IMMGNOSIS: KNOWLEDGE ENGINEERING FOR A STATELESS WEB-BASED EXPERT SYSTEM FOR IMMIGRATION LAW

by

DANIEL M. DE JUAN

(Under the Direction of Walter D. Potter)

ABSTRACT

In this thesis, I document practical and theoretical issues concerning the development of IMMGNOSIS, a stateless Web-based expert system that reasons over matters involving U.S. immigration law. I focus on the knowledge engineering aspects of the IMMGNOSIS project, detailing knowledge acquisition, knowledge representation, and inference. Additionally, I present a modified expert-system shell that efficiently handles multiple consultations in a stateless, Web-based environment while relying on only a single instance of the inference application. I evaluate the accuracy of the system's diagnoses, the performance of its stateless architecture, and the potential benefit of putting it into practical use. Finally, I present future plans for knowledge-base expansion and intelligent handling of conflict resolution.

INDEX WORDS: LEGAL EXPERT SYSTEM, STATELESS ARCHITECTURE, STATELESS INFERENCE, IMMIGRATION LAW, EXPERT SYSTEM SHELL, RULE-BASED REASONING, DEFEASIBLE REASONING

IMMGNOSIS: KNOWLEDGE ENGINEERING FOR A STATELESS WEB-BASED EXPERT SYSTEM FOR IMMIGRATION LAW

by

DANIEL M. DE JUAN

AB, University of Georgia, 2005

A Thesis Submitted to the Graduate Faculty of the University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2005

IMMGNOSIS: KNOWLEDGE ENGINEERING FOR A STATELESS WEB-BASED EXPERT SYSTEM FOR IMMIGRATION LAW

by

DANIEL M. DE JUAN

ACKNOWLEDGEMENTS

I would first and foremost like to acknowledge Dr. Donald Nute for taking a relentlessly active role in advising this project despite his recent graduation into retirement. Thanks also to Walter Potter and Khaled Rasheed for sitting on my committee. In addition to my academic advisors, I would also like to acknowledge Blair Dorminey for funding the research documented herein and for providing domain experts from his staff at Dorminey and Cox, LLC. Many thanks also to those experts, David Tooley, Aschalew Nigussie, and Jongnam Park, for the many hours they spent trudging through and interpreting INA statutes. Finally, I would like to thank my research partner and close friend, Vineet Khosla, who has worked tirelessly with me through every step of this endeavor.

PREFACE

This thesis documents a team research project conducted through a partnership between the Artificial Intelligence Center at the University of Georgia and Dorminey and Cox, LLC. I have worked with Vineet Khosla, a fellow MSAI student, whose expertise in JSP has been an invaluable asset. Khosla and I are writing our theses in tandem and striving to each cover only those aspects of the development process in which we have had direct influence. In doing so, we aim to produce distinct discussions, but there is some unavoidable overlap due to the fact that we have advised and inspired one another throughout the undertaking of this endeavor. I make a concerted effort to cover work that is uniquely his only to the extent that it is necessary for an understanding of the machinery of the complete system, and I reference his thesis in all such cases. Khosla's thesis focuses primarily on issues pertaining to the delivery of the IMMGNOSIS expert system over the Web in a stateless environment. He discusses issues concerning user interface, handling of multiple user sessions on a single server, and bridging the gap between Prolog and JSP. I focus on the expert system application itself, covering issues involving knowledge representation and engineering, stateless inference, and conflict resolution. However, in order to provide a self-contained discussion of inference in a stateless architecture, I am forced to provide some discussion of that architecture. As I have mentioned, I will keep my coverage of architectural issues to a minimum and will refer the reader to Vineet's thesis at any point where such discussion arises.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 EXPERT SYSTEMS AND LAW

Expert systems are the subset of knowledge-based systems[1] that reason over information pertaining to domains of human expertise in order to solve problems within those domains [1]. The traditional architecture of an expert system includes: a knowledge base where all domain knowledge is stored, an inference engine that reasons over the information in the knowledge base, an explanatory facility that displays the reasoning used to derive conclusions, a working memory that holds data and intermediate results, and a user interface through which a user communicates with the expert system [2]. These features provide the necessary means for modeling the reasoning process that guides a human expert from a given set of conditions to a recommendation.

Two quintessential domains of human expertise are medicine and law. Not surprisingly, these have been popular domains for expert system applications. MYCIN, one of the oldest and arguably the most famous of all expert systems, was designed at Stanford University in the 1970s to diagnose infectious blood diseases [3]. PUFF, another notable medical expert system developed at Stanford, interprets data related to various pulmonary diseases [4]. These systems, as well as most other medical expert systems, are intended to analyze raw data and provide a diagnosis without involving a human expert. When doctors have a set of symptoms and test results available to them, it is their familiarity with domain knowledge that enables them to

[1] Knowledge-based systems are functionally and architecturally equivalent to expert systems, but lack the requirement that knowledge bases must pertain to expert domains.

immediately derive a conclusion, and this expertise is what medical expert systems emulate. In the domain of law, however, expertise is of a different sort. It does not require familiarity with all existing legislation that applies to a particular legal domain. Rather, the essence of legal expertise is knowing how to interpret those statutes and use them to support an argument in a court of law. When presented with a case, even a capable expert must engage in a tedious process of researching legislation to find applicable statutes. Hence, legal expert systems have a somewhat different goal from medical expert systems. They are not intended to replace human experts, but rather to assist them by reducing the time they spend researching cases. Since this task involves objective analysis of the properties that comprise a client's case, it is well-suited to the application of an expert system. While a legal expert cannot typically memorize all of the legislation that pertains to his domain of practice, an expert system can. Thus, legal expert systems are practically valuable in that they provide a way of automating the research required for preliminary analysis of a case. However, once a system picks out appropriate statutes and infers a diagnosis, it is up to the attorney to exercise his or her expertise by arguing that diagnosis in court.

1.2 PROBLEM SPECIFICATION: IMMIGRATION LAW AS AN EXPERT SYSTEM DOMAIN

IMMGNOSIS is an expert system that reasons over U.S. immigration law, the body of knowledge specified in the Immigration and Nationality Act (INA). It can currently provide an initial determination of a subject's U.S. citizenship status, eligibility for naturalization, and admissibility status, and a visa recommendation module is currently under development [6]. The citizenship module determines whether a subject is a U.S. citizen at the time of the consultation.

Citizenship is desirable because of the rights that it affords, namely, the right to vote, the right to hold a U.S. passport (which is more widely accepted than most others), and the right to benefit from Medicare and social security. In most cases, a person attains U.S. citizenship either by being born in the United States or by being born to a U.S. citizen in a foreign country. These cases, however, are of little interest from a legal expert system standpoint because they do not typically require litigation, and they certainly do not justify the programming overhead associated with implementing an expert system. However, the more elusive rules—for example, those involving children born to unmarried parents, children born in outlying U.S. territories, and children born of unknown parentage—make the application of an expert system appropriate. For each of the above cases, the INA specifies a multiplicity of relevant statutes. Each statute further specifies a set of conditions which, if satisfied, legitimate a subject's claim to citizenship. Each statute typically corresponds to a range of birth dates. Since immigration legislation changes over time in the direction of increasing restrictiveness, the conditions in a particular statute tend to be more difficult to satisfy when the subject's date of birth is relatively recent.

The naturalization module is related to the citizenship module, with the difference being that instead of telling whether a subject is already a citizen, it determines whether he or she is in a position to become one, or naturalize. In the typical case, a person qualifies for naturalization by satisfying each of the following criteria: having had legal-permanent-resident status for some specified duration (described below), being physically present in the United States for some specified duration, and not being disqualified on the grounds of immoral character, threatening political ties, or any of a host of other disqualifying properties. As with the citizenship module, however, there is a vast set of rules that apply to special cases of naturalization, including marriage to a U.S. citizen, service in the armed forces, involvement with promoting U.S.

interests abroad, and so forth. Again, these less common cases complicate the problem and justify the implementation of an expert system.

The admissibility and visa recommendation modules are closely related to one another in that admissibility is a prerequisite for any visa application. Admissibility means, quite simply, that a person is allowed to enter the U.S. *supposing* that he or she has a visa or other means of entering legally. Admissibility is typically assumed, so the admissibility module actually tries to prove that a subject is *in*admissible. Admissibility can be lost by the subject's committing a crime of moral turpitude, being deemed a threat to national security, presenting fraudulent immigration documents, etcetera. Once a person has secured his or her admissibility, he or she can apply for an immigrant or non-immigrant visa, documents that allow a person to remain in the United States for an unlimited or limited period of time, respectively. Temporary, non-immigrant visas are typically issued for work, family visitation, or tourism, and the like. Immigrant visas result in legal-permanent-resident status, which grants a subject the right to remain in the United States indefinitely and constitutes the first step toward becoming a naturalized citizen. Citizenship, naturalization, admissibility, and visa attainment are closely related issues, and in some cases are prerequisites for one another. Chapter 3.2 describes the way in which IMMGNOSIS handles this interdependency of subdomains.

1.3 INTRODUCTION TO IMMGNOSIS: A STATELESS WEB-BASED EXPERT SYSTEM FOR IMMIGRATION LAW

IMMGNOSIS has two potential practical applications, either as an in-house research tool for immigration law firms or as a publicly-accessible tool for potential clients. From a business perspective, both applications have a potential for generating revenue for the proprietor, the first through subscriptions and the second by providing referrals to firms that register with the service. In either case, the system requires a centralized, Web-based architecture. That design gives the proprietor control over system access and enables him or her to provide immediate system updates while containing the proprietary knowledge in a single location [6].

When employed as an in-house research tool, the system replaces the traditional preliminary consultation, thereby removing the need for costly man-hours spent during the initial evaluation phase of each new case. Applied this way, the system decreases research time not only by automatically analyzing the worth of pursuing a case, but also by providing specific justifications from statutes in the Immigration and Nationality Act which could be used to support the case in court. In the publicly-accessible implementation, IMMGNOSIS allows a far greater number of users to access the system than if it were used strictly in-house. The system could analyze a potential client's case regardless of physical locality and without requiring interpersonal communication. If IMMGNOSIS concludes that a user's case is worth pursuing, the program can refer him or her to registered firms within a specified proximity and can store his or her consultation in a database for future reference by those firms. In either application, IMMGNOSIS allows potential clients to make an initial determination of the worth of pursuing their cases while circumventing, or at least minimizing, the financial and temporal costs associated with initiating a traditional consultation with a human attorney. It also benefits legal practitioners by automatically sorting potential cases and retrieving relevant statutes without requiring in-house research.

This thesis documents the development of the IMMGNOSIS knowledge base as well as the stateless Web-based environment in which it runs. I have worked with Vineet Khosla, a fellow student in the MSAI program at UGA, to modify XSHELL [7], a state-dependent expert system shell for Prolog. The modification, STATELESS XSHELL, enables the running of a single Prolog session capable of processing multiple consultations simultaneously over the Web. Rather than treating each consultation as a continuing session and storing up client information from beginning to end, STATELESS XSHELL limits each transaction to the acquisition of a single piece of knowledge about a user. It relies on a portable blackboard architecture wherein a user's blackboard is written to, and erased from, working memory with each query. This structure allows expert systems that run STATELESS XSHELL to process queries from multiple users in parallel without having to remember previous states. With each query, the shell unpacks the current blackboard, reasons over it, and repackages it along with a new question for the user in a Java Server Pages (JSP) file, JSP being the part of the Java technology family responsible for dynamic Web page design. The shell does not have to keep track of separate blackboards nor remember the information it has gleaned from each user.

In revamping the XSHELL expert system shell, we also focused on increasing usability by including an intuitive user interface and by allowing users to edit their responses in the middle of a consultation. These capabilities are crucial to any practical implementation because the application is intended for use by laymen lacking expertise in the fields of artificial intelligence and, in the second application discussed above, immigration law. Anyone lacking experience with expert systems will benefit from a simple user interface, and anyone lacking expertise in immigration law will appreciate the ability to edit answers to troublesome questions on the fly. Details relating to the user interface and the role that JSP plays in the modified shell are documented in Khosla's thesis [6].

The present thesis focuses primarily on the knowledge engineering and knowledge processing aspects of the IMMGNOSIS project. Chapter 2 describes the knowledge acquisition process, specifying each of the major phases involved in gleaning knowledge from domain experts and preparing it for an appropriate representation in the knowledge base. Chapter 3 presents a detailed explanation of the structure of the knowledge base that we have used to represent domain knowledge, beginning with a defense of our theoretical commitment to a rule-based approach and moving into a discussion of the specific structures we have used to represent rules and the conditions that comprise them. Chapter 4 offers a discussion of the inference engine and the means by which it operates in a stateless environment. This discussion is divided into a presentation of XSHELL, the original state-dependent shell [7], an overview of the stateless architecture of the new system, and a detailed explanation of the changes in the modified shell that enable it to perform inference in a stateless environment. Finally, Chapter 5 concludes with an evaluation of the system and a discussion of plans for future development.

CHAPTER 2

KNOWLEDGE ACQUISITION

All domain knowledge has been provided by Blair Dorminey and his staff at Dorminey and Cox, LLC. in Athens, GA, a firm that handles cases concerning various issues in immigration law. Dorminey has supported this project from its beginning by providing both funding and expertise. Vineet Khosla and I met with members of Dorminey's staff, specifically David Tooley, Aschalew Nigussie, and Yongnam Park, on a weekly basis throughout development. While each member of the staff has collaborated to some extent on the entire knowledge base, each domain expert has focused on one particular area within the domain. Tooley worked with me to develop a knowledge base for the citizenship module. Nigussie and I collaborated in developing rules for both the naturalization and admissibility modules. Park has recently begun working with Khosla to develop a knowledge base pertaining to immigrant and non-immigrant visa recommendations, a module that is currently under development [6]. As I mentioned, I have encoded most of the knowledge for the first three subdomains, and Khosla is currently working with Park on the visa module.

We focus on one subdomain at a time, beginning by specifying the major divisions within that subdomain and the properties that separate them. In citizenship, for example, a major criterion for division is the subject's place of birth. Three possibilities exist—the subject can be born within the United States, within one of its outlying territories, or outside of the United States and its territories—each of which will take the subject into separate subsets of the citizenship rule base, and we organize the progression of the knowledge acquisition process based on these subsets. Once the major divisions within a subdomain have been established, the expert responsible for that subdomain reads and interprets relevant statutes in the Immigration and Nationality Act, the federal document that contains the legislation pertaining to all immigration law. Once the expert settles on an interpretation, he translates the statutes into English rules in the form of an outline. This process of knowledge interpretation and extraction constitutes the most significant bottleneck in the knowledge engineering process.

Once the rules are established by domain experts, Khosla and I interpret them from a knowledge engineer's perspective, determining the necessary objects, properties, and relations that are required for representing them in a knowledge base. We also mark any conditions or consequents that are unclear to us or that require further interpretation. The process of converting rules from English outlines to Prolog clauses involves a continual interaction between knowledge engineers and domain experts that is driven by questioning and explication. Rules in the final knowledge base often differ significantly in their representation from the original rules. Since we represent disjunctive conditions over sets of Prolog clauses, a single statute might have several corresponding rules in the knowledge base. The following figures illustrate this discrepancy. Figure 2.1 shows a sample rule as we receive it from the domain experts. Figure 2.2 shows the corresponding rule in the IMMGNOSIS rule base. (To avoid divulging sensitive, proprietary content, both figures present the *base-case*—the most basic, run-of-the-mill example of a citizenship rule.)

The first rule in Figure 2.2 is the rule that determines the ultimate diagnosis for the citizenship base case. The following four rules define a *complex condition*, a term I discuss in Chapter 3. For now, a complex condition is just a way of specifying multiple disjunctive conditions that lead to the same conclusion using only a single condition name. That complex condition happens to be referenced by all subsequent diagnosis rules that result in a positive identification, so it provides a means of significantly limiting the size of the rules in the knowledge base and preserving a structural correlation between the coded rules and the original statutes. Refer to Chapter 3 for a detailed discussion of the implementation of complex conditions.

In order to validate the final rules, we familiarize domain experts with the representations used in the knowledge base with the hope that reading the coded rules will enable them to detect any logical errors that might have occurred during the conversion. Experts also conduct consultations throughout development, intentionally answering questions in a manner that leads them to each of the intended diagnoses for the rules they have provided.

```
I.      Citizenship at Birth

        A.      If you are born in the United States, you are a citizen (US Const.
                14th Amend).

                 i.     Exceptions

                          1.  Renunciation

                                  a.  you are not a citizen if you have renounced
                                      your citizenship

                          2.  Jurisdiction

                                  a.          You are not a citizen if you are an
                                      American Indian belonging to a tribal nation

                                  b.          You are not a citizen if you were born
                                      on US soil to a member of an occupying force

                                  c.          You are not a citizen if you are born
                                      to a diplomat in the US and have no citizen
                                      parent. (US Const. 14th Amend).
```

Figure 2.1: A BASIC CITIZENSHIP RULE AS PROVIDED BY A DOMAIN EXPERT

```
xkb_identify(1,’ref withheld’,citz,[neg_citizen]) :-
            disqualified.

xkb_identify(1,’U.S. Const 14th Ammendment’,citz,[citizen]) :-
      \+ disqualified,
      parm(birthplace,m,1).   % born in one of 50 US


disqualified :-
      prop(renounced_citizenship).

disqualified :-
      prop(american_indian_belonging_to_tribal_nation).

disqualified :-
      prop(born_to_occupying_force).

disqualified :-
      no_parent_citizen_at_time_of_applicants_birth,
      prop(child_of_foreign_diplomat).
```

Figure 2.2: RULES CORRESPONDING TO FIGURE 2.1 AS THEY APPEAR IN THE KNOWLEDGE BASE[2]

[2] If Figure 2.2 is confusing to the reader, please refer to Chapters 3 and 4 for the syntax for IMMGNOSIS rules.

CHAPTER 3

KNOWLEDGE REPRESENTATION

3.1 KNOWLEDGE STRUCTURES

IMMGNOSIS currently relies on 152 primary rules, where primary rules are those that terminate in a diagnosis, and 142 rules that define the complex conditions used throughout the knowledge base. Each primary rule is comprised of a diagnosis as a consequent and a set of conjunctive attribute/value pairs as conditions. The inference engine searches the rule base in lexical order, checking the conditions for each rule from top to bottom in a depth-first backward-chaining approach. Hence, by ordering conditions from most general to most specific within the body of a rule, we guarantee that the system will never request a piece of knowledge that is superfluous to the shortest possible inference path. Since questioning occurs on a strictly need-to-know basis, the system is capable of exploring the search space efficiently and providing concise consultations.

Rule conditions are divisible into two general categories. The term *terminal condition* refers to any condition whose value can be determined from a single piece of information that the user provides. These correspond to the condition predicates that exist in the original XSHELL, detailed in Chapter 4.1. Terminal conditions include Boolean properties, discrete-valued parameters, and any mathematical or set theoretical properties that can be inferred from those terminal conditions. Thus, a condition requiring a particular year of birth and a condition requiring a year of birth within some specified range would each constitute a terminal condition. We represent terminal conditions using the same predicate names that are used in the original

XSHELL in order to accommodate existing XSHELL knowledge bases with STATELESS XSHELL. The predicates differ significantly from the original shell, however, in that they no longer have explicit definitions and no longer initiate procedures. These changes are discussed in Chapter 4.3.

A *complex condition*, on the other hand, refers to a property that is defined by a disjunctive set of rules that are in turn defined by conjunctions of terminal and/or complex conditions. The rule in Figure 2.3 contains a complex condition, `\+ disqualified`, which has both terminal and complex conditions in its definitions. Each disjunctive definition of a complex condition must ultimately terminate in a set of terminal conditions; hence the nomenclature. With each cycle of question and response, a new piece of information about a user enters the system and determines the success or failure for any terminal condition that refers to that piece of information. Thus, the inference engine can draw conclusions regarding the success or failure of complex conditions by attempting to satisfy their terminal conditions one question at a time. Complex conditions can be regarded as Boolean properties about which we expect the user to be ignorant, but whose values can be determined through further questioning. They are particularly useful for limiting the number and length of rules in the knowledge base. If, for example, several rules require the same set of closely related disjunctive conditions, we can represent this disjunction as a single condition and define it only once in the knowledge base. Without complex conditions, we would instead have to represent each rule that requires the satisfaction of the disjunction as three separate clauses, and with three original rules, we would effectively end up with two redundant definitions of the disjunction. In fact, for every *n* rules that require the same disjunction with *m* disjuncts, we would have $n - 1$ redundant definitions of the disjunction and *nm* rules if not for complex condition representation.

In addition to simplifying the knowledge base, complex conditions also make possible the implementation of a "module-within-module" architecture. Admissibility, for example, is a prerequisite for any visa recommendation. By requiring admissibility as a complex condition in each visa rule, we initiate a process that queries the admissibility module, treating the disjunctive rules for admissibility diagnoses as complex condition definitions, and seamlessly making an intermediate diagnosis in order to derive the final one.

3.2 DEFENSE OF THEORETICAL COMMITMENT TO RULE-BASED REPRESENTATION

Since immigration law is a largely rule-based domain, it translates naturally into a rule-based expert system application. Consequently, we use a traditional rule-based representation for domain knowledge. This approach differs significantly from case-based reasoning, an approach that has been well-documented in J. Popple's presentation of SHYSTER [5]. SHYSTER is a legal expert system that applies case-based reasoning to various domains of Australian law. Instead of rules and first-order logic, case-based reasoning systems use numerical similarity measures between input cases and cases in the knowledge base in order to derive diagnoses. Though the SHYSTER project has proven successful, several factors commit us to a rule-based approach. First, there is considerable difficulty in obtaining a sufficiently large database of case knowledge. Moreover, case-based reasoning requires a more elusive inference scheme. There are no predetermined criteria for deciding which features of a case are most important for deciding when a source case should or should not be taken to be analogous to an input case. This can potentially result in diagnoses that are based on arbitrary similarities among uninformative attributes. Finally, the most influential factor in determining our commitment to a rule-based system is that case knowledge is further removed from the actual INA statutes, which already take the form of rules in a loose sense. Consequently, rules offer a better suited representation for the knowledge that constitutes the domain of immigration law.

Another contending approach to representing the logic of immigration, perhaps a more intuitive approach to a person lacking familiarity with expert system design, is the implementation of a procedural decision-tree instead of rule-based inference. A rule-based approach is preferable for several reasons. First, domain knowledge is more transparent in a knowledge base than in a decision tree, especially if we assume a Web environment wherein the tree would be represented over numerous files in a hypertext document. Second, knowledge base maintenance requires less programming overhead in a rule-based representation than in a decision tree. If a condition set needs to be changed at some point in the development process, this change can be effected by simply changing the rule that contains it for the rule-based approach. In a decision tree, such a change would have to be accounted for throughout a potentially large subtree. This would involve changing pointers in each file that falls below the added, deleted, or modified node in the tree. Finally, and most importantly, a decision tree grows in complexity more quickly than a rule base when we consider complex conditions. In a decision tree, a single rule would have to have a branch for every possible definition of the complex condition that it contains, effectively resulting in more rules. In a rule-based representation, we simply define the complex condition over a set of clauses and include that condition in a single rule. The efficiency of a rule-based representation becomes even more apparent when considering the possibility of using the same complex condition in multiple rules. For the decision tree, each of the original rules must have a branch that corresponds to the shared complex condition that it contains, resulting in redundant manifestations of the definition of that complex condition. In the rule base, the complex condition needs to be defined only once and can then be referred to by any rule that requires it. Figures 3.1 and 3.2 illustrate this property of complex conditions through a fabricated example. In Figure 3.2, the two vertical clusters represent the two rules in Figure 3.1, where any failure of the first rule leads to the second rule, and any failure of the second rule leads to the rest of the knowledge tree. If inference proceeds to the second rule and determines that `condition5` is satisfied, then it ends up in a subtree that is an exact replica of the subtree that follows `condition2` in the first rule. This is, in effect, a redundant definition of the complex condition, which had to be specified only once in the rule-based representation.

One might argue that this redundancy could be avoided by constructing the tree such that the shared complex condition is checked before the terminal conditions that are unique to each rule. In this case, we would have only one representation of the complex condition and two rules that branch off of it. However, if we consider that two rules might have multiple complex conditions, some of which are shared between them and some of which are not, but which are shared with other rules in the knowledge base, then redundancy is unavoidable. Furthermore, manipulating the placement of complex conditions in order to avoid redundancy could disrupt the logical ordering of conditions from general to specific, invalidating the guarantee that the application will never ask for an unnecessary piece of information.

```
xkb_identify(1,'1st_fake_rule',citz,[citizen]) :-
      condition1,
      condition2,
      fake_complex_condition.

xkb_identify(2,'2nd_fake_rule',citz,[citizen]) :-
      condition3,
      condition4,
      fake_complex_condition.

fake_complex_condition :-
      condition5.

fake_complex_condition :-
      condition6.

fake_complex_condition :-
      condition7.
```

Figure 3.1: FABRICATED RULES DEMONSTRATING THE USE OF A COMPLEX CONDITION

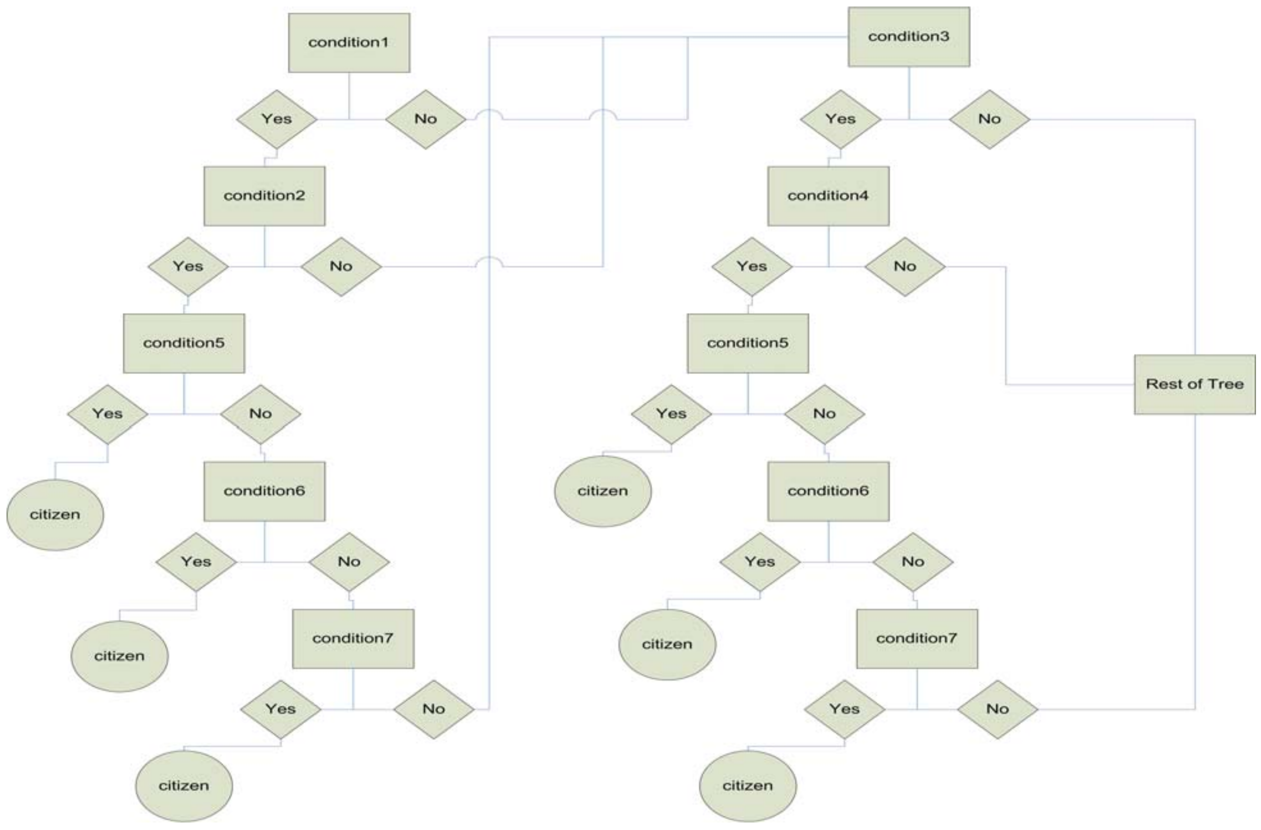

Figure 3.2: THE RULES FROM FIGURE 3.1 AS THEY WOULD APPEAR IN A DECISION-TREE REPRESENTATION

CHAPTER 4

INFERENCE IN A STATELESS ENVIRONMENT

4.1 STATE-DEPENDENT INFERENCE IN THE ORIGINAL SHELL

This section presents an overview of the inference procedure utilized by XSHELL [7], the unmodified, state-dependent shell, as a point of comparison for the stateless inference process that I describe in Section 3. The most important distinction between the two shells is that in XSHELL, the scope of a single query spans an entire consultation from start to finish. A query to the shell initiates a call to `xkb_identify/2`, which constitutes the head of each primary rule in the knowledge base. Prolog's built-in inference engine tries to satisfy each clause for `xkb_identify/2` in lexical order, initiating various procedures along the way as the inference engine attempts to satisfy the predicates that constitute rule conditions. These condition predicates correspond to the various kinds of terminal conditions described in Chapter 3. The condition predicates in XSHELL are `prop/2`, `parm/3`, `parmset/3`, and `parmrange/3`. `prop/2` handles Boolean-valued properties. `parm/3` deals with parameters that can take values from various sets including atoms, numbers, and menu choices. `parmset/3` and `parmrange/3` are set-theoretical predicates that determine whether a parameter is contained in a particular set or falls within a certain range, respectively. When any one of these predicates is called in XSHELL, its first clause tells Prolog to check whether the piece of knowledge is in the working knowledge base, or blackboard. If so, the predicate then checks whether the value on the blackboard matches the value in the condition. If so, the call succeeds. If not, it fails. If, on the other hand, there is no known value on the blackboard for that property, the predicate prompts Prolog to display an appropriate question in the console. Question text comes from one of two predicates, `xkb_question/4` or `xkb_menu/4`, depending on how the answer is delivered in the user interface [7]. In both predicates, the first argument is the name of the attribute. The subsequent arguments are text strings used to assemble the question and to display the attribute's value in the explanatory facility. (The explanatory facility is described in Section 3.) The user's answer to the question changes the blackboard and determines the success or failure of the condition predicate, and inference continues on either to the next condition (if the call ultimately succeeds) or to the next rule (if the call ultimately fails). All predicates for terminal conditions work in a similar manner in that they check the knowledge base, initiate a question if necessary, and effect any appropriate changes to the blackboard.

It should be evident from the discussion above that the current state of the system at any given time is crucial for determining whether a condition succeeds or fails and whether a question needs to be displayed to the user. The blackboard is built continually in working memory throughout the course of the consultation by the procedures specified in the definitions of terminal condition predicates, and its present state determines the course of the inference process. As Section 3 will demonstrate, the procedures contained in the definitions of terminal condition predicates in XSHELL will be of no use in a stateless inference environment.

4.2 OVREVIEW OF STATELESS ARCHITECTURE AS AN INTRODUCTION TO STATELESS INFERENCE

System architecture is discussed in greater detail in Khosla's thesis [6], and influence for the stateless system design comes from an earlier thesis by David Jennings [8]. I include this section primarily to provide relevance to the subsequent discussion of stateless inference. IMMGNOSIS is divisible into two distinct components: the expert system itself and the Web interface that delivers it. The IMMGNOSIS knowledge base and expert system shell are both implemented in LPA Prolog [9], and we use LPA's Intelligence Server (IS) to communicate with JSP, which is responsible for the front-end interface. IS provides mappings between data structures of JSP and Prolog and allows for JSP to initiate queries to the Prolog application.

At the start of a consultation, a user fills out a JSP form requesting general information from the user that is required by all modules in the knowledge base. This form writes the first blackboard for the consultation and queries the Prolog application by calling `stateless_xshell/3`, where the first argument is the blackboard string and the second is the unique session ID.[3] `stateless_xshell/3` then carries out the inference process described in the next section and generates a new JSP file to be displayed to the user. Information moves from JSP to Prolog in the form of a query, and it moves from Prolog back to JSP through a Prolog predicate, `generate_file/7`, which uses the information gleaned from the inference procedure to create a JSP file. The key to the statelessness of the system lies in the fact that a user's blackboard, the body of information pertaining to his particular consultation, is passed to and from the expert system with each interaction between JSP and Prolog. Consequently, a query in the stateless shell spans only one addition to the blackboard rather than an entire consultation. This feature eliminates the requirement that the expert system maintain information about each user in working memory throughout the course of each consultation. The following figure illustrates the flow of data throughout the system over the course of a single query.

[3] The third argument is a flag that tells the shell whether to conduct a consultation in the manner described in this thesis or to view/edit a saved one. The save/edit functions are documented in Khosla [6].

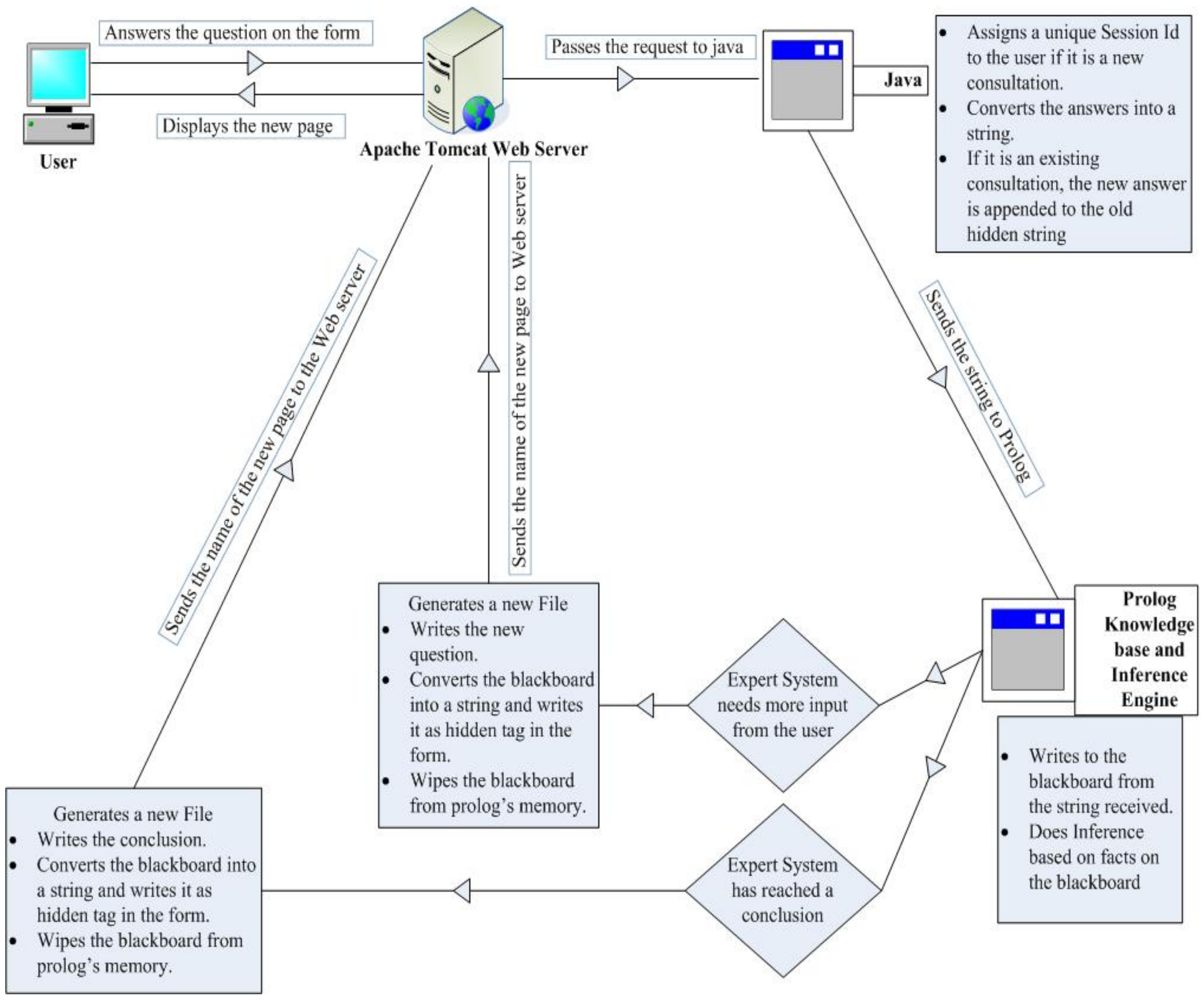
Answers the question on the form
Displays the new page
User
Apache Tomcat Web Server
Passes the request to java
Java
Assigns a unique Session Id to the user if it is a new consultation.
Converts the answers into a string.
If it is an existing consultation, the new answer is appended to the old hidden string
Sends the string to Prolog
Sends the name of the new page to the Web server
Sends the name of the new page to Web server
Prolog Knowledge base and Inference Engine
Writes to the blackboard from the string received.
Does Inference based on facts on the blackboard
Expert System needs more input from the user
Expert System has reached a conclusion
Generates a new File
Writes the new question.
Converts the blackboard into a string and writes it as hidden tag in the form.
Wipes the blackboard from prolog's memory.
Generates a new File
Writes the conclusion.
Converts the blackboard into a string and writes it as hidden tag in the form.
Wipes the blackboard from prolog's memory.

Figure 4.1: DATA FLOW FOR A SINGLE QUERY

4.3 STATELESS INFERENCE

In STATELESS XSHELL, inference proceeds in a manner that is quite different from the original version of XSHELL. In the original system, a single query to the inference engine is, in effect, a search through the entire knowledge base that sequentially builds a blackboard and terminates in a diagnosis. This implementation is not desirable for a Web-based expert system that handles requests from multiple users for several reasons. First, it does not make efficient use of system resources. The Web server would have to run separate simultaneous instances of the application for each user in order to keep track of their separate blackboards. Alternatively, the system could keep track of separate users' blackboards by distinguishing them in working memory with tags, but this would require significant programming overhead and does not effectively reduce the demand on system resources since the system must still remember multiple blackboards simultaneously. We have overcome these obstacles by redesigning the original expert system shell to run in a stateless environment.

A stateless expert system environment does not require the system to maintain information about previous actions or states in order to process a query. The benefit of using a stateless architecture is that a single instance of Prolog is sufficient for supporting multiple users while using minimal machine resources [8]. With each query in the stateless version, a string comes into the program containing all the facts that have been gleaned from the user so far. Prolog then unpacks the string, writes it to the blackboard, reasons as far as it can with the information it contains, erases the blackboard while repackaging its contents into a string, and builds a JSP page with a new question or a diagnosis and the hidden blackboard string. The query is thus satisfied and Prolog returns to its original state, awaiting a new query with a new blackboard string.

Since we do not want a query to span an entire consultation in the stateless version, we cannot rely on the built-in inference engine to process the clauses for `xkb_identify/2,` and we cannot rely on the procedures specified in the original definitions of terminal condition predicates to carry out the consultation. The remainder of this section demonstrates our alternative approach by explicitly describing each of the predicates that constitute the body of `stateless_xshell/3`, the predicate that JSP calls when it interacts with the expert system.

The first predicate in the body of `stateless_xshell/3` is `write_blackboard/1`, which takes the blackboard string as input, unpacks it, and places its contents on a temporary blackboard in working memory. It reads the current rule number and environment from the string and asserts them as facts using the predicate `known/`2, where the first argument is an attribute and the second is a value. After writing the current rule number and environment, it asserts `known/`2 clauses for all attribute/value pairs contained in the string. These will later be used to determine the success or failure of the terminal conditions that refer to them. The format of the blackboard string is a comma-separated list where the first two elements are the environment and rule number and the rest are pairs of attributes and values, where attributes are distinguished from their corresponding values by a colon. In the string that `write_blackboard/1` receives, each attribute name has been translated into a corresponding three-letter code in order to keep the string within the 4k spatial limitation imposed by HTTP's `get_string` method. Figure 4.3.1 shows a sample blackboard string and the set of `known/2` clauses that result in working memory.

Blackboard String:

```
`citz,1,aad:27,aae:3,aaf:1972,aai:y`
```

Blackboard in Working Memory:

```
known(env,citz).
known(current_rule_number,1).
known(day_of_birth,27).
known(month_of_birth,3).
known(year_of_birth,1972).
known(mother_citizen,y).
```

Figure 4.3: BLACKBOARD STRING AND THE RESULTING BLACKBOARD

The next predicate in `stateless_xshell/3` is `clause/2`, a built-in predicate that takes a rule head as input and returns that rule's conditions. We use this predicate to generate the list of conditions for the current rule by calling it with the first argument instantiated to `xkb_identify(Rule_Number,Reference,Environment,Diagnosis)`, where `Environment` and `Rule_Number` are instantiated according to the current blackboard. Much like `xkb_identify/2` in the original shell, `xkb_identify/4` constitutes the head of each primary rule in the knowledge base and contains the diagnosis and rule number as arguments. However, `xkb_identify/4` is never called in the new shell. Rather, we treat it as a tag for each rule and use it as a way of accessing that rule's conditions. We add an argument for environment in order to make it possible to load the knowledge bases for separate modules into working memory at the same time without unintentional interaction, and we also include an argument that refers to the INA statute that corresponds to the rule.

The next predicate, `process/5`, is responsible for carrying out inference in the new shell. Since we want a query to span only a single addition to the blackboard, `process/5` terminates each time it comes to a condition that either has no value or determines a diagnosis. When `process/5` is called, it is given the list of conditions for the current rule as input. `process/5` then checks each condition in the list against the contents of the blackboard and returns the necessary information that is required for generating the next JSP page that will be presented to the user. If all conditions succeed with the available information, `process/5` returns a diagnosis. If a condition in the list is encountered that has no value on the blackboard, `process/5` returns the appropriate question along with a list of possible answers. If a condition fails, `process/5` increments the current rule number, generates the list of conditions for the next rule, and repeats the aforementioned process. The conditions themselves, much like the clauses for `xkb_identify/4`, are never called in the traditional sense. In fact, they are not even independently defined in the new shell. Rather, `process/5` has an auxiliary predicate that determines whether these conditions succeed, fail, or require a question. Conditions predicates are not defined by procedures that manipulate the blackboard or prompt the user for information. These procedures are handled instead by JSP based on the information that `process/5` extracts. The first three arguments are the condition list, a flag for success, failure, or question, and the text for a question or diagnosis. The last two apply to cases that result in the generation of a new question, one being the set of possible answers and the other being the three-letter code that represents the property or parameter to which that question corresponds.

When `process/5` encounters a complex condition, it calls an auxiliary predicate that behaves in a manner similar to its own in that it generates a list of conditions for the complex condition and processes them, the only differences being that successfully processing the entire list results in the success of the complex condition rather than in a diagnosis and that failure requires backtracking to a new list of conditions without incrementing the current `xkb_identify/4` rule number that is stored in the blackboard. Recursion takes care of processing complex conditions that are defined in terms of further complex conditions.

The next predicate, `wipe_blackboard/2`, undoes the work of `write_blackboard/1`. That is to say, it erases all `known/2` clauses from the blackboard while assembling them into a list of attribute/value pairs which will later be assembled into a blackboard string.

Finally, the list of known attribute/value pairs, the return type flag, the text for the question or diagnosis, the list of possible answers, and three arguments that have to with user interface issues [6] are all sent to `generate_file/7`, which uses the information gleaned from the inference procedure to create a new JSP page with a hidden blackboard string and a new question or diagnosis [6]. It is in the body of `generate_file/7` that the list of attribute/value pairs is translated back into three-letter codes and assembled into a string. Prolog, after storing the blackboard in the user's new JSP file and having cleared its working memory, is thus rendered in the same state it was in before the query to `stateless_xshell/3`.

From this point, JSP handles the presentation of the appropriate question or diagnosis by displaying the page that results from the procedures defined in `generate_file/7`. If the page contains a question, then `generate_file/7` will have created the appropriate forms for displaying the question and its possible answers. Once the user responds, JSP routines make appropriate additions to the blackboard string and send a new query to `stateless_xshell/3`, starting the process over again. In the case of a diagnosis, the resulting JSP file will display the appropriate diagnosis text and prompt the user to continue searching for alternative diagnoses. If the user chooses to continue the consultation, then JSP increments the rule number in the blackboard string and queries `stateless_xshell/3`, forcing inference to proceed to the next rule.

The explanatory facility in STATELESS XSHELL is closely tied to the user interface, so for a detailed explanation, refer to Khosla'a thesis [6]. However, it is worth mentioning the way in which we generate relevant conditions in a stateless inference environment. When a diagnosis is reached, a flag generated by `process/5` directs Prolog to a special clause for `generate_file/7` that handles the generation of JSP files for diagnoses. Since STATELESS_XSHELL still has access to the current rule number and environment when `generate_file/7` is called, it calls `clause/2` with the appropriate instantiations for the rule number and environment arguments. `clause/2` returns a list of conditions for the rule that fired, and `generate_file/7` uses that list to access the appropriate clauses for `xkb_question/4` or `xkb_menu/4`, where those predicates contain strings that state the values for their corresponding attributes.

CHAPTER 5

CONCLUSION

5.1 EVALUATION

At present, IMMGNOSIS has not undergone intensive evaluation regarding the accuracy of its diagnoses. The system has not been released to the public, and it has never diagnosed cases that have actually been tried in a court of law. However, the experts involved in the knowledge acquisition process have validated the rule base throughout its development in the manner I described in Chapter 2. Diagnoses follow exactly as we would expect from examination of the rules in the knowledge base, and we are confident that these diagnoses represent the domain accurately to the extent that we are confident in our domain experts' interpretations of the INA. Blair Dorminey has applied IMMGNOSIS to a handful of actual cases that his firm was handling and testified that the system reduced his research time by an average of ninety percent, suggesting that we have accomplished our goal of reducing the number of man-hours required for analyzing immigration cases. In reality, an objective critique of IMMGNOSIS's practical performance will not be possible until the system has been evaluated in beta testing by a wider base of practicing legal experts.

We are much more confident, however, in our evaluation of the performance of the modified expert system shell and its stateless architecture. We have been testing the integrated system since November 2004, and it has proven itself capable of effectively reasoning over the knowledge base using the portable blackboard architecture.

Aspects of the system that Khosla covers in his thesis [6] have also been a success. In testing, subjects reported that the user interface is clear and intuitive, and we anticipate that anyone with minimal Web-browsing experience will have no difficulty interacting with the system. All of the user-interface features, from account creation to consultation storage and editing, work exactly as intended.

5.2 FUTURE DEVELOPMENT

The future development of IMMGNOSIS is three-tiered. Since immigration law is an ever-changing domain of expertise, the most obvious task for future development is that of modifying and adding to the content of the knowledge base. We are confident that the knowledge base is currently complete with respect to the subdomains that it covers, but immigration legislation changes on a regular basis. Consequently, knowledge base maintenance will be a continuous endeavor. Additionally, as we introduce the application to the public for beta testing, we hope to attain practical feedback and analysis from a larger body of domain experts than that which we have had available to us throughout the initial development process. This information will guide the process of validating and updating the existing rules in the knowledge base.

In addition to updating and validating the existing rules, we also hope to expand the coverage of the knowledge base by including several new subdomains. We are currently working on a module that makes recommendations regarding immigrant and non-immigrant visas. It suggests appropriate visa categories for potential applicants and assists those who already have a visa with extensions or changes of status. We would also like to include modules that address criminal offenses, good moral character, and any other concepts that we find to be important to the domain of immigration law.

Finally, plans for future development include expanding the functionality of the inference engine in terms of conflict resolution. The current system handles conflict by simply offering all diagnoses that the system can deduce from its knowledge base. It does not recognize conflicting diagnoses and does not provide any means of ranking multiple diagnoses when they are not in conflict with one another. We would like to improve the system's handling of multiple diagnoses by implementing certainty factors and defeasible reasoning. Certainty factors will be especially useful for the visa recommendation module, which is intended to offer multiple diagnoses, while defeasible reasoning will both eliminate the presentation of conflicting diagnoses and simplify knowledge base maintenance. The remainder of this section details the way in which certainty factors and defeasible reasoning provide the necessary means for achieving these goals.

In the visa recommendation module, it is possible for a user to qualify for multiple visas at the same time. That is, the domain allows the user to end up with multiple diagnoses that are not necessarily in conflict with one another. The system in its current state provides all diagnoses but provides no way of ranking them. We would like to provide a meaningful ranking of diagnoses in order to guide the user in whatever legal pursuits may follow from his or her consultation with IMMGNOSIS, and we plan to accomplish this goal by implementing a modified version of certainty factors. Certainty factors are a way of ranking diagnoses by combining the certainty that the expert associates with the correlation between a rule's conditions and consequent and the certainty that the user associates with his or her answers to individual questions [1]. We are considering a deviation from the traditional use of certainty factors by emphasizing desirability as well as certainty. The INA clearly defines the conditions that a person must satisfy before qualifying for a particular visa, so it would not be meaningful to associate certainty factors with the correlation between conditions and consequents. We would instead use a measure of desirability for that particular visa based on the length of stay, waiting period, financial costs, and so forth. Naturally, these desirability measures will come from our domain experts. Once these measures have been specified, we could combine them with the user's certainty regarding his or her answers to the questions that comprise each rule's conditions, resulting in a ranking that considers both certainty and desirability simultaneously.

While certainty factors will greatly increase the usefulness of the system, we currently face a much more pressing problem regarding conflict resolution. In modules other than visa recommendation, diagnoses are ultimately Boolean values. A user does or does not have a claim to U.S. citizenship, is or is not eligible for naturalization, etcetera. In the current system it is possible that the user might end up with conflicting diagnoses. In citizenship, for example, the first rule pertains to conditions that disqualify a user from any possible claim to citizenship and therefore results in negative diagnoses. However, if the user falls into one of these disqualifying categories and decides to continue searching for diagnoses, he or she will end up with a positive diagnosis for each positive identification rule that he or she satisfies. One way of overcoming this problem is to require the failure of each disqualifying condition in every rule that corresponds to a positive identification, which we have done in the citizenship module. Thanks to complex-condition representation, we have had to add to each positive identification rule in the knowledge base only one complex condition, `\+ disqualified`, rather than the negation of each of the properties that result in disqualification. However, this universal repetition of even one complex condition is a grossly inefficient way of resolving potential conflicts. Every citizenship rule currently requires a condition that serves only to redundantly reference the properties that are expressed in the first negative identification rule. We propose a defeasible inference engine as an efficient means of overcoming the problem of conflicting diagnoses.

In a defeasible inference application we divide the knowledge base into strict rules and defeasible rules. This is somewhat of an oversimplification, but it provides enough of an understanding to demonstrate the benefits of defeasible reasoning for conflict resolution. When a strict rule's conditions are satisfied, the defeasible inference engine can derive the consequent of the rule. Defeasible rules, however, require a further condition, namely, that the rule not be defeated. A rule is defeated when another rule can be satisfied without being defeated and has a consequent that conflicts with the consequent of the original rule. In the case of the `\+ disqualified` condition that permeates the citizenship module, we could handle conflict resolution by making every positive identification rule a defeasible one. This captures the logic of the original INA statutes better than the strict rule representation. The statutes are meant to convey that you are a citizen *in the general case* as long you satisfy such and such conditions. Once we incorporate this vulnerability into the positive identification rules, we simply specify the rules for `disqualified` as we have done in the current knowledge base, representing them as strict rules, but we add the knowledge that `disqualified` conflicts with any positive identification. The inference engine will recognize this potential for defeat when processing positive identification rules, and if it has determined that the conditions for such a rule are satisfied, it will then try to satisfy the rules that have `disqualified` as a consequent. If all rules for `disqualified` fail or are defeated by other rules, then the system will return the positive diagnosis. If, on the other hand, the inference engine is capable of satisfying a `disqualified` rule, then the system will return only the negative diagnosis. Under no circumstances will the system ever return conflicting diagnoses, and we will only have to refer to the set of `disqualified` rules in the one `xkb_identify/4` clause that results in a negative diagnosis based on disqualification. We will be able to represent all positive identification rules with conditions that more closely mirror the INA statutes, doing away with repeated reference to disqualification. Disqualification will still be checked each time a positive rule's conditions are satisfied, but the check will occur implicitly in the defeasible inference process rather than explicitly in each rule's conditions.

REFERENCES

[1] Jackson, P. (1999). *Introduction to expert systems* (3rd ed.). Harlow, England: Addison Wesley Longman Limited.

[2] Awad, E. M. (1996). *Building expert systems: Principles procedures, and applications*. St. Paul: West Publishing Company.

[3] Shortliffe, E. H. (1974), MYCIN: a rule-based computer program for advising physicians regarding antimicrobial therapy selection. *Proceedings of the ACM National Congress (SIGBIO SESSION), 739.*

[4] Aikens, J. S. (1983). PUFF: an expert system for interpretation of pulmonary function data. *Computers and Biomedical Research. 15, 199-208.*

[5] Popple, J. (1993). *Shyster: a pragmatic legal expert system.* The Australian National University, Canberra.

[6] Khosla, V. (2005). *IMMGNOSIS: architecture for a stateless, Web-based expert system for immigration law.* A master's thesis submitted to the graduate faculty at the University of Georgia

[7] Covington, M. A., Nute, D., & Vellino, A. (1997). *Prolog programming in depth*. New Jersey: Prentice-Hall Inc.

[8] Jennings, D. *JXSHELL: a Web-based expert system platform.* A master's thesis submitted to the graduate faculty at the University of Georgia

[9] Logic Programming Associates Ltd., available at http://www.lpa.co.uk