type of default reasoning that the act prescribes for dealing with abandoned infants is *nonmonotonic* [3]: Conclusions made by default in the absence of information to the contrary may have to be withdrawn if new information is made available later. Thus, in the British Nationality Act, an abandoned child may be a British citizen by Section 1-(2) because the identity of both his or her parents is unknown. Suppose, however, that the parents are subsequently identified, and it is determined that neither was a British citizen nor had settled in the United Kingdom when the child was born. Under these circumstances, the earlier conclusion that the child is a British citizen would have to be withdrawn. This possibility does not seem to have been anticipated by the drafters of the act, as there is no provision for it.

OTHER PROBLEMS IN THE FORMALIZATION

In addition to its problems with negation, extended Horn clause logic is not adequate for dealing with *counterfactual conditionals.* Such conditionals occur frequently in the British Nationality Act. Clause 14-(1)(e), for example, includes in its conditions the phrase

> . . . became a British citizen by descent or would have done so but for his having died or ceased to be a citizen . . . [by] renunciation.

The treatment of such counterfactual conditions is notoriously difficult. It is not obvious what factors need to be taken into account before it can be determined that a person would have been a citizen if the person had not died.

It can be argued, however, that, when the drafters used the counterfactual phrase in this clause, they did not intend such an open-ended analysis. The device is used merely for convenience: The drafters avoid listing a complicated set of conditions explicitly by specifying instead a modification to some other part of the legislation.

We decided to treat counterfactual conditions by writing additional alternative rules; one set describing, for example, the conditions for acquisition of citizenship at commencement for individuals who were alive on that date, and another set for individuals who had died before that date, but otherwise met all the other requisite conditions before death. This treatment of counterfactual conditions is extremely tedious. It requires a thorough analysis of the provisions of the act before the implicitly described rules can be reconstructed; it also substantially increases the number of rules in the formalization. All of this is precisely what the person drafting the rule was trying to avoid by using the counterfactual phrase in the first place. So, although we managed to represent the effect of counterfactual conditions within extended Horn clause logic, this treatment is not entirely satisfactory.

The main difficulty with counterfactual conditions of this type lies not in representing them, but in discovering what it is that the person drafting the condition actually intended. Assuming that this can be determined, there are then several techniques available for implementing the required reasoning. One approach is to use Prolog's extralogical primitives to modify the database temporarily by deleting facts, adding other assumptions in their place, and restoring the database when the appropriate conclusions have been derived. Although this solution works in Prolog, it does so by sacrificing the logic of the knowledge representation. A different and logically sound approach is to use an amalgamation of object language and metalanguage [4]. This uses a proof predicate with explicit parameters for the knowledge base, which can vary for different conditions of a rule. A Prolog implementation of such an amalgamation logic has been reported by Bowen and Weinberg [5].

A number of other phrases in the act seemed problematic upon first reading, but were not so in practice. Clause 3-(4), for example, states that

> If in the special circumstances of a particular case the Secretary of State thinks fit, he may treat subsection (2) as if the reference to twelve months were a reference to six years.

From the Secretary of State's point of view, Clause 3-(4) grants permission to apply discretion in certain circumstances. For the purposes of formulating rules dealing with the acquisition of citizenship, however, we were able to treat this kind of statement by writing two separate rules. One rule covered the standard case; the other covered the discretionary case, with an extra condition to indicate that the rule only applies if the Secretary of State decides that it should. This treatment gives a reasonable representation of the discretionary clause (for the purpose of determining citizenship), at the cost of increasing the number of rules in the formalization.

In addition to the problems discussed above, knowledge representation problems arose because of the large scale of the work. We have already stressed the trial-and-error development of our formalization. When the system was in its early stages and the number of rules in the formalization was small, restructuring the rules to incorporate the effect of later sections was relatively simple. As the formalization developed and there were hundreds of rules and many tens of predicates to consider, incorporating even a minor change was not always easy. In fact, whether a change was easy to incorporate or required a major restructuring of the rules was largely a matter of luck, usually depending on whether a convenient predicate had initially been chosen. We see no alternative to trial-and-error formalization, at least when dealing with legislation already in existence. Therefore, we are developing a programming environment that will incorporate metalevel data "dictionaries" and special-purpose editors to assist in the process.

THE STATE OF THE IMPLEMENTATION

The first four parts of the act, the definitions in Part 5, and the schedules that were needed for Parts 1–4 (approximately 50 pages of the 73-page act) were translated into extended Horn clause logic during July and August 1983 by a student, Fariba Sadri, without any expert legal assistance. Those sections in Part 5 and the schedules not translated into logic consist mostly of descriptions of amendments to other acts, repeals, offenses, and proceedings related to them, and decisions involving exercise of discretion by the Secretary of State.

The entire system, including APES, was implemented in micro-Prolog. At the time formalization was completed, only a small part of the act could be loaded, together with APES, into the small Z-80-based microcomputers on which micro-Prolog was then available. During October and November 1983, most of the work on the system was concerned with overcoming the space limitation. As of December 1983, the system ran a relatively self-contained part of the act, consisting of approximately 150 rules dealing with the acquisition of British citizenship, on a microcomputer with 128 kbytes of memory. This small demonstration system consists of rules for the sections of Part 1 that describe the acquisition of British citizenship, rules that formalize the relevant sections of Part 5 and the schedules, and rules that express certain general knowledge (such as the rule that a father or a mother is a parent, and facts about the lengths of the months). We estimate that a micro-Prolog system capable of addressing 512 kbytes of memory would be sufficient to run the complete act, which contains about 500 rules. Such micro-Prolog systems are now available for microcomputers and, in the form of sigma-Prolog, for a range of larger machines. These are recent developments, however, and at the time of this writing, we have not transferred the formalization to the larger systems. We have chosen instead to consolidate our experience by considering a number of other examples from the legal domain, some of which are listed in the concluding section of this article.

THE RELATIONSHIP WITH EXPERT SYSTEMS

The formalization of legislation by means of rules has almost all the characteristics of an expert system. It differs, however, in one important respect: Before knowledge can be formalized in a classical expert system, it has to be elicited from the subconscious of an expert. Feigenbaum and McCorduck [11] refer to this knowledge elicitation problem as "the most important of the central problems of artificial intelligence research" and "the critical bottleneck." The knowledge elicitation problem is almost entirely absent in the formalization of legislation. By its very nature, the law is well documented; its provisions are written down, and where they are not, decisions in previous cases are recorded for future reference. Even if this documentation is not already in a form that can be expressed directly in computer-intelligible terms, it provides a convenient framework around which the knowledge elicitation process can proceed.

This does not mean, however, that there are no knowledge representation problems. As previously mentioned, early versions of the formalization had to be refined by trial and error to incorporate the effects of later sections of the act. Other knowledge representation problems that arose in the course of the formalization were described earlier in this article. These problems are much less severe than the knowledge elicitation problem in general.

The advantages claimed for expert systems are primarily due to the explicit representation and separation of knowledge from its manipulation by means of deductive inference procedures. In this respect, the use of logic for knowledge representation and of deductive inference for knowledge processing is the purest form of expert-system technique. Knowledge expressed in such a form

- is easy for both naive users and experts to understand;
- is easy to modify (e.g., to correct errors, to enhance, and to reflect changes that occur over time);
- allows the inference procedure to interact naturally with the human user and to explain its conclusions.

These advantages hold equally for other applications implemented by the same techniques, and therefore for Prolog "programs" in general—provided they are structured according to logic programming principles.

Logic can also be used to formalize regulations, rules, and policies that do not have legal authority. An airline company, for example, might use such techniques to assist in drafting and applying rules for refunding tickets or changing reservations. A bank might use them for rules about banking charges. A customer might query such a system to obtain explanations for decisions that would otherwise be inscrutable. Such applications do not necessarily qualify as expert systems, because they can be used to assist in the formulation and debugging of rules in situations where expertise does not yet exist.

THE RELATIONSHIP WITH CONVENTIONAL PROGRAMMING TECHNIQUES

Conventional programming techniques can also be used for advanced applications such as expert systems and the formalization of legislation. Decision tables and decision trees are probably the most appropriate of the conventional techniques.

Figure 5 is an example of a possible decision tree for the British Nationality Act. Because they separate information from the way in which it is processed, decision trees share many of the advantages of rule-based systems in general. Moreover, because rules corresponding to branches of a decision tree have an especially simple structure, they can be implemented straightforwardly in conventional programming languages such as Cobol or Basic.

Although decision tables and trees can be regarded as representing rules, they are optimized for the solution of a predetermined class of problems. They do not aim to support the derivation of arbitrary logical consequences of the rules.

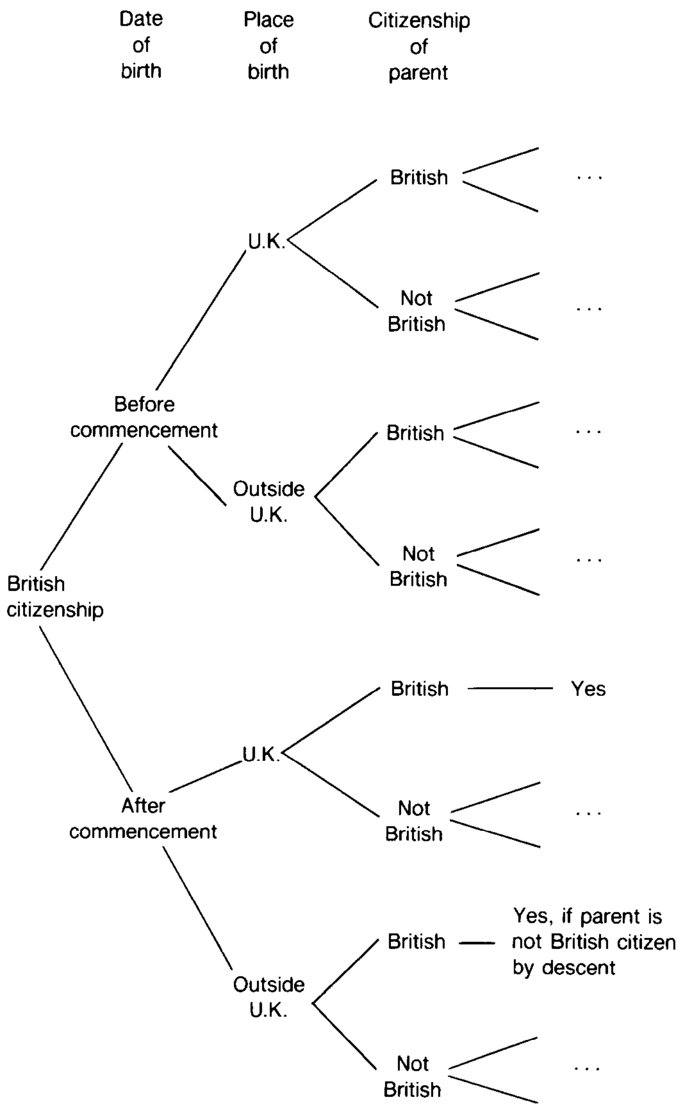
COMPARISON WITH RELATED WORK

There is now a substantial amount of literature concerned with the computer assistance of legal reasoning. Much of this deals with retrieval of legal documents and is not directly related to the work reported here.

Most of the more ambitious approaches are based on techniques developed in artificial intelligence and fall primarily into two categories: those that concentrate on reasoning by means of rules, and those that concentrate on case law and reasoning from examples. Our work falls within the former category; as does McCarthy's TAXMAN I [31], applied to corporate tax law and implemented in a version of micro-PLANNER [40], and Waterman and Peterson's rule-based analysis of personal injury claims, implemented in the programming language ROSIE [42] at the Rand Corporation. McCarthy's later work on TAXMAN II [32] has concentrated on reasoning with case law. Several other projects, including Gardner's treatment of offer and acceptance in contract law [13, 14] implemented in MRS [15], and Meldman's [33] study of the tort law of assault and battery represented in OWL [41], attempt to combine reasoning by rules with reasoning from previous cases.

Other projects that have recently investigated the application of Prolog to law include those of Hustler [21], MacRae [30], Hammond [17], Gordon [16], and Schlobohm [34]. Our project is similar to theirs, but emphasizes two considerations that have not always received the same attention in other Prolog projects:

1. We have concentrated on the formal representation of already written legislation. This has al-



A decision tree can be regarded as an optimized collection of rules—each complete branch corresponds to a single rule.

FIGURE 5. The Beginning of a Decision Tree for British Citizenship

lowed us to implement a more substantial application than would have been possible if we had had to deal with the knowledge elicitation problem.

2. We have attempted to distinguish as much as possible between using logic to represent legislation and using Prolog to implement such representations. Our primary commitment has been to the use of logic, and we have used this commitment to propose and test various extensions of Prolog.

Our work on the application of logic programming in law began with Sergot's investigation [35] of Stamper's LEGOL language and its relationship with logic programming. The aim of the LEGOL project was the development of a computer language for representing legislation and the structure of regulation-based organizations [38]. In LEGOL, Stamper developed a method of analysis based on a semantic model that provided the basis for a computer language in which rules could be written to simulate the effects of legal provisions. The language that emerged (in its executable versions at least) was conceived as an extended relational algebra with special operators for handling time [23]. Computation proceeded by executing the control structures and evaluating expressions of the algebra. A program written in the LEGOL language was built by combining rules using a variety of conventional program control structures including sequencing of rules, if-then-else statements, and iteration [22].

Sergot showed how LEGOL rules and control structures could be reinterpreted in logic programming terms. Such a translation ignores the emphasis placed by Stamper on the importance of LEGOL's semantic model, but it does suggest a method of computing with LEGOL rules that is independent of the semantic considerations. Interpretation of LEGOL rules as statements of logic gives a *description* of legislation as well as a simulation of its effects; it frees the LEGOL language from the need to be embedded within an algorithmic programming language; and, by executing LEGOL rules backwards in the spirit of logic programming, it allows recursion to be expressed directly, a feature missing from the original LEGOL algebra.

The other main difference between our approach and Stamper's is a methodological one. Representation of a fragment of legislation in LEGOL proceeds in two distinct steps: First, the LEGOL semantic model is used to analyze and identify the entities, concepts, and relationships present in the legislation. LEGOL rules are then formulated to manipulate the concepts identified in the analysis phase. Indeed Stamper's primary concern has been the development of the LEGOL semantic model; the implementation of practical applications is a secondary objective.

In contrast with the LEGOL methodology, we have stressed in this article the top-down, goal-directed development of our formalization of the British Nationality Act. We adopted this approach for purely practical reasons. It allowed us to delay addressing the more complex issues of knowledge representation until it became unavoidable to do so, and it enabled us to avoid considering how to represent the various commonsense knowledge needed to understand the legislation until we discovered what knowledge was required.

CONCLUSION

The British Nationality Act experiment has largely served its purpose, at least for the time being. It has demonstrated the feasibility and promise of applying Prolog logic to a potentially large class of applications dealing with the implementation of rules and regulations. These applications include not only cases of statutory law, such as the British Nationality Act 1981, but also applications normally associated with advanced data processing.

Following our formalization of the British Nationality Act, a variety of smaller projects have been completed within the Logic Programming Group at Imperial College. Those incorporating realistically sized fragments of legislation include a subset of the Immigration Act 1971 [39], regulations for government grants to industry [29], and a large company's pension regulations, with associated tax legislation [6]. A formalization of Statutory Sick Pay legislation [37] has also been attempted in collaboration with the group. Application of logic programming in law, and some of the more general problems referred to in this article, is discussed in more detail in [36].

The British Nationality Act 1981 has also proved to be a rich source of problems for the use of logic for knowledge representation. It has highlighted problems with negation as failure and counterfactual conditionals in particular. We believe that the formalization of legislation and the development of computer systems to assist with legal problem solving and decision making provide a rich domain for developing and testing artificial intelligence technology. More tentatively, the accumulated experience of managing complex systems of law may teach us something about the maintenance of large bodies of complex software.

Acknowledgments. We are grateful to Chris Moss for suggesting the formalization of the British Nationality Act 1981 and to Ronald Stamper and other members of the LEGOL project for their early interactions with our work, and for their comments and their encouragement.

REFERENCES
Reference [12] is not cited in text.
1. Allen, L.E. Symbolic logic: A razor-edged tool for drafting and interpreting legal documents. *Yale Law J. 66* (May 1957), 833–879.
2. Bench-Capon, T., and Sergot, M.J. Towards a rule-based representation of open texture in Law. Rep., Dept. of Computing, Imperial College, London, 1985. Also presented at the 2nd International Conference on Computers and Law, Houston, Tex., June 1985.
3. Bobrow, D.G., Ed. Non-monotonic logic. Special issue on. *Artif. Intell. 13* (1980).
4. Bowen, K.A., and Kowalski, R.A. Amalgamating language and metalanguage in logic programming. In *Logic Programming*. K.L. Clark and S.A. Tarnlund, Eds. APIC Studies in Data Processing, vol. 16. Academic Press, New York, 1982, pp. 153–172.
5. Bowen, K.A., and Weinberg, T. A meta-level extension of PROLOG. In *Symposium on Logic Programming* (Boston, Mass.). IEEE Computer Society Press, 1985, pp. 48–53.
6. Chan, D. A logic based legal expert. M.S. thesis, Dept. of Computing, Imperial College, London, 1984.
7. Clark, K.L. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum, New York, 1978, pp. 293–322.
8. Clark, K.L., and McCabe, F.G. *Micro-PROLOG: Programming in Logic.* Prentice-Hall, Englewood Cliffs, N.J., 1984.
9. de Marco, T. *Structured Analysis and System Specification.* Prentice-Hall, Englewood Cliffs, N.J., 1979.
10. Dijkstra, E.W. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, N.J., 1976.
11. Feigenbaum, E.A., and McCorduck, P. *The Fifth Generation.* Addison-Wesley, Reading, Mass., 1983.
12. Fuchi, K. The direction the FGCS project will take. *New Generation Comput. 1* (1983), 3–9.
13. Gardner, A.V.D.L. An artificial intelligence approach to legal reasoning. Rep. STAN-CS-85-1045, Dept. of Computing Science, Stanford Univ., 1984. To be published by Bradford Books/MIT Press.
14. Gardner, A.V.D.L. Overview of an artificial intelligence approach to legal reasoning. In *Computing Power and Legal Reasoning*, C. Walter, Ed. West, St. Paul, Minn., 1985, pp. 247–274.
15. Genesereth, M.R., Greiner, R., Grinberg, M.R., and Smith, D.E. The MRS dictionary. Memo HPP-80-24, Stanford Heuristic Programming Project, Stanford Univ., Calif., Dec. 1980; revised Jan. 1984.
16. Gordon, T.F. Object-oriented predicate logic and its role in representing legal knowledge. In *Computing Power and Legal Reasoning*, C. Walter, Ed. West, St. Paul, Minn., 1985, pp. 163–203.
17. Hammond, P. Representation of DHSS regulations as a logic program. Rep. 82/26, Dept. of Computing, Imperial College, London, 1982, pp. 225–235. Also in *Proceedings of the 3rd BCS Expert Systems Conference*, British Computer Society, Cambridge, Dec. 1983, pp. 225–235.
18. Hammond, P., and Sergot, M.J. A PROLOG shell for logic based expert systems. In *Proceedings of the 3rd BCS Expert Systems Conference*. British Computer Society, Cambridge, Dec. 1983, pp. 95–104.
19. Hammond, P., and Sergot, M.J. *APES Reference Manual.* Logic Based Systems, Richmond, Surrey, England, 1984.
20. Her Majesty's Stationery Office. The British Nationality Act 1981, Chapter 61, London, 1981.
21. Hustler, A. Programming law in logic. Res. Rep. CS-82-13, Dept. of Computer Science, Univ. of Waterloo, Canada, 1982.
22. Jones, S. Control structures in legislation. In *Computer Science and Law*, B. Niblett, Ed. Cambridge University Press, New York, 1980, pp. 157–169.
23. Jones, S., Mason, P., and Stamper, R. LEGOL 2.0: A relational specification language for complex rules. *Inf. Syst. 4*, 4 (Dec. 1979), 293–305.
24. Kowalski, R.A. *Logic for Problem Solving.* Elsevier North-Holland, New York, 1979.
25. Kowalski, R.A. Logic programming. In *Proceedings IFIP-83 Congress.* Elsevier North-Holland, New York, 1983, pp. 133–145.
26. Kowalski, R.A. Software engineering and knowledge-based systems in new generation computing. *Future Generation Comput. Syst. 1*, 1 (1984), 39–49.
27. Lakatos, I. Proofs and refutations. *Br. J. Philos. Sci. 14* (1963), 1–25, 120–139, 221–243, 296–342.
28. Lloyd, J.W., and Topor, R.W. Making Prolog more expressive. *J. Logic Program. 1*, 3 (Oct. 1984), 225–240.
29. Lowes, D. Assistance to industry: A logical approach. M.S. thesis, Dept. of Computing, Imperial College, London, 1984.
30. MacRae, C.D. User control knowledge in a tax consulting system. In *2nd International Conference on Computers and Law* (Houston, Tex., June). 1985.
31. McCarthy, L.T. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harvard Law Rev. 90* (1977), 837–893.
32. McCarthy, L.T., and Sridharan, N.S. The representation of an evolving system of legal concepts: I. Logical templates. In *Proceedings, 3rd Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (Victoria, B.C.). Canadian Society for Computational Studies of Intelligence, 1980, pp. 304–311.
33. Meldman, J.A. A structural model for computer-aided legal analysis. *Rutgers J. Comput. Law 6* (1977), 27–71.
34. Schlobohm, D.A. A Prolog program which analyzes income tax issues under Section 318(a) of the internal revenue code. In *Computing Power and Legal Reasoning*, C. Walter, Ed. West, St. Paul, Minn., 1985, pp. 765–815.
35. Sergot, M.J. Programming law: LEGOL as a logic programming language. Rep., Dept. of Computing, Imperial College, London, 1980.
36. Sergot, M.J. Representing legislation as logic programs. Rep., Dept. of Computing, Imperial College, Aug. 1985. To be published in *Machine Intelligence 11.*
37. Sharpe, W.P. Logic programming for the law. In *Research and Development in Expert Systems: Proceedings of the 4th Technical Conference of the British Computer Society Specialist Group on Expert Systems, Warwick*, M.A. Braines, Ed. Cambridge University Press, New York, Dec. 1985.
38. Stamper, R. LEGOL: Modelling legal rules by computer. In *Computer Science and Law*, B. Niblett, Ed. Cambridge University Press, New York, 1980, pp. 45–71.
39. Suphamongkhon, K. Towards an expert system on immigration legislation. M.S. thesis, Dept. of Computing, Imperial College, London, 1984.
40. Sussman, G., Winograd, T., and Charniak, E. Micro-Planner reference manual (revised). A.I. Memo 203A, Artificial Intelligence Laboratory, M.I.T., Cambridge, Mass., 1971.
41. Szolovits, P., Hawkinson, L.B., and Martin, W.A. An overview of OWL, a language for knowledge representation. MIT/LCS/TM-86, Laboratory for Computer Science, M.I.T., Cambridge, Mass., 1977.
42. Waterman, D.A., and Peterson, M.A. Models of legal decisionmaking. Rep. R-2717-ICJ, Institute for Civil Justice, Rand Corporation, Santa Barbara, Calif., 1981.

CR Categories and Subject Descriptors: I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*law*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*logic programming*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*predicate logic*; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software—*Prolog, APES*; J.1 [**Computer Applications**]: Administrative Data Processing—*law*; K.6.3 [**Management of Computing and Information Systems**]: Software Management—*software development*
General Terms: Design, Experimentation, Human Factors, Languages, Reliability, Theory
Additional Key Words and Phrases: British citizenship, legislation, rule-based systems

Authors' Present Address: M.J. Sergot, F. Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond, and H.T. Cory, Dept. of Computing, Imperial College, University of London, 180 Queens Gate, London, SW7 2BZ, England.