WUENIC – A Case Study in Rule-based Knowledge Representation and Reasoning

Robert Kowalski[1] and Anthony Burton[21]

[1] Imperial College London, rak@doc.ic.ac.uk
[2]World Health Organization, Geneva, burtona@who.int

Abstract. WUENIC is a rule-based system implemented as a logic program, developed by WHO and UNICEF for estimating global, country by country, infant immunization coverage. It possesses many of the characteristics of rule-based legislation, facilitating decisions that are consistent, transparent and replicable. In this paper, we focus on knowledge representation and problem-solving issues, including the use of logical rules versus production rules, backward versus forward reasoning, and rules and exceptions versus argumentation.

Keywords: WUENIC, rules and exceptions, logic programming, argumentation

1 Introduction

Rule-based law has many characteristics in common with logical reasoning, facilitating decision-making that is consistent, transparent and replicable. In this paper, we describe and discuss a rule-based system for estimating infant immunization coverage for 194 countries and territories.

The system, WUENIC (WHO and UNICEF Estimates of National Immunization Coverage), developed by WHO and UNICEF, is implemented in pure Prolog, and was partly inspired by the representation of the British Nationality Act as a logic program [1]. However, whereas the representation of the British Nationality Act required only the formalisation of an existing body of legal rules, the development of WUENIC involved the additional feature of further developing and refining the rules themselves. Not surprisingly, we found that the process of developing and testing the formalisation greatly assisted the process of rule development and refinement.

During the project, the working group preparing the estimates also explored the use of production rules and argumentation as alternative knowledge representation approaches. In this paper, we compare these approaches with our use of logic programming in the context of the WUENIC application. We also compare backward and forward reasoning as alternative ways of executing the WUENIC program.

[1] The author is a staff member of the World Health Organization. The author alone is responsible for the views expressed in this publication, and they do not necessarily represent the decisions, policy or views of the World Health Organization Department of Immunization, Vaccines and Biologicals.

1.1 WUENIC

Since 2000 the World Health Organization (WHO) and the United Nations Children's Fund (UNICEF) have made annual estimates of national infant immunization coverage for selected vaccines [2]. Among other uses, these estimates are used to track progress towards the Millennium Development Goal 4 of reducing child mortality. They are also used as an information resource by the GAVI Alliance (formally known as the Global Alliance for Vaccines and Immunisation) for funding immunisation in developing countries.

The WHO/UNICEF estimates are based on reports from national authorities and are supplemented with results from nationally representative surveys. In addition, local staff, primarily national immunization system managers and WHO/UNICEF regional and national staff, are also consulted for additional information that might influence or bias these two primary sources of empirical data.

Given this data and additional information, the WHO/UNICEF working group uses a set of rules to derive its official estimates, reconciling any inconsistencies in the data. Until we began formalizing the rules in the summer of 2009, the informal rules were imprecise and sometimes applied inconsistently.

In the summer of 2009, Anthony Burton, who had little previous experience of programming in Prolog and had read about the representation of the British Nationality Act as a logic program [1], approached Robert Kowalski to discuss the possibility of applying similar knowledge representation methods to the informal WHO/UNICEF rules. By May 2010, the working group had formalized a major portion of the rules, and Anthony Burton had implemented a Prolog program (called WUENIC). Since then, the program has been used to help produce the annual estimates of immunization coverage for the years 2009 and 2010 [11].

Prolog is a logic programming language with non-logical features that are primarily used for efficiency purposes. WUENIC is implemented in pure Prolog, without any of these non-logical features.

1.2 Logic Programs

Logic programs are a simplified form of logic, in which knowledge is expressed as conditional sentences of the form ***if*** *condition(s)* ***then*** *conclusion,* or equivalently in the form *conclusion* ***if*** *condition(s).* Such conditionals (also called *implications, clauses* or *rules*) combine an atomic *conclusion* with a conjunction of *conditions.*

The very first sentence of the British Nationality Act is expressed in an English style that is close to logic programming form:

> 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.

One difference between the English sentence and its logic programming form is that the English sentence inserts the logical conditions “born in the United Kingdom after commencement” into the middle of the logical conclusion “a person shall be a British citizen”. In logic programming style, the same sentence might be written in the form:

> *a person X acquires british citizenship by subsection 1.1 at time T*
> ***if*** *X is born in the uk at time T*
> ***and*** *T is after commencement*
> ***and*** *Y is a father of X or Y is a mother of X*
> ***and*** *Y is a british citizen at time T or Y is settled in the uk at time T*

Here *X, Y* and *T* are *universally quantified variables*, meaning that the conditional holds for *all* values of the variables *X, Y* and *T.*

To use the conditional to determine whether a person acquires citizenship by subsection 1.1, it is necessary to provide additional conditionals defining such notions as being *settled in the uk* and to supply additional facts. A *fact* is an atomic sentence without variables and without qualifying conditions, such as *Mary is a person* and *John is the father of Mary*. Mathematically, it is convenient to regard a fact as a conditional in which there are no variables and the number of conditions is zero.

Logic programs are commonly characterized as a *declarative representation*, in which “knowledge” is expressed by declarative sentences. However, they can also be used as a *procedural representation*, to express goal-reduction procedures of the form ***to solve*** *conclusion* ***solve*** *condition(s)*. Such goal-reduction procedures are obtained by applying *backward reasoning* to sentences of the form *conclusion* ***if*** *condition(s),* attempting to show that the *conclusion* holds by showing that the *condition(s)* hold.

For example, backward reasoning applied to the logic programming form of the first sentence of the British Nationality Act turns it into the goal-reduction procedure:

> ***to show*** *that X acquires british citizenship by subsection 1.1 at time T*
> ***find*** *T such that X is born in the uk at time T*
> ***and show*** *T is after commencement*
> ***and find*** *Y such that Y is a father of X or Y is a mother of X*
> ***and show*** *Y is a british citizen at time T or Y is settled in the uk at time T*

Such backward reasoning with conditionals gives logic programming the capabilities of a high-level, procedural programming language, making the use of other, more conventional programming languages theoretically unnecessary[2].

Backward reasoning is an *analytical* form of reasoning, which reduces concepts to sub-concepts. It contrasts with forward reasoning, which is s*ynthetic*, and which generates new concepts from existing concepts. We will see later in the paper that the distinction between backward and forward reasoning was an important issue in the development of WUENIC.

[2] This is why Prolog programs are written condition-first in the form *conclusion :- conditions*, because this syntax is neutral with respect to the declarative and procedural interpretations.

1.3 Rules and Exceptions

In normal logic programming, the conditions of a clause are a conjunction of atomic predicates and negations of atomic predicates. For example:

> *The Secretary of State may by order deprive a person of a citizenship status by subsection 40.-(2)* ***if*** *the Secretary of State is satisfied that deprivation is conducive to the public good,* ***and*** *he is* **not** *forbidden from depriving the person of citizenship status by subsection 40.-(2).*
> *The Secretary of State is forbidden from depriving a person of citizenship status by subsection 40.-(2) if he is satisfied that the order would make the person stateless.*

Here the first sentence represents a general rule, and the second sentence represents an exception to the rule. The negative condition of the first sentence qualifies the general rule, so that the rule does not apply to exceptions. Taken together, the rule and exception intuitively imply:

> *The Secretary of State may by order deprive a person of a citizenship status by subsection 40.-(2)* ***if*** *the Secretary of State is satisfied that deprivation is conducive to the public good,* ***and*** *he is* **not** *satisfied that the order would make the person stateless.*

This implication has the abstract form:

From *A if B and* **not** *E* and *E if C* infer *A if B and* **not** *C*

which is contrary to the laws of classical logic. In classical logic, it would be necessary to state the rule, exception and inference in the (arguably) less natural form:

From *A if B and* **not** *E* and **not** *E if* **not** *C* infer *A if B and* **not** *C.*

The logic programming formulation, in contrast with classical logic, interprets negative conditions of the form ***not*** *E* as *negation as failure*:

not *E* holds if and only if ***E*** fails to hold.

Hundreds - if not thousands – of research publications have sought to provide semantics for this form of negation. In the WUENIC project, we have found it useful to interpret negation as failure in terms of the argumentation semantics of Dung [4]:

not *E* holds if and only if
every attempted counter-argument to show that ***E*** holds fails.

Dung's semantics can also be applied directly to rules and exceptions expressed more informally in ordinary English. For example, in the British Nationality Act, the rule and exception are actually expressed in the potentially contradictory form:

> 40.-(2) The Secretary of State may by order deprive a person of a citizenship status if the Secretary of State is satisfied that deprivation is conducive to the public good.
>
> 40.-(4) The Secretary of State may not make an order under subsection (2) if he is satisfied that the order would make a person stateless.

In classical logic, these two sentences have the form *A if B* and ***not*** *A if C*, which is inconsistent if both *B* and *C* hold.

The argumentation semantics avoids the inconsistency by making it possible to assign different priorities to arguments constructed by means of general rules and to arguments constructed by means of exceptions. In the argumentation semantics, an argument supported by a general rule *A if B* can be attacked by a counter-argument supported by an exception ***not*** *A if C*, but an argument supported by the exception cannot be counter-attacked by an argument supported by the general rule.

In WUENIC, we found it easier to represent rules and exceptions using negation as failure with its argumentation semantics, than to use argumentation applied directly to ordinary, informal English. The use of negation as failure made it easy, in turn, to represent rules and exceptions directly in Prolog.

1.4 Argumentation

In Dung's argumentation, whether an argument succeeds or fails depends on whether or not it can defeat all counter-arguments by recruiting support from defending arguments. The following example is from [3]:

Rule 1:	All thieves should be punished.
Rule 2:	Thieves who are minors should not be punished.
Rule 3:	Any thief who is violent should be punished.

Suppose that John is a thief who is a violent minor. Then by rule 1, there is an argument that he should be punished; by rule 2, a counter-argument that he should not be punished; and by rule 3, another argument that he should be punished.

In Dung's theory of argumentation, the second argument attacks the first, and the first and third arguments attack the second. Together, the first and third argument succeed, because for every counter-argument (the second one) that attacks one of them, there is at least one argument (in fact two arguments, the first and third) that counter-attacks and defeats it.

The second argument also succeeds, because for every counter-argument (the first and third) that attacks the second argument, there exists an argument (the second argument itself) that counter-attacks and defeats them both.

However, this analysis in terms of arguments fails to do justice to the intuition that the first rule is a general rule, the second rule is an exception to the general rule, and the third rule is an exception to the exception. Arguments constructed by means of an exception to a rule attack arguments constructed by means of the rule, but not vice versa. So in this case, only the first and third argument, taken together, should succeed, and the second argument should fail. Dung's theory accommodates this intuition, because it allows attack relations to be specified abstractly and arbitrarily.

In argumentation, the attack relations between arguments and the resulting success or failure of arguments is embedded both in the semantics and in the proof theory of the logic of argumentation. It is not represented explicitly in the representation of the arguments themselves. In contrast, in normal logic programming, the attack and defeats relations are represented explicitly by negative conditions in the rules. Here is one such representation, applied to the rules for punishing thieves:

a person should be punished
if *the person is a thief*
and *the person is* **not** *an exception to the punishment rule.*

a person is an exception to the punishment rule
if *the person is a minor*
and *the person is* **no***t an exception to the exception to the punishment rule.*

a person is an exception to the exception to the punishment rule
if *the person is violent.*

This representation has the advantage that additional exceptions can be catered for by adding additional clauses, without changing the rule. For example:

a person is an exception to the punishment rule
if *the person is* **not** *of full mental capacity.*

Dung's argumentation semantics of logic programs with negative conditions [5, 6] has practical significance, because it means that explanations of conclusions derived by means of logic programs can be expressed in argumentation terms.

1.5 Production Rules

In Artificial Intelligence, the notion of *rule* has two different interpretations: as a conditional in logic programming, and as an expression of the form ***if*** *conditions* ***then*** *actions* in production systems. Both kinds of rules have a similar ***if-then*** syntactic form. But rules in logic programming have both a declarative, logical semantics and an "operational semantics" in terms of backward and forward reasoning. Production rules, on the other hand, do not have a declarative semantics, and have an "operational semantics" in terms of state transitions determined by means of *actions*.

Production rules of the form ***if*** *conditions* ***then*** *actions* are executed by checking whether the *conditions* hold in the current state, and, if they do, then performing the *actions* to transform the current state into a new one. This execution strategy resembles and is sometimes confused with forward reasoning.

The confusion between logic programming rules and production rules was a factor in our discussions about knowledge representation in WUENIC. It also helped to motivate the first author to investigate the relationship between the two kinds of rules in greater detail [8, 9]. In the case of WUENIC, we decided to focus our efforts on formalising the rules in normal logic programming form, which had the advantage that they could then be translated directly into pure Prolog.

2 WUENIC

The WHO/UNICEF working group had been applying an informal set of rules [2], before starting the project in the summer of 2009. At first, the most natural formalization seemed to be in terms of production rules. For example:

if *for a Country, Vaccine and Year, there are nationally reported data*
and *there is* ***no*** *survey data*
then *the WUENIC estimate* **is** *the reported data.*

if *for a Country, Vaccine and Year, there is nationally reported data*
and *there is survey data*
and *the survey data is within 10% of the reported data*
then *the WUENIC estimate* **is** *the reported data.*

if *for a Country, Vaccine and Year, there is nationally reported data*
and *there is survey data*
and *the survey data is* **not** *within 10% of the reported data*
then *the WUENIC estimate* **is** *the survey data.*

It was natural to think of the ***then*** part of the rule as an action. But it is also possible to think of the same rules in more logical terms, where ***is*** in the conclusion is equality, and the *WUENIC estimate*, the *reported data* and the *survey data* are functions that take *Country, Vaccine* and *Year* as input and return percentages as output.

2.1 Functions Versus Relations

In logic programming, input-output functions are represented as relations, as in relational databases. For example, instead of representing the mother of a person as the output of a motherhood function:

mother-of(john) = mary.

the function is represented as a relation:

mother(john, mary).

Instead of using equality (or ***is***) to define a new function, as in:

parent(X)= {*Y* | *mother-of(X) = Y* or *father-of(X) = Y*} a new relation is defined using the logical connective ***if***:

> *parent(X, Y)* ***if*** *mother(X, Y)*
> *parent(X, Y)* ***if*** *father(X, Y)*

In ordinary mathematical logic, definitions are usually represented by means of equivalences (***if and only if***). But in logic programming, definitions are represented by means of the ***if*** halves of equivalences, and the ***only-if*** halves are implicit.

Treating functions as relations in this way, the first production rule above becomes the logic programming rule:

> ***if*** *the nationally reported data for a Country, Vaccine and Year is R*
> ***and*** *there is* **no** *survey data S for the Country, Vaccine and Any-Year*
> ***then*** *the WUENIC estimate for the Country, Vaccine and Year is R.*

In Prolog syntax, this is written in the form:

```
wuenic(Country, Vaccine, Year, R) :-
reported(Country, Vaccine, Year, R),
not(survey(Country, Vaccine, Any-Year, S)).
```

where the variables Country, Vaccine, Year, R are universally quantified, but the condition not(survey(Country, Vaccine, Any-Year, S)) means there does not exist Any-Year and S such that survey(Country, Vaccine, Any-Year, S).

Whereas true production systems execute production rules only in the forward direction, from condition to actions, Prolog executes logic programs by backward reasoning, from conclusion to conditions. However, the declarative semantics of logic programs is independent from the manner in which they are executed, and in theory logic programs can be executed by backward or forward reasoning.

2.2 The General Structure of the Rules

After we decided to formalize the WUENIC rules in logic programming form and to implement them in Prolog, we turned our attention to the overall structure of the rules.

The same rules apply to all countries, and are applied to individual countries without reference to other countries. A single run of the rules, for a given country, produces estimates for all years in the period from 1997 to the current year and for all vaccines for which the country is required to report coverage. Estimates are produced for the entire period, because the estimates for previous years might be influenced by new information, such as a new survey, obtained in the current year.

We decided to organize the rules into four levels. The first three levels produce preliminary estimates for the individual vaccines. The fourth level reconciles any discrepancies between vaccines.

Level one. The primary data, consisting of the nationally reported immunisation coverage and any nationally representative surveys, are evaluated separately, and if necessary are ignored or adjusted. For example, the reported data are adjusted downward if they are over 100%, which can happen if the reporting authorities use a number for the target population that is smaller than the actual number of children vaccinated. Survey data may be ignored if the sample size is too small.

Level two. If data are available from only a single source (national reports or surveys) for every year for the given country and vaccine, then the estimates are based on that source alone. Otherwise, the estimates are made at "anchor years", which are years in which there are both reported data and survey data. If the (possibly adjusted) survey data do not support the (possibly adjusted) reported data, then the (possibly adjusted) survey data override the reported data, and are used as the WUENIC estimate, unless there is a compelling reason to do otherwise. A compelling reason might be an exception represented by another, overriding rule, or as we will see later by a "fact" representing a working group decision.

Level three. Estimates at non-anchor years, for which there are only reported data, but not survey data, are influenced by the estimates at the nearest anchor years. If the estimates at the nearest anchor years are based on the (possibly adjusted) reported data, then the estimate at the non-anchor year is similarly based on the reported data. But if the estimates at the nearest anchor years are based on the (possibly adjusted) survey data, then the estimate at the non-anchor years is based on the reported data, but calibrated to the level of the estimates in the anchor years.

Level four. The resulting estimates for the different vaccines are cross-checked for consistency between related vaccines, and adjustments are made if necessary. For example, the estimates for the first and third doses of the same vaccine are compared, to ensure that the estimate for the third dose is not higher than the estimate for the first dose. This reflects the fact that every child who receives a third dose of a vaccine must have received a first dose.

2.3 Two ways of Logically Representing the Problem of Going from *a* to *z*

It is natural to interpret the four levels as four phases, in which the goal of producing the estimates for a given country is reduced to the consecutive subgoals of:

first evaluating and possibly adjusting the data,
then producing preliminary estimates at anchor years,
then producing preliminary estimates at non-anchor years, and
finally making possible adjustments for estimates of related vaccines.

This goal-reduction procedure, in turn, has a natural interpretation as an application of backwards reasoning to a logical conditional:

if *for a Country, the data are evaluated and possibly adjusted*
and *the possibly adjusted data are used to produce*
preliminary estimates at anchor years
and *the preliminary estimates at anchor years are used to produce*
preliminary estimates at non-anchor years
and *the preliminary estimates of related vaccines are*

compared and possibly adjusted
***then** the WUENIC estimates for the Country are*
the resulting possibly adjusted estimates

However, this representation clashes with the natural way of representing the individual rules within the different levels. It took us some time to diagnose the problem, and longer to solve it.

The problem, discussed at length in Chapter 4 of [12], is that there are two ways of logically representing the problem of going from *a* to *z*. The first representation is analogous to the treatment of levels as phases:

***if** you can go directly from X to Y **then** you can go from X to Y*
***if** you can go directly from X to Y **and** you can go from Y to Z*
***then** you can go from X to Z*

With this representation, given a set of direct connections represented as facts, say, of the form *you can go directly from a to b*, the problem of going from *a* to *z* is represented by the goal of showing:

you can go from a to z

The second representation is analogous to the natural way of representing the individual rules. In this representation, a direct connection, say, from *a* to *b* is represented by a rule:

***if** you can go to a **then** you can go to b*

With this representation, given the set of direct connections represented as rules, the problem of going from *a* to *z* is represented by:

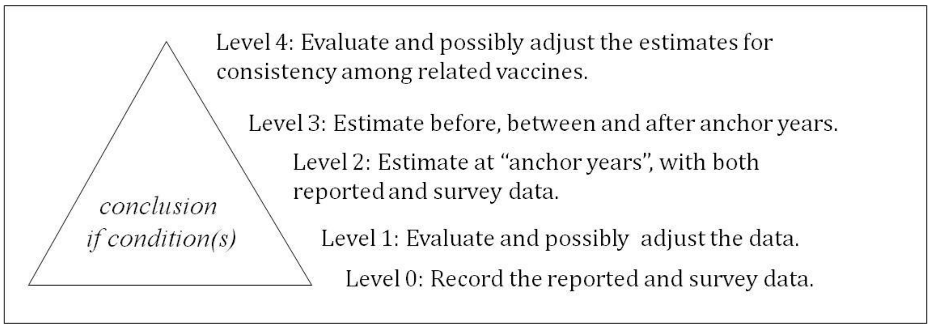
***assuming that** you can go to a **and showing that** you can go to z*

The problem of going from *a* to *z* is typical of a wide class of problems, including the WUENIC problem of generating immunization estimates.

In general, the two representations have both a declarative and a procedural interpretation. But the first representation is lower-level, and easier for the programmer to control. The second representation is higher-level, and easier for non-programmers to understand. We decided to use the second representation, but did not anticipate the control problems that would arise with the Prolog implementation.

2.4 Forward Versus Backward Reasoning

With the WUENIC program written in this way, the problem of generating the immunization estimates from the given initially reported and survey data can be viewed as the problem of filling in a triangle:


Level 4: Evaluate and possibly adjust the estimates for consistency among related vaccines.
Level 3: Estimate before, between and after anchor years.
Level 2: Estimate at "anchor years", with both reported and survey data.
conclusion
if condition(s)
Level 1: Evaluate and possibly adjust the data.
Level 0: Record the reported and survey data.

The conclusions of the rules represent predicates at higher levels of the triangle, and the conditions of the rules represent predicates at lower levels. Forwards reasoning with the rules fills in the triangle from the bottom up, treating the successive levels as phases. Backward reasoning, as in Prolog, fills in the triangle from the top down.

In theory, because we had written the rules purely declaratively, it should not have mattered whether the rules were used to reason forwards or backwards, and indeed that was the case. The rules gave the intended results even though they were used to reason backwards and to fill in the triangle top down. But they were impossibly inefficient, taking about 20 minutes to produce the estimates for a single country. This was longer than we could afford within the schedule of the working group meetings.

At this point, most novice Prolog programmers would probably have thrown up their hands in despair, discarded the Prolog implementation, and perhaps even reprogramed the rules in a production system language, which would run the rules more naturally and more efficiently in the forward direction. A more experienced Prolog programmer, on the other hand, might have rewritten the program at a lower level, analogous to the first representation of the problem of going from *a* to *z*. However, we were able to preserve the naturalness of the representation and reduce the execution time to about 30 seconds, by changing only a few lines of code.

The inefficiency of backward reasoning in this case is due to the repeated recalculation of the lower-level sub-goals. For example, if there are 10 years between two adjacent anchor years, backward reasoning recalculates the estimates at the two anchor years ten times. The Prolog solution is to cut the program in half, run the level two goals first, assert their solutions into the Prolog database, and then run the level four rules, accessing the asserted level two solutions without re-computing them.

The alternative and preferable solution, is either to execute the rules in the forward direction, or to execute them in the backward direction, but save the solution of sub-goals automatically, so they do not need to be re-solved later. The latter of these two alternatives is provided by Prolog systems, such as XSB Prolog, which use the technique of tabling [7] to generate the solution of every subgoal only once. We transported the WUENIC system to XSB Prolog soon after diagnosing the problem.

2.5 Rule Refinement and Working Group Decisions

The WUENIC rules are analogous both to a collection of legal rules and to an expert system. As with other expert systems, it was useful to develop the rules by successive refinement. We developed the first collection of rules to cover the most commonly occurring cases, and then refined the rules when they did not deal satisfactorily with more complicated cases. In theory, this process of refinement is never-ending, since the rules can always be improved. But in practice, we needed to firm up the rules, to make it easier to communicate them and to apply them in practice.

Partly as a compromise between facilitating future refinement of the rules and of finalizing them as early as possible, we introduced the notion of working group decisions, represented by "facts, which can be changed from one run to another without changing the rules themselves. In theory, any rule can by overridden by a working group decision, similarly to the way in which an exception overrides a general rule. This is represented by adding an extra negative condition to the rule and by adding an extra rule if necessary. For example:

if *the nationally reported data for a Country, Vaccine and Year is R*
and *there is* ***no*** *survey data S for the Country, Vaccine and Any-Year*
and *there is* ***no*** *working group decision to assign an estimate W*
for the Country, Vaccine and Year
then *the WUENIC estimate for the Country, Vaccine and Year is R.*

if *there is a working group decision to assign an estimate W*
for a Country, Vaccine and Year
then *the WUENIC estimate for the Country, Vaccine and Year is W.*

Working group decisions are added as "facts" to the Prolog database. The Prolog implementation combines these facts with other facts, applies the rules, and derives the WUENIC estimates. In practice, we found it useful to use working group decisions interactively, judging the quality of the resulting estimates, and repeating the process with modified decisions, until we are satisfied with the results.

Working group decisions are a convenient way of representing decisions in cases where there are no well-defined rules, or where is not convenient to state the rules explicitly. In both cases, they are analogous to the powers of discretion given to the Secretary of State in certain provisions of the British Nationality Act. For example:

> 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 fulfils 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.

Here the word, "may" seems to suggest the need for a probabilistic or modal logic. In fact, it is just a way of emphasizing that the conclusion depends upon the decision of the Secretary of State. In logic programming form, the provision can be expressed without probability or modality in the form:

the secretary of state ***will*** *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*
fulfils 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 positive condition *the secretary of state thinks fit to* ***grant*** *the person a certificate of naturalization* can also be written as a negative condition *the secretary of state does* ***not*** *think fit to* ***withhold*** *a certificate of naturalization from the person*, analogously to the way we represent working group decisions in WUENIC.

3 Discussion and Conclusions

The WUENIC approach builds upon the well-established use of logic programming for representing and reasoning about rules in legal documents. It also has many features in common with rule-based expert systems and business rule applications [8].

The confusion between logic programming rules and production rules became an early issue in the development of WUENIC. This affected not only the representation of the rules and the choice of implementation language, but it also added to the motivation of the first author to attempt to clarify the relationship between the two [9]. It also contributed to the development of the logic-based production system language LPS [10], which combines logic programming and production systems in a logic-based framework. Roughly speaking, in LPS, logic programs are used, as in WUENIC, to define concepts that are needed for decision-making in an organisation, whereas production rules are used to generate associated actions and workflows.

In addition to considering different kinds of rules, we also considered rule-based versus argumentation-based representations and implementations. Here the relationship between logic programming and Dung's argumentation semantics is already well understood [5], and it is relatively easy to translate one representation into the other. As a consequence, it is possible to implement an argumentation approach using logic programming rules, and to use the relationship to explain the resulting conclusions in argumentation terms.

In Dung' argumentation, an argument succeeds if it can be extended to a set of defending arguments that collectively defeats every argument that attacks any argument in the defending set. It suffices for the defending set to contain only one such defeating argument for every attacking argument. However, in the WUENIC application, arguments can attack and defeat one another to varying degrees, and the more supporting and defeating arguments there are the better.

This broader kind of argumentation in the WUENIC application means that WUENIC estimates have varying degrees of confidence and uncertainty. We are currently exploring the problem of representing and reasoning with uncertainty.

In addition to the problem of understanding the relationships between different knowledge representation and problem solving paradigms, we had greater than expected problems with our chosen logic programming paradigm. We did not anticipate the difficulties of choosing between different logic programming representations and the problems of improving the efficiency of our preferred representation. On the one hand, this highlights the difficulties that other implementers may face with other applications. On the other hand, it draws attention to the potential of tabling [7] to overcome some of the problems of programming in Prolog, and to facilitate its wider application in the future.

Acknowledgments. We are pleased to acknowledge the support and contributions and of the other members of the Working Group: Marta Gacic-Dobo from WHO and David Brown and Rouslan Karimov from UNICEF. Thanks also to the anonymous referees for their comments, which helped to improve the paper.

References

1. Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., Cory, H. T.: The British Nationality Act as a Logic Program, CACM, 29(5) 370-386 (1986)
2. Burton A, Monash R, Lautenbach B, Gacic-Dobo M, Neill M, Karimov R, Wolfson L, Jones G, Birmingham M.:WHO and UNICEF Estimates of National Infant Immunization Coverage: Methods and Processes. Bull. World Health Organization. 87, 535–541 (2009)
3. Prakken, H., Sartor, G.: A Dialectical Model of Assessing Conflicting Arguments in Legal Reasoning. Journal of Artificial Intelligence and Law, 4(3-4) (1996)
4. Dung, P. M.: On the Acceptability of Arguments and its Fundamental Role in Non-monotonic Reasoning, Logic Programming and N-person Games. Journal of Artificial Intelligence, 77, 321-357 (1995)
5. Bondarenko, A., Dung, P. M., Kowalski, R., Toni, F.: An Abstract Argumentation-theoretic Approach to Default Reasoning. Journal of Artificial Intelligence 93(1-2), 63-101 (1997)
6. Dung, P. M., Kowalski, R., Toni, F.: Dialectic proof procedures for Assumption-Based, Admissible Argumentation. Journal of Artificial Intelligence 170(2), 114-159 (2006)
7. Sagonas, K., Swift, T., Warren, D. S.: XSB as an Efficient Deductive Database Engine. SIGMOD Rec. 23(2) 442-453 (1994)
8. Benjamin N. Grosof, Yannis Labrou, Hoi Y. Chan.: A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In: Proceedings of the 1st ACM conference on Electronic commerce. 68-77. ACM, New York, New York (1999)
9. Kowalski R.: Computational Logic and Human Thinking: How to be Artificially Intelligent. Cambridge University Press, Cambridge (2011)
10. Kowalski, R., and Sadri, F.: Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents, In Web Reasoning and Rule Systems (eds. A. Polleres and T. Swift) Springer, LNCS 5837 (2009)
11. Brown DW, Burton A, Gacic-Dobo M, Karimov RI.: A Summary of Global Routine Immunization Coverage through 2010. Open Infect Dis J. 5:115-117 (2011)
12. Kowalski R.: Logic for Problem Solving, Elsevier Science Publishing Company (1979). Also at http://www.doc.ic.ac.uk/~rak/