Ibrahim Tadros
6.871: Knowledge Based Application Systems
Final Project

6.871 Knowledge-Based Application Systems

Final Project

Constitution Builder

What is Federalism?

Federalism as defined by the Merriam-Webster Dictionary is the distribution of power in an organization (as a government) between a central authority and the constituent units. Unlike a unitary state, sovereignty in a federal state is split between at least two territorial levels such that the units at each level have final authority and can act independently of the other in some area. This means that the citizens of such a country have an obligation to two or more authorities. The division of such responsibilities usually involves the central government controls crucial matters such as defense and foreign policies, while the sub-units might also have international roles. The advantages of setting up a federal state include is it allows more opportunities for political participation by allowing each sector of the population to participate equally in decision making. It also increases the access to the government, by setting up sub-governments in the different sectors of a country. It also allows a greater diversity of opinion in public policies, and decreases the number of decisions and compromises at the national level.

Although federalism seems like an ideal political structure, it does have its limitations. Disadvantages of federalism include:

- It can lead to duplication of government and inefficient, overlapping or contradictory policies in different parts of the country

- It can lead to inequality between the States and lead to unhealthy competition and rivalry
- It can lead to neglect in important areas of public policy
- It can lead to over-government

Currently, there are around 21 countries that can be categorized as federal states, and they are:
Argentina, Australia, Austria, Belgium, Bosnia and Herzegovina, Brazil, Canada, Comoros, Ethiopia, Germany, India, Malaysia, Mexico, FS Micronesia, Nigeria, Russia, Serbia and Montenegro, Switzerland, United Arab Emirates, United States, Venezuela.

Project Aim:

Having discussed what federalism is, the purpose of this project was to design a system which enables a political body to determine whether federalism would be a good option to adopt for a certain country. The system incorporates several factors that help determine whether or not such a political system is applicable given a set scenario. The different criteria used are:

- Population
- Land Area
- Racial Diversity
- Religious Diversity
- Ethnic Diversity
- Linguistic Diversity

Population:

In general, it is pretty difficult to implement a federal structures where you have a small population. Since you need to have several sub-governments, if a country is initially small, the sub-governments will be much small to be able to be effective. Nevertheless, small federal states such as the Federal State of

Micronesia (with a population of 108,105[1]) do exist, but are not the norm.

Land Area:

Similar to the rational behind population, dividing up a small country in terms of physical area will be harder to do compared to a larger country. If a country is small in size, the population is able to access the government quite easily compared to a country that is large.

Racial, Religious, Cultural and Ethnic Diversity:

Divisions that will be done in a country will be based on a set of criteria. Each sub-government must encompass a certain group of the population with similar backgrounds. They can either share the same race (e.g. Kurds in Iraq), same religion (e.g. Hindus in India), same cultural background or have the same ethnic origin. This is the most important factor as each government needs to be united, and federal states work best when each sub-government is composed of a predominantly homogenous population.

Moreover, even if you have a country with great diversity, it will be much harder to establish a federal state if the population itself is segmented geographically. This is a very important aspect that needs to be taken into account in whether a country can adopt federalism.

Furthermore, even if a country fits the criteria mentioned above, whereby it has a decent population count with a decent area and a diverse population, the diversity ratios of the different groups need to be in such a way that no one group has a much higher prevalence over

[1] CIA Factbook, estimated figures for July 2005.

another. For example, lets say we have two countries with 3 distinct racial groups.

Racial Group	Country A	Country B
Group 1	40%	70%
Group 2	35%	20%
Group 3	35%	10%

A federal system would work better in country A because the percentages are pretty equal. This means that if a federal state were to be implemented in country A, then the three different sectors will have equal say in the central government. Contrastingly, in country B, group A will have a much larger sub-government than the other two groups meaning that more people are needed to represent them, and they will control more of the internal and external politics in the central government which might lead to racism against one race, or not providing equal rights to the whole population.

Initial Thoughts:

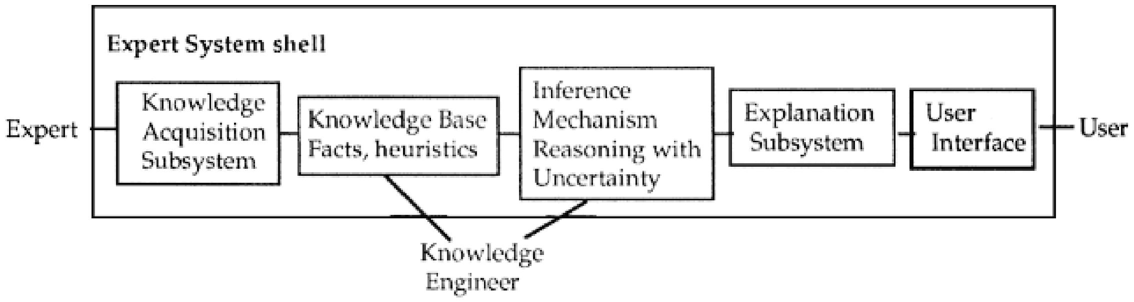
When we started thinking about this project, we were hoping to achieve several aspects a country should take when writing its constitution. This turned out to be a project of a larger scale and scope of our class, and we therefore decided to scale down, and look at one particular aspect of building a constitution, that is to help countries decide whether to adopt a federal ideology or not.

Problem Solving Paradigm:

After studying several designs of expert-based systems in class, and through the readings provided, we decided to go ahead with using a flow-chart model. As mentioned earlier, our system chooses different criteria in order to fill its knowledge base and be able to provide the user with an end recommendation. It explicitly asks the user for information covering the set criteria, and uses that information using a logical approach in order to decide whether federalism can actually be implemented. The most straight-forward and efficient architecture we decided to use was using rules. The reason this was chosen is because our system is designed in such a way that we have several layers of queries and buildup, and using rules is the best way to our knowledge in achieving our goal.

Design Details:

To begin with, this is an expert-based system. In other words, and expert needs to be involved in order to provide the actual background and framework that we should follow before deciding on any of the set rules and design decisions. The diagram below outlines how an expert-based system is generally presented[2]:


Expert System shell
Expert
Knowledge Acquisition Subsystem
Knowledge Base Facts, heuristics
Inference Mechanism Reasoning with Uncertainty
Explanation Subsystem
User Interface
User
Knowledge Engineer

To begin with, the expert we discussed our ideas with was professor Cindy Skach, who is currently an assistant professor of Government at

[2] Ch 3: Expert Systems Building Tools: Definitions. http://www.wtec.org/loyola/kb/c3_s2.htm

Harvard University. Professor Skach was pretty busy, but we spent most of our time with her teaching fellow Rosalind Dixon. As well as talking to two experts, we used a few texts that were recommended by our experts (list attached in bibliography section). After gaining insight on what she thinks were the most important topics that need to be covered, we took the role of the 'Knowledge engineer' and designed our expert-based system using the knowledge we have gained in our 6.871 class[3]. By being able to create such a system, we will be able to build our knowledge base using information from our expert coupled with queries that will be requested from the user. The knowledge base will be built using a sequence of IF-THEN rules that will lead to conclusions about our final goal. The building of the knowledge base actually occurs in the Knowledge Acquisition system, and will then be stored in the Knowledge Base subsystem. The Explanation subsystem basically explains the system's actions. This is built into Joshua, by using the 'trace' function, where you can see how certain conclusions have been achieved. The reasoning engine is where the manipulation takes place of the information in the knowledge base to form a line of reasoning to solve the problem. The user inputs the queries using a 'User Interface' which is the crucial link between the user, and infrastructure of our system. The most important aspect of any user interface is that it should be simple enough for a user who knows nothing about computer programming to be able to work with, and that's what we strived to achieve.

<u>Variables:</u>

[3] In particular, with the aid of Profesor Davis, Jacob Eisenstein, and JOSHUA

Ibrahim Tadros
6.871: Knowledge Based Application Systems
Final Project

The variables we have used in this project are summarized in the diagram on the following page to show how they are connected. I will also provide a brief written description to the different variables used.

Inputs:

Primary Inputs:

- **Population** (raw value)
- **land-area** (square kilometers)
- **racial-count**: number of different races in the country (raw value)
- **religion-count**: number of different religions practiced in the country (raw value)
- **ethnic-count:** number of different ethnicities prevalent in the country (raw value)
- **linguistic-count**: number of different languages spoken in the country (raw value)
- **first-racial-p, first-religion-p, first-ethnic-p, first-linguistic-p**: population percentage of first racial, religious, ethnic and linguistic group (decimal value between 0 and 1)
- **second-racial-p, second-religion-p, second-ethnic-p, second-linguistic-p**: population percentage of second racial, religious, ethnic and linguistic group (decimal value between 0 and 1)
- **third-racial-p, third-religion-p, third-ethnic-p, third-linguistic-p**: population percentage of third racial, religious, ethnic and linguistic group (decimal value between 0 and 1)

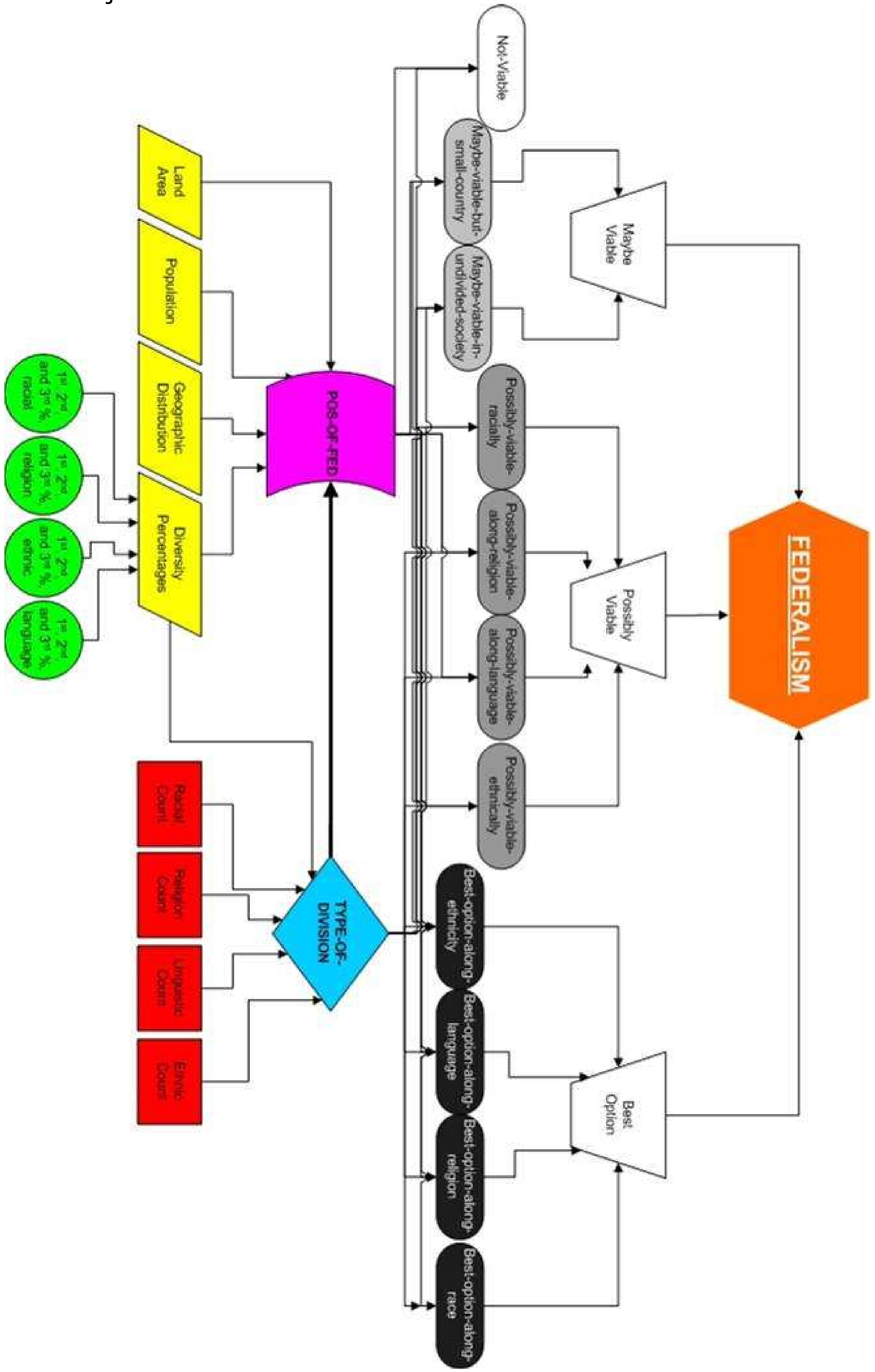
Intermediate Inputs:

- **type-of-division**: the most dominant type of division (racial/religious/ethnic/linguistic)
- **pos-of-fed:** *possibility of federalism*. This is a decimal value between 0 and 1 representing how well federalism can be implemented given the scenario.

Output:

- **federalism**: final decision on whether federalism can be applied. (not-viable/maybe-viable-in-undivided-society/maybe-viable-but-country-very-small/possibly-viable-racially/possibly-viable-along-religion/possibly-viable-ethnically/possibly-viable-along-language/best-option-along-race/best-option-along-religion/best-option-along-ethnicity/best-option-along-language)

Ibrahim Tadros
6.871: Knowledge Based Application Systems
Final Project



Land Area
Population
Geographic Distribution
Diversity Percentages
1st, 2nd and 3rd % racial
1st, 2nd and 3rd % religion
1st, 2nd and 3rd % ethnic
1st, 2nd and 3rd % language
Racial Count
Religion Count
Linguistic Count
Ethnic Count
POS-OF-FED
TYPE-OF-DIVISION
Not-Viable
Maybe-viable-but-small-country
Maybe-viable-in-undivided-society
Possibly-viable-racially
Possibly-viable-along-religion
Possibly-viable-along-language
Possibly-viable-ethnically
Best-option-along-ethnicity
Best-option-along-language
Best-option-along-religion
Best-option-along-race
Maybe Viable
Possibly Viable
Best Option
FEDERALISM

One small note on the diagram is that I have added an extra layer, just make the diagram clearer. It is the layer containg the three general options of *maybe viable, possible viable, and best option*.

In my opinion, the most crucial variable is *pos-of-fed*. This variable controls the level of applicablilty of federalism to the given situation. Everytime a rule is satisfied that a stronger claim for federalism, the value is increased. This will be discussed in more detail.

<u>Rules Discussion</u>:

The rules in our system were written using the LISP language, in order to be able to implement it using the JOSHUA infrastructure. The rules are written in such a way to be able to achieve a layered-design architecture using simple *modus ponens* backward chaining of IF-THEN rules. The design decision to use backward chaining is because we are working bottom-up, basically answering the user's question of whether his scenario is actually applicable of applying federalism in the given scenario. The way our rules are designed is the user is queried with the basic questions concerning land area, population and diversity details. The system uses the data in order to draws simple sub-decisions, creating a new level of inputs. These new inputs will then be combined together in order to move to larger goal, that of deciding what the state the scenario fits the federal model.

The rules we have designed are simple IF-THEN statements. The expression taken in as a condition is evaluated, and if it is evaluated to true, then the statement is executed. For example, if we look at rule 5:

```
if      racial_count <= 2
        then type_of_division = null
```

In plain English, if the number of racial divisions in a country is less than 2, that means there are very few racial divisions, and it is not racially diverse, and therefore the type of division is set to null.

We came up with 69 rules that can be grouped in 5 distinct groups:

Rules Dealing with Country Size:

These rules will determine whether the size and population of the country are good enough for federalism to be implemented. The larger the country, and the higher the population, the higher the chances for federalism. The value of pos-of-fed is highest when it is a large and populous country, while it is lowest when it is a small country with a low population.

Rules Assigning Type of Division:

Initially, the rules check if they country can be divided by race. If so, it compares that to the next division, which is religion. If religion is a better option for division (i.e, percentages are closer to each other), then the value of pos-fed is set to the religion. This goes through all the diversty categories, and chooses the one with the best set of values. If none exist, the value is stored as null.

Rules dealing with calculating possibility:

These rules check to see the respective percentages of the highest three divisions within one of our diversity categories. The previous group or rules have already assigned which type of division will be used, and therefore we will jump to respective rule that deals with it. For example, if the rules assigning the type of division concluded that we should divide the country by ethnicity, then rules 26-29 and 38-40 will be triggered. The pos-of-fed is then assigned according to how close the percentages are to each other.

Rules About Geographic Distribution:

These set of rules determine whether a the diversity category is spread out geographically. The more spread out the divisions are, the more applicable federalism is. The pos-of-fed is then set according to these relationships.

Rules Assessing Kind of Federalism or None At All:

These are the final set of rules which determine whether federalism is applicable, and to what extent, and according to what diversity category. Certain values have been assigned to the pos-of-fed along the way in such a manner to create 5 distinct categories.

- If the country has no type of division and the rules of country size have assigned it a low pos-of-fed, and the possibility of federalism is less than 0.3, then *federalism is not viable* (rule 59)
- If the country has no divisions, but has a pos-of-fed value larger than 0.3, *then federalism maybe viable but there are no divisions* (rule 60)*.*
- If the country has divisions, but has a pos-of-fed lower than 0.3, *then the federalism is possible, but the country is just too small* (rule 61)
- If the country has a pos-fed that is between 0.3 and 0.6, then *federalism is possibly viable, according to the type-of-division chosen* (rules 62-65)*.*
- If the country has a pos-fed that is larger than 0.6, then *federalism is the best option, according to the type of division chosen* (rules 66-69)*.*

Program Logic:

The logic of our design follows very smoothly from the order the rules were written. The initial step of the process is when the user queries the system whether federalism can be applied, using the *ask* feature in JOSHUA. Our program then checks the step right below the final result of giving a recommendation, and sees which inputs it needs to get the values for. The values required at this point are those of the type of division (type-of-division) and the possibility of federalism (pos-of-fed). In order to be able to return a value for those variables, the program needs to dig in deeper to figure out what values it needs from the user in order to give a recommendation. It queries the user for more information on population statistics, land area, cultural, racial ethnical and linguistic diversities. After retrieving the required data, it incorporates that in calculating a numerical value for the pos-of-fed. After that, it checks to see according to what diversity category, if any, the country can be divided into, and that value will be stored as the value for the type-of-division. Each time the program steps through, the value of both variables change dynamically as the new data is incorporated into the analysis.

Working Example:

While working on this project, the main case study we were thinking of along the way was the case of Iraq. Therefore, I will use that example in order to show how our system successfully gives a recommendation. Iraq has a population of around 25,000,000 and an area of 435,000. The three main ethnic groups are Kurdish, Sunnis and Shiites. The country has gone through a terrible political roller-coaster over the past century, where a minority population (Saddam Hussein's Sunni background) ruled a majority of the population with a different ethnic background (around 60% of the population are Shiites). Moreover, the Kurds have their own language, with a different culture which supports the idea of a federal state. If our program is correct, it should suggest a federal structure giving the three sectors a major role in the government. This is what our program provides:

```
(ask [federalism IRAQ ?x] #'print-answer-with-certainty)
What is IRAQ's number of races: 1
What is IRAQ's most prevalent races population percentage
(enter 0.3 for 30%): 1
What is IRAQ's number of religions practiced: 1
What is IRAQ's most followed religions population percentage
(enter 0.3 for 30%): 1
What is IRAQ's number of living ethnicities: 3
What is IRAQ's most populous ethnicity's population percentage
(enter 0.3 for 30%): 0.6
What is IRAQ's number of spoken languages: 2
What is IRAQ's most spoken languages population percentage
(enter 0.3 for 30%): 1
What is IRAQ's population: 25,000,000
What is IRAQ's land area: 500,000
What is IRAQ's second most populous ethnicity's population
percentage (enter 0.3 for 30%): 0.25
What is IRAQ's geographical grouping along the lines of
community (none/ethnic/religion/race/linguistic): ethnic
[FEDERALISM IRAQ BEST-OPTION-ALONG-ETHNICITY] 0.55474997
[FEDERALISM IRAQ POSSIBLY-VIABLE-ETHNICALLY] 0.45
[FEDERALISM IRAQ MAYBE-VIABLE-IN-UNDIVIDED-SOCIETY] 1.0e-5
```

Discussion of the Constitution-Builder:

As mentioned earlier, we initially wanted to provide a system where it takes all different aspects into account to aid in writing a constitution, but that was too large of a topic to cover in the scope of our class. After deciding to focus on the application of federalism, and having that new aim in mind, we have designed a system that can give a confident decision on whether a country should adopt federalism based on population, geography, languages spoken, different religions, ethnicities and races. If our system was inputted with a country with a large area and a large population with three different religions practiced in different segments of the country, the system will suggest adopting federalism as a best option. Moreover, if you have a small country with a homogenous population, the system will advise against it. Even if we had scenarios of different diversity criteria intermingled between each other, our program will still provide the best possible scenario. The idea behind providing several options is to show the user that there are actually several plausible scenarios that can be implemented. Nevertheless, the system provides a certainty factor with each option for the user to take into account when choosing which option to use.

Constitution-Builder Limitation:

As much as I love our system, and thought it provided pretty decent results, there are several cases where our system fails. For simplicity we chose to discard countries that have small areas with small population because we assumed it will be much harder to implement. This is not always the case. Countries such as the FS of Micronesia with a small population and area, but still practice federalism. Another situation where our system might provide the correct result but with a lower certainty is when you have a country divided geographically by a certain diversity category, while the percentages are better due to another category. For example, a country might have three different languages, and the languages are spoken by the population pretty homogenously, but the population is segmented in two areas by religion where one religion is practiced much higher than the other. Our program will return federalism based on language, while an expert might suggest a division by religion.

One idea to solve such problems will be to incorporate other countries as examples for reference in our knowledge base. For example, the situation in Singapore is composed of three ethnic groups (Chinese 75%, Malay 15%, Indian 10%). This case is very similar to Iraq, and therefore we can use this to provide a suggestion for Iraq.

System Limitation:

By system limitation, I am discussing the problems faced by using Joshua. Joshua is a pretty simple language, with simple syntax which is good when we need to write small rule-based systems which are not that complicated. When programs become too big, Joshua becomes either too redundant, or too limited to be able to use. For example we could have written a smaller number of rules had Joshua been able to implement a more general form of a certain value. For example, we could have only needed three values that represent the three percentages of a certain diversity category, and those values change dynamically as we run through the program. Instead, we had to initialize 12 variables instead of 3.
More concretely, we pretty much repeated the same 2 rules repeated 3 times for each diversity category (2X3X4) = 24 rules. Not only does it not allow us to write a general method, these rules had to be written to go around the simplicity of Joshua, because it does not have a built in increment function. In our design, our ideal thoughts were to increment the value of the pos-of-fed variable every time we come across a better input that supports federalism.

Conclusion:

As a conclusion, I really learnt a lot from working on this project. I remember when we first wrote our proposal and handed it in, Professor Davis' comments were that the project was too broad.

Usually, professors in other subjects suggest that you do more work, since the project is not focused enough. I had no idea how much work it was going to take to work on such a small aspect of a larger project. It is amazing how complicated and time-consuming a small module of a large-scale project can be. The other cool aspect of this project was to actually see that what you learn in class actually works in theory. It is truly amazing to be able to collect the expert's knowledge on a specific subject, and actually be able to you use artificial intelligence to simulate the expert's thinking process. Let's hope that someone takes this project further, as some governments might truly benefit from it. As obvious as it may sound, some governments do not see the obvious solutions, and rather use complicated approaches to solve a simple situation.

Constitution Builder Code:

```
;;; Investment Knowledge Base for Joshua
;;; for use with Rule-Based Systems Exercises
;;; 6.871 Spring 2004

(in-package :ju)

; (ask [federalism iraq response] #'print-answer-with-certainty)



(defun print-answer-with-certainty (backward-support &optional (stream *standard-
output*))
  (check-type backward-support cons "backward-support from a query")
  (let ((predication (ask-database-predication backward-support)))
    (check-type predication predication "a predication from a query")
    (terpri stream)
    (ji::truth-value-case  (predication-truth-value predication)
      (*true*
       (prin1 predication stream))
      (*false*
       (write-string "[not " stream)
       (ji::print-without-truth-value predication stream)
       (write-string "]" stream)))
    (format stream " ~d" (certainty-factor predication))))



(defgeneric possesive-suffix (predication))
(defgeneric first-prompt (predication))
(defgeneric second-prompt (predication))
(defgeneric third-prompt (predication))
(defgeneric possible-values (predication))
(defgeneric get-an-answer (predication &optional stream))
(defgeneric appropriate-ptype (predication))
(defgeneric accept-prompt (predication))
(defgeneric question-prefix (predication))
(defgeneric remaining-object-string (predication))

;;; The base mixin
(define-predicate-model question-if-unknown-model () () )

(clim:define-gesture-name :my-rule :keyboard (:r :control :shift))
(clim:define-gesture-name :my-help :keyboard (:h :control :shift))
(clim:define-gesture-name :my-why :keyboard (:w :control :shift))

(defparameter *mycin-help-string*
  "
 ctrl-?  - to show the valid answers to this question
 meta-r  - to show the current rule
 meta-y  - to see why this question is asked
 meta-h  - to see this list"
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; explaining why we're asking what we're asking
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun print-why (trigger rule &optional (stream *standard-output*))
  (format stream "~%We are trying to determine ")
  (if (predicationp trigger)
    (progn (format stream "~a " (question-prefix trigger)) (say trigger stream))
    (princ trigger stream))
  (if (null rule)
    (format stream "~%This is a top level query")
    (let* ((debug-info (ji::rule-debug-info rule))
           (sub-goals (let ((ji::*known-lvs* nil))(eval (ji::rule-debug-info-context
debug-info)))))
      (format stream "~%This is being asked for by the rule ~a in order to determine:~%"
              rule)
      (format stream "~a " (question-prefix ji::*goal*)) (say ji::*goal* stream)
      (typecase sub-goals
        (ji::and-internal
         (let ((remaining-preds (rest (predication-statement sub-goals)))
               (good-answers nil)
               (remaining-stuff nil)
               (first-remaining-object-string nil))
           (labels ((do-good-preds ()
                      (when remaining-preds
                        (let ((first (pop remaining-preds)))
                          (cond
                           ((not (predicationp first))
                            (push (copy-object-if-necessary first) good-answers)
                            (do-good-preds))
                           (t
                            (let ((found-it nil))
                              (ask first
                                   #'(lambda (just)
                                       (push (ask-database-predication just) good-
answers)

                                       (setq found-it t)
                                       (do-good-preds))
                                   :do-backward-rules nil
                                   :do-questions nil)
                              (unless found-it
                                (with-statement-destructured (who value) first
                                  (declare (ignore who))
                                  (with-unification
                                    (unify trigger first)
                                    (setq first-remaining-object-string (remaining-
object-string first))

                                    (unify value first-remaining-object-string)
                                    (setq remaining-stuff
                                          (loop for pred in remaining-preds
                                                if (predicationp pred)
                                                collect (with-statement-destructured (who
value) pred
                                                          (declare (ignore who))
                                                          (unify value (if
(joshua:unbound-logic-variable-p value)
                                                                           (remaining-
object-string pred)
                                                                           (joshua:joshua-
logic-variable-value value)))
                                                          (copy-object-if-necessary
pred))
                                                else collect (copy-object-if-necessary
pred)))))))))))))
             (do-good-preds))
           (loop for pred in (nreverse good-answers)
                 for first-time = t then nil
                 if first-time
                 do (format stream "~%It has already been determined whether: ")
                 else do (format stream "~%and whether: ")
                 do (say pred stream))
           (format stream "~%It remains to determine ~a ~a ~a"
                   (question-prefix trigger) first-remaining-object-string (remaining-
stuff-suffix trigger))
           (loop for pred in remaining-stuff
                 do (format stream "~%and ~a ~a ~a" (question-prefix pred) (remaining-
object-string pred) (remaining-stuff-suffix pred)))))
        (otherwise ))
      )))
```

```
(defmethod remaining-stuff-suffix ((pred predication)) "is")
(defmethod remaining-stuff-suffix ((expression cons)) "")
(defmethod predication-value-description ((pred predication)) (remaining-object-string
pred))


;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  PROTOCOL HACKING
;;;;;;;;;;;;;;;;;;;;;;;;;

(defmethod say ((expression cons) &optional (stream *standard-output*))
  (princ expression stream))

(defmethod remaining-object-string ((expression cons)) (format nil "~a" expression))

(defmethod question-prefix ((expression cons)) "whether")

(defmethod get-an-answer ((predication question-if-unknown-model) &optional (stream
*standard-output*))
  "Print the prompt for this parameter (or make one up) and read the reply."
  (fresh-line)
  (flet ((mycin-help (stream action string-so-far)
           (declare (ignore string-so-far))
           (when (member action '(:help :my-help :my-rule :my-why))
             (fresh-line stream)
             (case action
               (:my-why
                (print-why predication ji::*running-rule* stream)
                )
               (:my-rule
                (format stream "You are running the rule ~a" ji::*running-rule*))
               (:my-help
                (format stream *mycin-help-string*)
                ))
             (fresh-line stream)
             (write-string "You are being asked to enter " stream)
             (clim:describe-presentation-type (appropriate-ptype predication) stream)
             (write-char #\. stream)
             )))
    (let ((clim:*help-gestures* (list* :my-help :my-why :my-rule clim:*help-gestures*)))
      (clim:with-accept-help ((:top-level-help #'mycin-help))
        (clim:accept (appropriate-ptype predication)
                     :stream stream
                     :prompt (accept-prompt predication))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;  Our pseudo mycin contains 3 types of predications
;;;;  boolean valued, numeric valued, and those that take one of
;;;;  a set of values
;;;;  For each type we provide say methods
;;;;   and a bunch of subordinate methods to make dialog almost English
;;;;   and to do CLIM accepts correctly
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;; boolean values
(define-predicate-model value-is-boolean-mixin () () )

(define-predicate-method (say value-is-boolean-mixin) (&optional (stream *standard-
output*))
  (with-statement-destructured (who yesno) self
    (format stream "~A~A ~A ~A"
            who (possesive-suffix self)
            (if (joshua:joshua-logic-variable-value yesno) (first-prompt self) (second-
prompt self))
            (third-prompt self))))

(defmethod remaining-object-string ((predication value-is-boolean-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~A ~A ~a"
            (joshua:joshua-logic-variable-value who)
            (first-prompt predication) (third-prompt predication))))

(defmethod appropriate-ptype ((predication value-is-boolean-mixin)) '(clim:member yes
no))

(defmethod accept-prompt ((predication value-is-boolean-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~%Is it the case that ~a~a ~a ~a"
            who (possesive-suffix predication)
            (first-prompt predication)
            (third-prompt predication))))

(defmethod question-prefix ((predication value-is-boolean-mixin)) "whether")

(defmethod possible-values ((predication value-is-boolean-mixin)) '("yes" "no"))

(defmethod remaining-stuff-suffix ((pred value-is-boolean-mixin)) "")
(defmethod predication-value-description ((pred value-is-boolean-mixin)) "foobar")


;;;; numeric values

(define-predicate-model value-is-numeric-mixin () () )
(define-predicate-method (say value-is-numeric-mixin) (&optional (stream *standard-
output*))
  (with-statement-destructured (who number) self
    (if (joshua:unbound-logic-variable-p number)
      (format stream "is ~a~a ~a"
              who (possesive-suffix self) (first-prompt self))
      (format stream "~A~A ~A is ~A ~A"
              who (possesive-suffix self)
              (first-prompt self)
              (joshua:joshua-logic-variable-value number)
              (second-prompt self)))))

(defmethod remaining-object-string ((predication value-is-numeric-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~A~A ~A"
            (joshua:joshua-logic-variable-value who) (possesive-suffix predication)
            (first-prompt predication))))


(defmethod appropriate-ptype ((predication value-is-numeric-mixin)) 'number)

(defmethod accept-prompt ((predication value-is-numeric-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~%What is ~a~a ~a"
            who (possesive-suffix predication) (first-prompt predication))))

(defmethod question-prefix ((predication value-is-numeric-mixin)) "what")


;;; variety of possible values

(define-predicate-model value-is-option-mixin ()  () )

(define-predicate-method (say value-is-option-mixin) (&optional (stream *standard-
output*))
  (with-statement-destructured (who option) self
    (format stream "~A~A ~A ~A ~A"
            who (possesive-suffix self)
            (first-prompt self)
            (second-prompt self)
            (joshua:joshua-logic-variable-value option))))
```

```
(defmethod remaining-object-string ((predication value-is-option-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~A~A ~A"
            (joshua:joshua-logic-variable-value who) (possesive-suffix predication)
            (first-prompt predication))))

(defmethod appropriate-ptype ((predication value-is-option-mixin)) `(member ,@(possible-
values predication)))

(defmethod accept-prompt ((predication value-is-option-mixin))
  (with-statement-destructured (who value) predication
    (declare (ignore value))
    (format nil "~%What is ~a~a ~a"
            who (possesive-suffix predication) (first-prompt predication))))

(defmethod question-prefix ((predication value-is-option-mixin)) "whether")


;;; Predicate defining macro

(defmacro define-predicate-with-ancillary-info ((pred-name mixin)
                                                &key
                                                possesive-suffix
                                                prompt1 prompt2 prompt3
                                                possible-values
                                                missing-value-prompt
                                                )
  `(eval-when (:compile-toplevel :execute :load-toplevel)
     (define-predicate ,pred-name (who value) (,mixin question-if-unknown-model cf-mixin
ltms:ltms-predicate-model))
     (defmethod possesive-suffix ((predication ,pred-name)) () ,possesive-suffix)
     (defmethod first-prompt ((predication ,pred-name)) () ',prompt1)
     (defmethod second-prompt ((predication ,pred-name)) () ',prompt2)
     ,(when prompt3 `(defmethod third-prompt ((predication ,pred-name)) () ',prompt3))
     ,(when possible-values `(defmethod possible-values ((predication ,pred-name))
',possible-values))
     ,(when missing-value-prompt `(defmethod missing-value-prompt ((predication ,pred-
name)) ',missing-value-prompt))
  ))

;;; predicates that take numeric values
(define-predicate-with-ancillary-info (population value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "population")
(define-predicate-with-ancillary-info (land-area value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "land area" :prompt2 "sq km")
(define-predicate-with-ancillary-info (pos-of-fed value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "possibility of federalism" :prompt2 "on a scale of 0
to 1")
(define-predicate-with-ancillary-info (racial-count value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "number of races" :prompt2 "is")
(define-predicate-with-ancillary-info (religion-count value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "number of religions practiced":prompt2 "is")
(define-predicate-with-ancillary-info (ethnic-count value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "number of living ethnicities":prompt2 "is")
(define-predicate-with-ancillary-info (linguistic-count value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "number of spoken languages" :prompt2 "is")
(define-predicate-with-ancillary-info (first-racial-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most prevalent race's population percentage (enter
0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (second-racial-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "second most prevalent race's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (third-racial-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "third most prevalent race's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (first-religion-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most followed religion's population percentage
(enter 0.3 for 30%)":prompt2 "is")
```

```
(define-predicate-with-ancillary-info (second-religion-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "second most followed religion's population
percentage (enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (third-religion-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "third most followed religion's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (first-ethnic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most populous ethnicity's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (second-ethnic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "second most populous ethnicity's population
percentage (enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (third-ethnic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "third most populous ethnicity's population
percentage (enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (first-linguistic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most spoken language's population percentage (enter
0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (second-linguistic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "second most spoken language's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (third-linguistic-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "third most spoken language's population percentage
(enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (first-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most important diversity's highest population
percentage (enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (second-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most important diversity's second highest population
percentage (enter 0.3 for 30%)":prompt2 "is")
(define-predicate-with-ancillary-info (third-p value-is-numeric-mixin)
    :possesive-suffix "'s" :prompt1 "most important diversity's third highest population
percentage (enter 0.3 for 30%)":prompt2 "is")

;;; Predicates that take one of a set of values
(define-predicate-with-ancillary-info (type-of-division value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "most prominent type of division
(none/racial/religion/ethnic/linguistic)" :prompt2 "is"
  :possible-values (none racial religion ethnic linguistic))
(define-predicate-with-ancillary-info (geographic-grouping value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "geographical grouping along lines of community
(none/racial/religion/ethnic/linguistic)" :prompt2 "is"
  :possible-values (none racial religion ethnic linguistic))
(define-predicate-with-ancillary-info (federalism value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "option on federalism" :prompt2 "is"
  :possible-values (not-viable maybe-viable-in-undivided-society maybe-viable-but-
country-very-small possibly-viable-racially possibly-viable-along-religion possibly-
viable-ethnically possibly-viable-along-language best-option-along-race best-option-
along-religion best-option-along-ethnicity best-option-along-language))


;; using this model, the system will ask the user any time
;; it needs a specific fact to continue backward chaining.

;;; we should only be asking a question under the following
;;; circumstances:
;;;
;;; the predication being asked contains no logic variables
;;; eg. [has-health-insurance matt yes], not
;;; [has-health-insurance matt ?x]
;;;
;;; AND
;;;
;;; that predication is not already in the database
;;;
;;; AND
;;;
;;; any other predication matching the predicate and ?who
;;; eg. [has-health-insurance matt no] is not already in the
;;; database.
```

```
;;;
;;; AND
;;;
;;; there is no rule we can use to find out the answer
;;;
;;; this can be told by check [known [has-health-insurance matt ?]]


(define-predicate already-known (predicate object))

;;; if after doing the normal processing nothing is found
;;; then finally ask the guy a question if appropriate
(define-predicate-method (ask question-if-unknown-model) (intended-truth-value
continuation do-backward-rules do-questions)
  (let ((answers nil)
        (predicate (predication-predicate self)))
    (flet ((my-continuation (bs)
             (let* ((answer (ask-query bs))
                    (database-answer (insert (copy-object-if-necessary answer))))
               (pushnew database-answer answers))))
      (with-statement-destructured (who value) self
        (declare (ignore value))
        (with-unbound-logic-variables (value)
          (let ((predication `[,predicate ,who ,value]))
            ;; first see if there's an answer already in the database
            ;; may want to change this to asserting already-know predication, but I'm
trying to avoid that
            (ask-data predication intended-truth-value #'my-continuation)
            (unless answers
              ;; Now go get stuff from rules.
              (when do-backward-rules
                (ask-rules predication intended-truth-value #'my-continuation do-
questions))
              ;; now go hack questions
              (unless answers
                (when do-questions
                  (ask-questions predication intended-truth-value #'my-continuation))))))
        ;; if he's doing a raws database fetch, don't ask
        (when (and (null answers) (or do-backward-rules do-questions))
          (unless (joshua:unbound-logic-variable-p who)
            (let* ((answer (get-an-answer self))
                   (database-answer (tell `[,predicate ,who ,answer]
                                          :justification '((user-input 1.0)))))
              (pushnew database-answer answers))))))
    (loop for answer in answers
          when (eql (predication-truth-value answer) intended-truth-value)
          do (with-stack-list (just self intended-truth-value answer)
               (with-unification
                 (unify self answer)
                 (funcall continuation just)))))
  ;; make it clear that there is no interesting return value
  (values))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                             ;;;
;;; Inference Rules (For importance, higher values go first.)   ;;;
;;;                                                                         ;;;
;;;                                                             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;RULES ABOUT: population and land-area minimum requirements

(defrule population-low-cutoff (:backward :certainty 0.5 :importance 90)
  if [and [population ?who ?x]
  [land-area ?who ?y]
  (and(< ?x 10000000)(< ?y 100000))]
  then [pos-of-fed ?who 0.2])
```

```
(defrule population-medium-cutoff (:backward :certainty 0.5 :importance 89)
  if [and [population ?who ?x]
  [land-area ?who ?y]
  (and (< ?x 10000000)(>= ?y 100000))]
  then [pos-of-fed ?who 0.35])


(defrule population-medium-cutoff2 (:backward :certainty 0.5 :importance 88)
  if [and [population ?who ?x]
  [land-area ?who ?y]
  (and (>= ?x 10000000)(< ?y 100000))]
  then [pos-of-fed ?who 0.35])


(defrule population-high-cutoff (:backward :certainty 0.5 :importance 87)
  if [and [population ?who ?x]
  [land-area ?who ?y]
  (and(>= ?x 10000000)(>= ?y 100000))]
  then [pos-of-fed ?who 0.5])

;;;;RULES ABOUT: Diversity typecasting


(defrule race-diversity (:backward :certainty 0.8 :importance 85)
  if [and [racial-count ?who ?x]
  [first-racial-p ?who ?y]
  (and (> ?x 2)(< ?y 0.9))]
  then [type-of-division ?who racial])


(defrule none-check-1 (:backward :certainty 0.00001 :importance 84)
  if [and [racial-count ?who ?x]
          (<= ?x 2)]
         then [type-of-division ?who none])

(defrule religion-none (:backward :certainty 0.8 :importance 83)
  if [and [religion-count ?who ?x]
         [first-religion-p ?who ?y]
         (and(> ?x 2)(< ?y 0.9))]
         then [type-of-division ?who religion])



(defrule religion-racial (:backward :certainty 0.9 :importance 82)
  if [and [religion-count ?who ?x]
         [first-religion-p ?who ?y]
         [type-of-division ?who racial]
         [first-racial-p ?who ?w]
  (and(> ?x 2)(< ?y 0.9)(< ?y ?w))]
  then [type-of-division ?who religion])

(defrule ethnic-none (:backward :certainty 0.8 :importance 81)
  if [and
        [ethnic-count ?who ?x]
         [first-ethnic-p ?who ?y]
         (and(> ?x 2)
             (< ?y 0.9))]
  then [type-of-division ?who ethnic])

(defrule ethnic-racial (:backward :certainty 0.9 :importance 80)
  if [and [ethnic-count ?who ?x]
         [first-ethnic-p ?who ?y]
         [type-of-division ?who racial]
         [first-racial-p ?who ?w]
 (and(> ?x 2)
  (< ?y 0.9)
  (< ?y ?w))]
  then [type-of-division ?who ethnic])

(defrule ethnic-religion (:backward :certainty 0.9 :importance 79)
  if [and [ethnic-count ?who ?x]
          [first-ethnic-p ?who ?y]
          [type-of-division ?who religion]
          [first-religion-p ?who ?w]
 (and (> ?x 2)
  (< ?y 0.9)
  (< ?y ?w))]
  then [type-of-division ?who ethnic])

(defrule linguistic-none (:backward :certainty 0.8 :importance 78)
  if [and [linguistic-count ?who ?x]
          [first-linguistic-p ?who ?y]
  (and(> ?x 2)
  (< ?y 0.9))]
  then [type-of-division ?who linguistic])

(defrule linguistic-racial (:backward :certainty 0.9 :importance 77)
  if [and [linguistic-count ?who ?x]
          [first-linguistic-p ?who ?y]
          [type-of-division ?who racial]
          [first-racial-p ?who ?w]
  (and(> ?x 2)
  (< ?y 0.9)
  (< ?y ?w))]
  then [type-of-division ?who linguistic])

(defrule linguistic-religion (:backward :certainty 0.9 :importance 76)
  if [and [linguistic-count ?who ?x]
          [first-linguistic-p ?who ?y]
          [type-of-division ?who religion]
          [first-religion-p ?who ?w]
 (and (> ?x 2)
  (< ?y 0.9)
  (< ?y ?w))]
  then [type-of-division ?who linguistic])

(defrule linguistic-ethnic (:backward :certainty 0.9 :importance 75)
  if [and [linguistic-count ?who ?x]
          [first-linguistic-p ?who ?y]
          [type-of-division ?who ethnic]
          [first-ethnic-p ?who ?w]
  (and(> ?x 2)
      (< ?y 0.9)

  (< ?y ?w))]
  then [type-of-division ?who linguistic])

(defrule none-final (:backward :certainty 0.7 :importance 74)
 if [or  [and  [linguistic-count ?who ?w]
               [religion-count ?who ?y]
               [racial-count ?who ?x]
               [ethnic-count ?who ?z]
                  (and (< ?x 2)
                       (< ?y 2)
                       (< ?z 2)
                       (< ?w 2))]
          [and [first-linguistic-p ?who ?d]
            [first-racial-p ?who ?a]
            [first-religion-p ?who ?b]
            [first-ethnic-p ?who ?c]
            (and (> ?a .9) (> ?b .9)(> ?c .9)(> ?d .9))]]
  then [type-of-division ?who none])



;;;; Rules about percentages

(defrule none-p-1 (:backward :certainty 1.0 :importance 73)
  if [type-of-division ?who none]
  then [first-p ?who 1])
```

```
(defrule none-p-2 (:backward :certainty 1.0 :importance 73)
  if [type-of-division ?who none]
  then    [second-p ?who 0])

(defrule none-p-3 (:backward :certainty 1.0 :importance 73)
  if [type-of-division ?who none]
  then [third-p ?who 0]])

(defrule racial-p-h (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
          [second-racial-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.5))]
  then [pos-of-fed ?who 0.8])

(defrule racial-p-m (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
          [second-racial-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.35))]
  then [pos-of-fed ?who 0.65])


(defrule racial-p-l (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
          [second-racial-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.2))]
  then [pos-of-fed ?who 0.5])


(defrule religion-p-h (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
          [second-religion-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.5))]
  then [pos-of-fed ?who 0.8])


(defrule religion-p-m (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
          [second-religion-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.35))]
               then [pos-of-fed ?who 0.65])

(defrule religion-p-l (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
          [second-religion-p ?who ?y]
          [pos-of-fed ?who ?z]
          (and (<= ?x 0.5)
               (>= ?y 0.3)
               (= ?z 0.2))]
  then [pos-of-fed ?who 0.5])

(defrule ethnic-p-h (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.5))]
  then [pos-of-fed ?who 0.8])

(defrule ethnic-p-m (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.65])

(defrule ethnic-p-l (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.2))]
  then [pos-of-fed ?who 0.5])

(defrule linguistic-p-h (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
         [second-linguistic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.5))]
              then [pos-of-fed ?who 0.8])

(defrule linguistic-p-m (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
         [second-linguistic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.65])

(defrule linguistic-p-l (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
         [second-linguistic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (<= ?x 0.5)
              (>= ?y 0.3)
              (= ?z 0.2))]
  then [pos-of-fed ?who 0.5])

(defrule racial-p-2-h (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
         [second-racial-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
```

```
               (= ?z 0.5))]
  then [pos-of-fed ?who 0.7])


(defrule racial-p-2-m (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
         [second-racial-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.55])


(defrule racial-p-2-l (:backward :certainty 0.7 :importance 73)
  if [and[type-of-division ?who racial]
        [first-racial-p ?who ?x]
         [second-racial-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.2))]
              then [pos-of-fed ?who 0.4])


(defrule religion-p-2-h (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
         [second-religion-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.5))]
   then [pos-of-fed ?who 0.7])


(defrule religion-p-2-m (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
         [second-religion-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.55])

(defrule religion-p-2-l (:backward :certainty 0.7 :importance 72)
  if [and[type-of-division ?who religion]
        [first-religion-p ?who ?x]
         [second-religion-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.2))]
              then [pos-of-fed ?who 0.4])

(defrule ethnic-p-2-h (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.5))]
              then [pos-of-fed ?who 0.7])

(defrule ethnic-p-2-m (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
```

```
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.55])

(defrule ethnic-p-2-l (:backward :certainty 0.7 :importance 71)
  if [and[type-of-division ?who ethnic]
        [first-ethnic-p ?who ?x]
         [second-ethnic-p ?who ?y]
         [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.2))]
              then [pos-of-fed ?who 0.4])


(defrule linguistic-p-2-h (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
       [second-linguistic-p ?who ?y]
       [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.5))]
  then [pos-of-fed ?who 0.7])

(defrule linguistic-p-2-m (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
       [second-linguistic-p ?who ?y]
       [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.35))]
              then [pos-of-fed ?who 0.55])

(defrule linguistic-p-2-l (:backward :certainty 0.7 :importance 70)
  if [and[type-of-division ?who linguistic]
       [first-linguistic-p ?who ?x]
       [second-linguistic-p ?who ?y]
       [pos-of-fed ?who ?z]
         (and (>= ?x 0.6)
              (>= ?y 0.2)
              (= ?z 0.2))]
  then [pos-of-fed ?who 0.4])



;;;;; Rules about geographic grouping

(defrule geo-grouping-check-l (:backward :certainty 0.8 :importance 67)
  if [and (not [geographic-grouping ?who none])
     [pos-of-fed ?who ?z]
     (<= ?z 0.2)]
  then    [pos-of-fed ?who 0.25])

(defrule geo-grouping-check-m (:backward :certainty 0.8 :importance 66)
  if [and (not [geographic-grouping ?who none])
     [pos-of-fed ?who ?z]
     (and (<= ?z 0.5)(> ?z 0.2))]
     then          [pos-of-fed ?who 0.55])

(defrule geo-grouping-check-h (:backward :certainty 0.8 :importance 66)
  if [and (not [geographic-grouping ?who none])
     [pos-of-fed ?who ?z]
     (> ?z 0.5)]
  then    [pos-of-fed ?who 0.85])
```

```
(defrule racial-grouping-l (:backward :certainty 0.9 :importance 66)
  if [and [geographic-grouping ?who racial]
         [type-of-division ?who racial]
         [pos-of-fed ?who ?z]
         (<= ?z 0.2)]]
  then   [pos-of-fed ?who 0.3])

(defrule racial-grouping-m (:backward :certainty 0.9 :importance 66)
  if [and [geographic-grouping ?who racial]
         [type-of-division ?who racial]
         [pos-of-fed ?who ?z]
           (and (<= ?z 0.5)(> ?z 0.2))]
           then   [pos-of-fed ?who 0.6])

(defrule racial-grouping-h (:backward :certainty 0.9 :importance 66)
  if [and [geographic-grouping ?who racial]
         [type-of-division ?who racial]
         [pos-of-fed ?who ?z]
         (> ?z 0.5)]
  then   [pos-of-fed ?who 0.9])

(defrule religion-grouping-l (:backward :certainty 0.9 :importance 65)
  if [and [geographic-grouping ?who religion]
        [type-of-division ?who religion]
         [pos-of-fed ?who ?z]
         (<= ?z 0.2)]
  then   [pos-of-fed ?who 0.3])

(defrule religion-grouping-m (:backward :certainty 0.9 :importance 65)
  if [and [geographic-grouping ?who religion]
        [type-of-division ?who religion]
         [pos-of-fed ?who ?z]
          (and (<= ?z 0.5)(> ?z 0.2))]
then     [pos-of-fed ?who 0.6])


(defrule religion-grouping-h (:backward :certainty 0.9 :importance 65)
  if [and [geographic-grouping ?who religion]
        [type-of-division ?who religion]
         [pos-of-fed ?who ?z]
         (> ?z 0.5)]
  then   [pos-of-fed ?who 0.9])



(defrule ethnic-grouping-l (:backward :certainty 0.9 :importance 64)
  if [and [geographic-grouping ?who ethnic]
         [type-of-division ?who ethnic]
         [pos-of-fed ?who ?z]
         (<= ?z 0.2)]
  then   [pos-of-fed ?who 0.3])

(defrule ethnic-grouping-m (:backward :certainty 0.9 :importance 64)
  if [and [geographic-grouping ?who ethnic]
           [type-of-division ?who ethnic]
           [pos-of-fed ?who ?z]
           (and
            (<= ?z 0.5)
            (> ?z 0.2))]
  then   [pos-of-fed ?who 0.6])



(defrule ethnic-grouping-h (:backward :certainty 0.9 :importance 64)
  if [and [geographic-grouping ?who ethnic]
         [type-of-division ?who ethnic]
         [pos-of-fed ?who ?z]
           (> ?z 0.5)]
  then   [pos-of-fed ?who 0.9])
```

```
(defrule linguistic-grouping-l (:backward :certainty 0.9 :importance 63)
  if [and [geographic-grouping ?who linguistic]
     [type-of-division ?who linguistic]
     [pos-of-fed ?who ?z]
     (<= ?z 0.2)]
  then    [pos-of-fed ?who 0.3])

(defrule linguistic-grouping-m (:backward :certainty 0.9 :importance 63)
  if [and [geographic-grouping ?who linguistic]
     [type-of-division ?who linguistic]
     [pos-of-fed ?who ?z]
     (and (<= ?z 0.5)(> ?z 0.2))]
  then    [pos-of-fed ?who 0.6])



(defrule linguistic-grouping-h (:backward :certainty 0.9 :importance 63)
  if [and [geographic-grouping ?who linguistic]
     [type-of-division ?who linguistic]
     [pos-of-fed ?who ?z]
     (> ?z 0.5)]
     then [pos-of-fed ?who 0.9])




;;;; Rules about final decision based on pre-calculated factors


(defrule not-viable (:backward :certainty 1.0 :importance 62)
  if [and [type-of-division ?who none]
          [pos-of-fed ?who ?y]
          (< ?y 0.3)]
  then [federalism ?who not-viable])


(defrule maybe-viable-1 (:backward :certainty 1.0 :importance 61)
  if [and [type-of-division ?who none]
          [pos-of-fed ?who ?y]
          (and (> ?y 0.3)(< ?y 0.6))]
  then [federalism ?who maybe-viable-in-undivided-society])

(defrule maybe-viable-2 (:backward :certainty 1.0 :importance 59)
  if [and (not[type-of-division ?who none])
          [pos-of-fed ?who ?y]
          (> ?y 0.3)]
  then [federalism ?who maybe-viable-but-country-very-small])


(defrule possibly-viable-racially (:backward :certainty 0.9 :importance 58)
  if [and [type-of-division ?who racial]
          [pos-of-fed ?who ?y]
          (and (> ?y 0.3)(< ?y 0.6))]
  then [federalism ?who possibly-viable-racially])

(defrule possibly-viable-along-religion (:backward :certainty 0.9 :importance 57)
  if [and [type-of-division ?who religion]
          [pos-of-fed ?who ?y]
          (and (> ?y 0.3)(< ?y 0.6))]
  then [federalism ?who possibly-viable-along-religion])

(defrule possibly-viable-ethnically (:backward :certainty 0.9 :importance 56)
  if [and [type-of-division ?who ethnic]
          [pos-of-fed ?who ?y]
          (and (> ?y 0.3)(< ?y 0.6))]
  then [federalism ?who possibly-viable-ethnically])

(defrule possibly-viable-along-language (:backward :certainty 0.9 :importance 55)
  if [and [type-of-division ?who linguistic]
          [pos-of-fed ?who ?y]
          (and (> ?y 0.3)(< ?y 0.6))]
  then [federalism ?who possibly-viable-along-language])

(defrule best-option-along-race (:backward :certainty 1.0 :importance 54)
  if [and [type-of-division ?who racial]
          [pos-of-fed ?who ?y]
          (> ?y 0.6)]
  then [federalism ?who best-option-along-race])

(defrule best-option-along-religion (:backward :certainty 1.0 :importance 53)
  if [and [type-of-division ?who religion]
          [pos-of-fed ?who ?y]
          (> ?y 0.6)]
  then [federalism ?who best-option-along-religion])

(defrule best-option-along-ethnicity (:backward :certainty 1.0 :importance 52)
  if [and [type-of-division ?who ethnic]
          [pos-of-fed ?who ?y]
          (> ?y 0.6)]
  then [federalism ?who best-option-along-ethnicity])

(defrule best-option-along-language (:backward :certainty 1.0 :importance 51)
  if [and [type-of-division ?who linguistic]
          [pos-of-fed ?who ?y]
          (> ?y 0.6)]
  then [federalism ?who best-option-along-language])

;;;;;;;;;;;;;;;;;--------------------------



(defun rules-concluding-predicate (pred)
  (let ((answers nil))
    (map-over-backward-rule-triggers `[,pred ? ?]
                                     #'(lambda (trigger) (pushnew (ji::backward-trigger-
rule trigger) answers)))
    answers))


(defun predicates-rule-relies-on (rule)
  (let ((answers nil))
    (labels ((do-one-level (stuff)
                 (let ((connective (when (predication-maker-p stuff) (predication-maker-
predicate stuff))))
                   (case connective
                     ((and or)
                      (with-predication-maker-destructured (&rest more-stuff) stuff
                        (loop for thing in more-stuff
                              do (do-one-level thing))))
                     ((nil))
                     (otherwise
                      (pushnew connective answers))
                     ))))
        (do-one-level (ji::rule-debug-info-context (ji::rule-debug-info rule))))
    answers))


(defun graph-rule-tree (predicates &key (orientation :vertical) (size :small) (stream
*standard-output*))
  (terpri stream)
  (clim:with-text-size (stream size)
    (clim:format-graph-from-roots
     (loop for pred in predicates
           collect (list 'predicate pred))
     #'(lambda (thing stream)
         (destructuring-bind (type name) thing
           (case type
             (predicate
              (clim:surrounding-output-with-border (stream)
                (princ name stream)))
             (rule
              (clim:surrounding-output-with-border (stream :shape :oval)
                (princ name stream))))))
     #'(lambda (thing)
         (destructuring-bind (type name) thing
           (case type
             (predicate (loop for r in (rules-concluding-predicate name)
                              collect (list 'rule r)))
             (rule (loop for p in (predicates-rule-relies-on name)
                         collect (list 'predicate p))))))
     :stream stream
     :orientation orientation
     :merge-duplicates t
     :duplicate-test #'equal)))

(clim-env::define-lisp-listener-command (com-graph-rules :name t)
                                        ((predicates `(clim:sequence (member ,@(loop for
pred being the hash-keys of ji::*all-predicates* collect pred)))
                                                     :prompt "A sequence of predicates")
                                         &key
                                         (orientation `(clim:member :vertical
:horizontal) :default :vertical)
                                         (size `(clim:member :tiny :very-small :small
:normal :large :very-large :huge)
                                               :default :small)
                                         (to-file 'clim:pathname :default nil)
                                         (page-orientation '(clim:member :portrait
:landscape)
                                                           :default :portrait
                                                           :prompt "If to file, print in
portrait or landscape format")
                                         (multi-page 'clim:boolean :default nil :prompt
"If to file, segment into multiple pages")
                                         (scale-to-fit 'clim:boolean :default nil :prompt
"If to file, scale to fit one page"))
   (if to-file
     (with-open-file (file to-file :direction :output :if-exists :supersede :if-does-not-
exist :create)
       (clim:with-output-to-postscript-stream (stream file
                                                      :multi-page multi-page
                                                      :scale-to-fit scale-to-fit
                                                      :orientation page-orientation)
         (graph-rule-tree predicates :orientation orientation :size size :stream
stream)))
     (graph-rule-tree predicates :orientation orientation :size size)))
```

<u>Rules:</u>

```
1]    if      population < 10,000,000
              land_area < 100,000
      then    possibility_of_federalism = 0.2

2]    if      population < 10,000,000
              land_area >= 100,000
      then    possibility_of_federalism =  0.35

3]    if      population >= 10,000,000
              land_area < 100,000
      then    possibility_of_federalism = 0.35

4]    if      population >= 10,000,000
              land_area >= 100,000
      then    possibility_of_federalism = 0.5

5]    if      racial_count > 2 and
              First_racial_percentage < 0.9
      then    type_of_division = racial

6]    if      racial_count <= 2 and
      then    type_of_division = none

7]    if      religion_count > 2 and
              first_religion_percentage < 0.9
      then    type_of_division = religion

8]    if      religion_count > 2 and
              first_religion_percentage < 0.9 and
              type_of_religion = racial
              first_religion_division < first_racial_division
      then    type_of_division = religion

9]    if      ethnic_count > 2 and
              first_ethnic_percentage < 0.9 and
      then    type_of_division = ethnic


10]   if      ethnic_count > 2 and
              first_ethnic_percentage < 0.9 and
```

```
                type_of_division = racial and
                first_ethnic_division < first_racial_division
        then    type_of_division = ethnic

11]     if      ethnic_count > 2 and
                first_ethnic_percentage < 0.9 and
                type_of_division = religion and
                first_ethnic_division < first_religion_division
        then    type_of_division = ethnic

12]     if      ling_count > 2 and
                first_ling_percentage < 0.9 and
        then    type_of_division = linguistic

13]     if      ling_count > 2 and
                first_ling_percentage < 0.9 and
                type_of_division = racial and
                first_ling_division < first_racial_division
        then    type_of_division = linguistic

14]     if      ling_count > 2 and
                first_ling_percentage < 0.9 and
                type_of_division = religion and
                first_ling_division < first_religion_division
        then    type_of_division = linguistic

15]     if      ling_count > 2 and
                first_ling_percentage < 0.9 and
                type_of_division = ethnic and
                first_ling_division < first_ethnic_division
        then    type_of_division = linguistic

16]     if      (ling_count < 2 and
                Religion_count < 2 and
                Racial_count < 2 and
                Ethnic_count < 2) or
                (first_ling_percentage > 0.9 and
                First_religion_percentage > 0.9 and
                First_racial_percentage > 0.9 and
                First_ethnic_percentage > 0.9)
        Then    type_of_division = none

 17]    if      type-of-division = none
```

```
       then  first-p = 1


18]    if    type-of-division = none
       then  second-p = 0

19]    if    type-of-division = none
       then  third-p = 0

20]    if    type-of-division = racial and
             first-racial-p <= 0.5 and
             second-racial-p >= 0.3 and
             pos-of-fed = 0.5
       then  pos-of-fed = 0.8

21]    if    type-of-division = racial and
             first-racial-p <= 0.5 and
             second-racial-p >= 0.3 and
             pos-of-fed = 0.35
       then  pos-of-fed = 0.65

22]    if    type-of-division = racial and
             first-racial-p <= 0.5 and
             second-racial-p >= 0.3 and
             pos-of-fed = 0.2
       then  pos-of-fed = 0.5

23]    if    type-of-division = religion and
             first-religion-p <= 0.5 and
             second-religion-p >= 0.3 and
             pos-of-fed = 0.5
       then  pos-of-fed = 0.8

24]    if    type-of-division = religion and
             first-religion-p <= 0.5 and
             second-religion-p >= 0.3 and
             pos-of-fed = 0.35
       then  pos-of-fed = 0.65

25]    if    type-of-division = religion and
             first-religion-p <= 0.5 and
             second-religion-p >= 0.3 and
             pos-of-fed = 0.2
```

```
        then  pos-of-fed = 0.5

26]     if    type-of-division = ethnic and
              first-ethnic-p <= 0.5 and
              second-ethnic-p >= 0.3 and
              pos-of-fed = 0.5
        then  pos-of-fed = 0.8

27]     if    type-of-division = ethnic and
              first-ethnic-p <= 0.5 and
              second-ethnic-p >= 0.3 and
              pos-of-fed = 0.35
        then  pos-of-fed = 0.65

28]     if    type-of-division = ethnic and
              first-ethnic-p <= 0.5 and
              second-ethnic-p >= 0.3 and
              pos-of-fed = 0.2
        then  pos-of-fed = 0.5

29]     if    type-of-division = linguistic and
              first-linguistic-p <= 0.5 and
              second-linguistic-p >= 0.3 and
              pos-of-fed = 0.5
        then  pos-of-fed = 0.8

30]     if    type-of-division = linguistic and
              first-linguistic-p <= 0.5 and
              second-linguistic-p >= 0.3 and
              pos-of-fed = 0.35
        then  pos-of-fed = 0.65

31]     if    type-of-division = linguistic and
              first-linguistic-p <= 0.5 and
              second-linguistic-p >= 0.3 and
              pos-of-fed = 0.2
        then  pos-of-fed = 0.5

32]     if    type-of-division = racial and
              first-racial-p >= 0.6 and
              second-racial-p >= 0.2 and
              pos-of-fed = 0.5
        then  pos-of-fed = 0.7
```

```
33]   if     type-of-division = racial and
             first-racial-p >= 0.6 and
             second-racial-p >= 0.2 and
             pos-of-fed = 0.35
      then   pos-of-fed = 0.55

34]   if     type-of-division = racial and
             first-racial-p >= 0.6 and
             second-racial-p >= 0.2 and
             pos-of-fed = 0.2
      then   pos-of-fed = 0.4

35]   if     type-of-division = religion and
             first-religion-p >= 0.6 and
             second-religion-p >= 0.2 and
             pos-of-fed = 0.5
      then   pos-of-fed = 0.7

36]   if     type-of-division = religion and
             first-religion-p >= 0.6 and
             second-religion-p >= 0.2 and
             pos-of-fed = 0.35
      then   pos-of-fed = 0.55

37]   if     type-of-division = religion and
             first-religion-p >= 0.6 and
             second-religion-p >= 0.2 and
             pos-of-fed = 0.2
      then   pos-of-fed = 0.4

38]   if     type-of-division = ethnic and
             first-ethnic-p >= 0.6 and
             second-ethnic-p >= 0.2 and
             pos-of-fed = 0.5
      then   pos-of-fed = 0.7

39]   if     type-of-division = ethnic and
             first-ethnic-p >= 0.6 and
             second-ethnic-p >= 0.2 and
             pos-of-fed = 0.35
      then   pos-of-fed = 0.55
```

```
40]   if     type-of-division = ethnic and
             first-ethnic-p >= 0.6 and
             second-ethnic-p >= 0.2 and
             pos-of-fed = 0.2
      then   pos-of-fed = 0.4

41]   if     type-of-division = linguistic and
             first-linguistic-p >= 0.6 and
             second-linguistic-p >= 0.2 and
             pos-of-fed = 0.5
      then   pos-of-fed = 0.7

42]   if     type-of-division = linguistic and
             first-linguistic-p >= 0.6 and
             second-linguistic-p >= 0.2 and
             pos-of-fed = 0.35
      then   pos-of-fed = 0.55

43]   if     type-of-division = linguistic and
             first-linguistic-p >= 0.6 and
             second-linguistic-p >= 0.2 and
             pos-of-fed = 0.2
      then   pos-of-fed = 0.4


44]   if     geographic-grouping <> none and
             pos-of-fed < 0.2
      then   pos-of-fed =  0.25

45]   if     geographic-grouping <> none and
             pos-of-fed > 0.2
             pos-of-fed <= 0.5
      then   pos-of-fed =  0.55

46]   if     geographic-grouping <> none and
             pos-of-fed > 0.5
      then   pos-of-fed =  0.85

47]   if     geographic-grouping = racial and
             type_of_division = racial and
             pos-of-fed < 0.2
        then       pos-of-fed =  0.3
```

```
48]   if      geographic-grouping = racial and
              type_of_division = racial and
              pos-of-fed > 0.2
              pos-of-fed <= 0.5
      then    pos-of-fed =  0.6

49]   if      geographic-grouping = racial and
              type_of_division = racial and
              pos-of-fed > 0.5
      then    pos-of-fed =  0.9

50]   if      geographic-grouping = religion and
              type_of_division = religion and
              pos-of-fed < 0.2
      then    pos-of-fed =  0.3

51]   if      geographic-grouping = religion and
              type_of_division = religion and
              pos-of-fed > 0.2
              pos-of-fed <= 0.5
      then    pos-of-fed =  0.6

52]   if      geographic-grouping = religion and
              type_of_division = religion and
              pos-of-fed > 0.5
      then    pos-of-fed =  0.9

53]   if      geographic-grouping = ethnic and
              type_of_division = ethnic and
              pos-of-fed < 0.2
      then    pos-of-fed =  0.3

54]   if      geographic-grouping = ethnic and
              type_of_division = ethnic and
              pos-of-fed > 0.2
              pos-of-fed <= 0.5
      then    pos-of-fed =  0.6

55]   if      geographic-grouping = ethnic and
              type_of_division = ethnic and
              pos-of-fed > 0.5
      then    pos-of-fed =  0.9
```

```
56]  if     geographic-grouping = linguistic and
            type_of_division = linguistic and
            pos-of-fed < 0.2
     then   pos-of-fed =  0.3

57]  if     geographic-grouping = linguistic and
            type_of_division = linguistic and
            pos-of-fed > 0.2
            pos-of-fed <= 0.5
     then   pos-of-fed =  0.6

58]  if     geographic-grouping = linguistic and
            type_of_division = linguistic and
            pos-of-fed > 0.5
     then   pos-of-fed =  0.9

59]  if     type_of_division = null and
            possibility_of_federalism < 0.3
     then   federalism = FEDERALISM NOT VIABLE

60]  if     type_of_division = null and
            possibility_of_federalism > 0.3
     then   federalism = FEDERALISM MAYBE VIABLE BUT NO
     SIGNIFICANT DIVISIONS IN SOCIETY TO SUSTAIN A DIVIDED
     STATE

61]  if     type_of_division <> null and
            possibility_of_federalism < 0.3
     then   federalism = Federalism MAYBE VIABLE but country is too
     small to bear costs of federalism

62]  if     type_of_division = racial and
            possibility_of_federalism > 0.3 and
            possibility_of_federalism < 0.6
     then   federalism = Federalism POSSIBLY VIABLE along RACIAL
     seperations

63]  if     type_of_division = religion and
            possibility_of_federalism > 0.3 and
            possibility_of_federalism < 0.6
     then   federalism = Federalism POSSIBLY VIABLE along
     RELIGION seperations
```

```
64]  if    type_of_division = ethnic and
           possibility_of_federalism > 0.3 and
           possibility_of_federalism < 0.6
     then  federalism = Federalism POSSIBLY VIABLE along ETHNIC
     seperations

65]  if    type_of_division = linguistic and
           possibility_of_federalism > 0.3 and
           possibility_of_federalism < 0.6
     then  federalism = Federalism POSSIBLY VIABLE along
     LINGUISTIC seperations

66]  if    type_of_division = racial and
           possibility_of_federalism > 0.6
     then  federalism = Federalism BEST OPTION along RACIAL
     seperations


67]  if    type_of_division = religion and
           possibility_of_federalism > 0.6
     then  federalism = Federalism BEST OPTION  along RELIGION
     seperations

68]  if    type_of_division = ethnic and
           possibility_of_federalism > 0.6
     then  federalism = Federalism BEST OPTION  along ETHNIC
     seperations

69]  if    type_of_division = linguistic and
           possibility_of_federalism > 0.6
     then  federalism = Federalism BEST OPTION along LINGUISTIC
     seperations
```

Ibrahim Tadros
6.871: Knowledge Based Application Systems
Final Project

Bibliography:

1) Patterns of Democracy : Government Forms and Performance in Thirty-Six Countries by Arend Lijphart

2) Comparative constitutional Engineering by Giovanni Sartori

3) Comparative Politics Today: A World View, Eighth Edition by Gabriel A. Almond, et al

4) The value patterns of democracy: A case study in comparative analysis (Reprint / Institute of Industrial Relations and Institute of International Studies) by Seymour Martin Lipset

5) Politics in Western European democracies: patterns and problems by Gary C Byrne