Travel Advisor for Sri Lanka

6.871 - Knowledge-Based Application Systems

Final Project Report by Sumudu W. Watugala

May 12, 2005

Project Group Members: Sarah Mirza, Sumudu W. Watugala

What task does your system do?

The Travel Advisor will help people traveling to Sri Lanka plan their trips according to their interests, concerns, and expectations from the trip. Oftentimes people choose a destination based on what they read in brochures, which may not paint the complete picture of the area people are interested in visiting. Our aim is to provide travel destinations based on decisions inferred from the traveler's specifications.

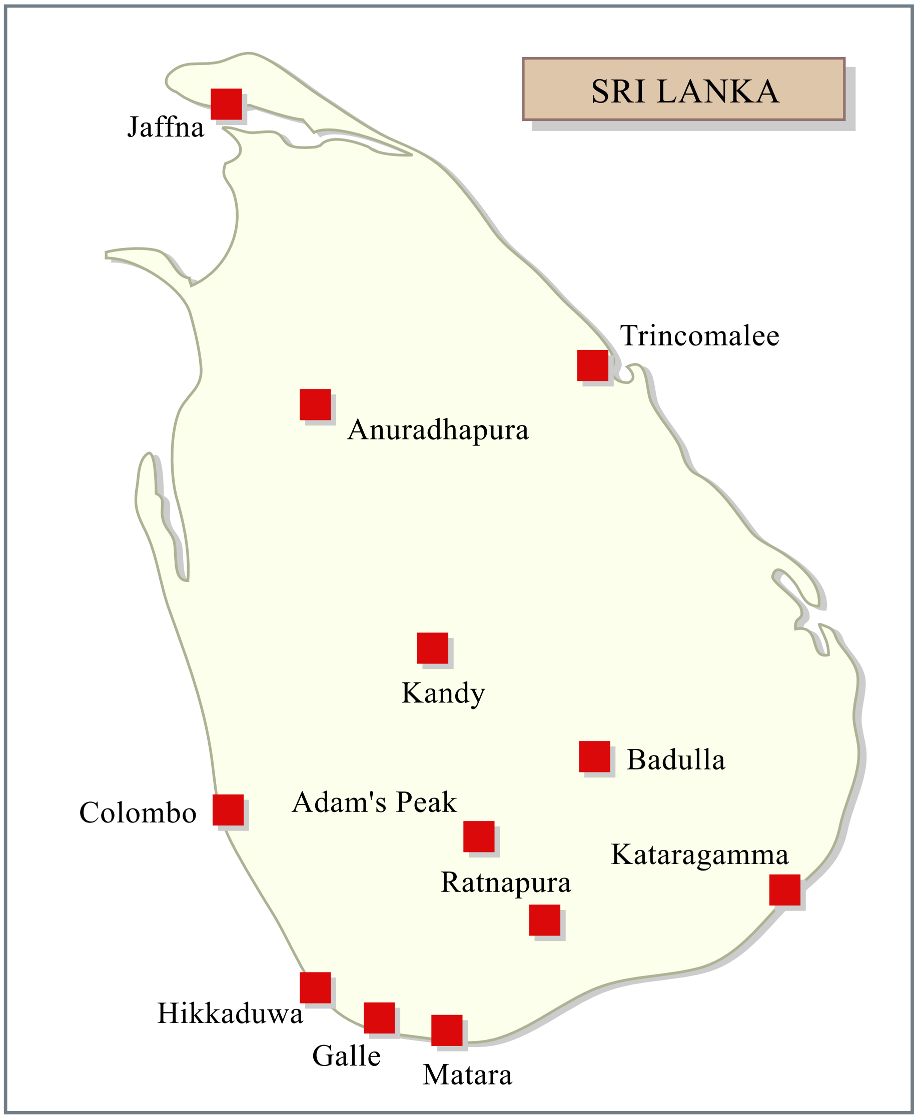

Figure by MIT OCW.

Figure 1: A pictorial map of Sri Lanka – the red squares denote the destinations included in the Travel Advisor

As seen in Figure 1, there are many locations within Sri Lanka that attract tourists. However, to construct a working KBS given realistic considerations, we have limited the scope of the project to 12 diverse destinations.

The inputs to the system include questions about the traveler's demographics and interests. The demographics are important as they allow us to get an understanding of the very basic needs of the traveler. Based on this information we can better infer his/her preferences or interests. More specifically, the inputs the system takes include information about marital-status, children, budget, number of people traveling, and date constraints etc.

Inputs about the traveler's interests will allow us to make some judgments about the type of person making the trip and what things would interest him most. As mentioned earlier, there are many things to do in the 12 cities we have selected. Therefore, inferring the specific interests of the traveler is crucial in finding the optimal destination. Some of these questions are about their general interests in everyday life and we use them to infer interest level specific to a location.

The system is designed to return a list of 12 destinations along with their certainty based on the traveler's interests. This is especially useful if the traveler has already been to the highest ranking location, or would like to visit more than one location. Since our system does not currently handle an itinerary based output, we decided that listing the ranked locations would be the best alternative. In addition, if the traveler's interests and needs are such that it does not match any one location perfectly, the Travel Advisor will still output suggestions useful and accurate (given the situation).

****

- **An example run of the Travel Advisor**

(Figure 2 shows assertions added to the database during this run)

(ask [suggested-location sumu ?x] #'print-answer-with-certainty)

Is it the case that SUMU has a date constraint: No

###Tries to asses if certain locations will have to be avoided due to the monsoon season. For example: if either there is no date constraint or if the travel period falls between January to March, then locations that are in the line of the South-Western monsoons should be avoided.

Is it the case that SUMU has children: No

Is it the case that SUMU is married: Yes

Is it the case that SUMU's group has been properly vaccinated: Yes

###These answers will be used to assess safety and health requirements of the group that is traveling.

Is it the case that SUMU has a budget for the trip.: No

###Since Sumu does not have a budget limit for the trip, this will be taken to mean that her budget range is “deluxe”. If Sumu answered that she does have a budget, the budget would be categorized according to budget and group size.

What is SUMU's vacation length: 10

###This determines how close Sumu should stay to the airport. If her visit is really short, it would not make sense for her to travel a long distance from the airport (there is only one international airport in Sri Lanka), to stay only a day or two at her destination.

Is it the case that SUMU does enjoy going to museums.: No

Is it the case that SUMU does have an opinion whether Tutankhamen was murdered.: Yes

*Is it the case that SUMU did enjoy any of the movies: The Mummy, Troy, Ann
a and the King, Elizabeth.: Yes*

###This set of answers will determine how interested Sumu is about historical sites. If she answers yes to all the 3 questions above, it shall be inferred that it is essential that her destination has historical significance. Conversely, if she answers no to all the above, she shall be categorized as being averse to historical destinations. If she answers yes to 1 or 2 of the questions, she will be classified as “indifferent” to historical attractions.

Is it the case that SUMU does enjoy going shopping.: Yes

Is it the case that SUMU does going to clubs, restaurants.: Yes

Is it the case that SUMU does enjoy going to theater.: Yes

###This set of answers will determine how interested Sumu is about urban life. The inference about the category will be done as explained above for interest in history.

Is it the case that SUMU does like to go hiking: Yes

Is it the case that SUMU does like scenery (you know the green stuff you don't have at MIT): Yes

*Is it the case that SUMU does care about the plight of endangered animals:
No*

###This set of answers will determine how interested Sumu is about wildlife and forests. The inference about the category will be done as explained above for interest in history.

```
Is it the case that SUMU does enjoy reading about different cultures.: No

Is it the case that SUMU does enjoy ethnic cuisine.: Yes

Is it the case that SUMU did enjoy any of the movies: Seven Years in Tibet
, Monsoon Wedding, Whale Rider, Ghandi.: Yes
```

###This set of answers will determine how interested Sumu is about cultures/traditions. The inference about the category will be done as explained above for interest in history.

```
Is it the case that SUMU does have an interest in sunbathing (mmmm warm sa
nd, lazing around: Yes

Is it the case that SUMU does have an interest in water related activities
 e.g.: snorkeling, jet-skiing: Yes

Is it the case that SUMU does have an interest in going to a beach resort:
 Yes
```

###This set of answers will determine how interested Sumu is about beaches and coastal areas. The inference about the category will be done as explained above for interest in history.

```
[SUGGESTED-LOCATION SUMU JAFFNA] 0.9920305
[SUGGESTED-LOCATION SUMU HIKKADUWA] 0.9920305
[SUGGESTED-LOCATION SUMU UDA-WALAWE] 0.9842187
[SUGGESTED-LOCATION SUMU GALLE] 0.9920305
 [SUGGESTED-LOCATION SUMU ANURADHAPURA] 0.984061
[SUGGESTED-LOCATION SUMU KATARAGAMA] 0.984061
[SUGGESTED-LOCATION SUMU RATNAPURA] 0.99203044
[SUGGESTED-LOCATION SUMU BADULLA] 0.99203044
[SUGGESTED-LOCATION SUMU TRINCOMALEE] 0.9920305
[SUGGESTED-LOCATION SUMU UNA-WATUNA] 0.99597543
[SUGGESTED-LOCATION SUMU KANDY] 0.99601525
[SUGGESTED-LOCATION SUMU COLOMBO] 0.9980076
          >>suggestion with highest certainty is Colombo
```

****

Figure 2 shows the state of the database after running the above example. It gives an idea about why Colombo was ranked higher than the other destinations. See Appendix A for further detail about the rules and inference.

```
:Show Joshua Database (matching what [default All]) All (opposite truth-value too? [default Yes]) Yes
True things
[AIRPORT-PROXIMITY-REQUIRED SUMU LOW]
[DATE-CONSTRAINED SUMU NO]
[CULTURAL-MOVIES SUMU YES]
[ETHNIC-FOOD SUMU YES]
[HAS-BUDGET SUMU NO]
[HISTORY-INTEREST SUMU INDIFFERENT]
[LIKE-BEACHRESORTS SUMU YES]
[SAFETY-REQUIREMENT SUMU MEDIUM]
[VACATION-LENGTH SUMU 10]
[CULTURE-INTEREST SUMU INDIFFERENT]
[LIKE-SCENERY SUMU YES]
[LIKE-SUNBATHING SUMU YES]
[HEALTH-REQUIREMENT SUMU MEDIUM]
[THEATER-INTEREST SUMU YES]
[VACCINATED SUMU YES]
[CULTURE-CURIOSITY SUMU NO]
[BUDGET SUMU DELUXE]
[SHOPPING-INTEREST SUMU YES]
[MUSEUM-INTEREST SUMU NO]
[SUGGESTED-LOCATION SUMU JAFFNA]
[SUGGESTED-LOCATION SUMU HIKKADUWA]
[SUGGESTED-LOCATION SUMU UDA-WALAWE]
[SUGGESTED-LOCATION SUMU GALLE]
[SUGGESTED-LOCATION SUMU RATNAPURA]
[SUGGESTED-LOCATION SUMU UNA-WATUNA]
[SUGGESTED-LOCATION SUMU BADULLA]
[SUGGESTED-LOCATION SUMU ANURADHAPURA]
[SUGGESTED-LOCATION SUMU KATARAGAMA]
[SUGGESTED-LOCATION SUMU TRINCOMALEE]
[SUGGESTED-LOCATION SUMU KANDY]
[SUGGESTED-LOCATION SUMU COLOMBO]
[LIKE-WATER-ACTIVITIES SUMU YES]
[CLUB-INTEREST SUMU YES]
[LIKE-ENDANGERED-SPECIES SUMU NO]
[KING-TUT-OPINION SUMU YES]
[URBAN-INTEREST SUMU ESSENTIAL]
[COAST-INTEREST SUMU ESSENTIAL]
[HAS-CHILDREN SUMU NO]
[IS-MARRIED SUMU YES]
[HISTORY-MOVIES SUMU YES]
[GOES-HIKING SUMU YES]
[WILDLIFE-INTEREST SUMU INDIFFERENT]
False things

"None"
```

Figure 2: The Joshua database after running the example explained above

What range of problems can the system handle?

For a traveler unfamiliar with Sri Lanka and the climate/economy of a South Asian country, this system gives a good indication about the best possible destinations and things that the traveler should be concerned about.

If the traveler's interests diverge greatly from the attractions provided by all 12 destinations it is difficult for the system to be of much use. There is an implicit assumption here that the user is aware of and is interested in some aspect of Sri Lanka.

For instance, if some user loves living in cities and hates coasts, different cultures, forests, etc., there is little that the system can do to make his/her vacation to a tropical, mostly rural island like Sri Lanka a success.

What does it know?

We used rules as our knowledge representation. Rules seem to be best suited to capture the knowledge required for the system to work as specified. The system will basically rank destinations according to certain conditions that will be specified in rules.

𝒲⌉ ⊓∫⌉⌈ 𝒥≀∫⟨⊓⊣ ⊔≀ ⌊⊓⟩↕⌈ ≀⊓∇ ∇⊓↕⌉↖⌊⊣∫⌉⌈ ∇⌉⊣∫≀\⟩\} ∫†∫⊔⌉⇕↙ 𝒜{⊔⌉∇ ⊔⟨⌉ ⟩\⊔∇≀⌈⊓⌋⊔⟩≀\ ⊔≀ ⊔⟨⟩∫ ⊔≀≀↕ ⟩\ ⌋↕⊣∫∫⇔ ⊣\⌈ ⊣{⊔⌉∇ ⊓∫⟩\} ⟩⊔ {≀∇ ⊔⟨⌉ ∈\⌈ √∇≀⌊↕⌉⇕↖ ∫⌉⊔⇔ ⊒⌉ ⌋⊣⇕⌉ ⊔≀ ⊔⟨⌉ ⌋≀\⌋↕⊓∫⟩≀\ ⊔⟨⊣⊔ 𝒥≀∫⟨⊓⊣ ⊒⟩↕↕ ⌊⌉ ⊣⌈⌉⊔⊓⊣⊔⌉ {≀∇ ⊔⟨⌉ ∇⌉⊔⊓⟩∇⌉⇕⌉\⊔∫ ≀{ ≀⊓∇ ∫†∫⊔⌉⇕↙

No other knowledge representation system seems to be better suited for our purposes. We briefly considered using semantic nets for our project due to a desire to use Haystack to implement the system. However, we concluded that semantic nets would not facilitate the knowledge acquisition or management required for this system, even though the UI capabilities of Haystack were particularly attractive.

As seen in Figure 3, the locations that are suggested depend on 9 attributes. The value taken by each of these attributes for each location is determined by analyzing the answers to direct questions put to the user. As shown in the previous annotated example, these questions do not require that the *user* infer what category of interest or which budget range he/she has. Similarly, safety and health risks are determined by the knowledge based system using rules that take into consideration factors such as whether or not you have dependents and the necessary immunity status.

These attributes can take boolean, numeric, or one of previously specified values. These values will increase the certainty of a specific location according to the rules. The rules implicitly contain properties of locations that can be matched to the properties of the user. For instance, if the budget of the user is considered "economy", locations with low average costs will increase in certainty.

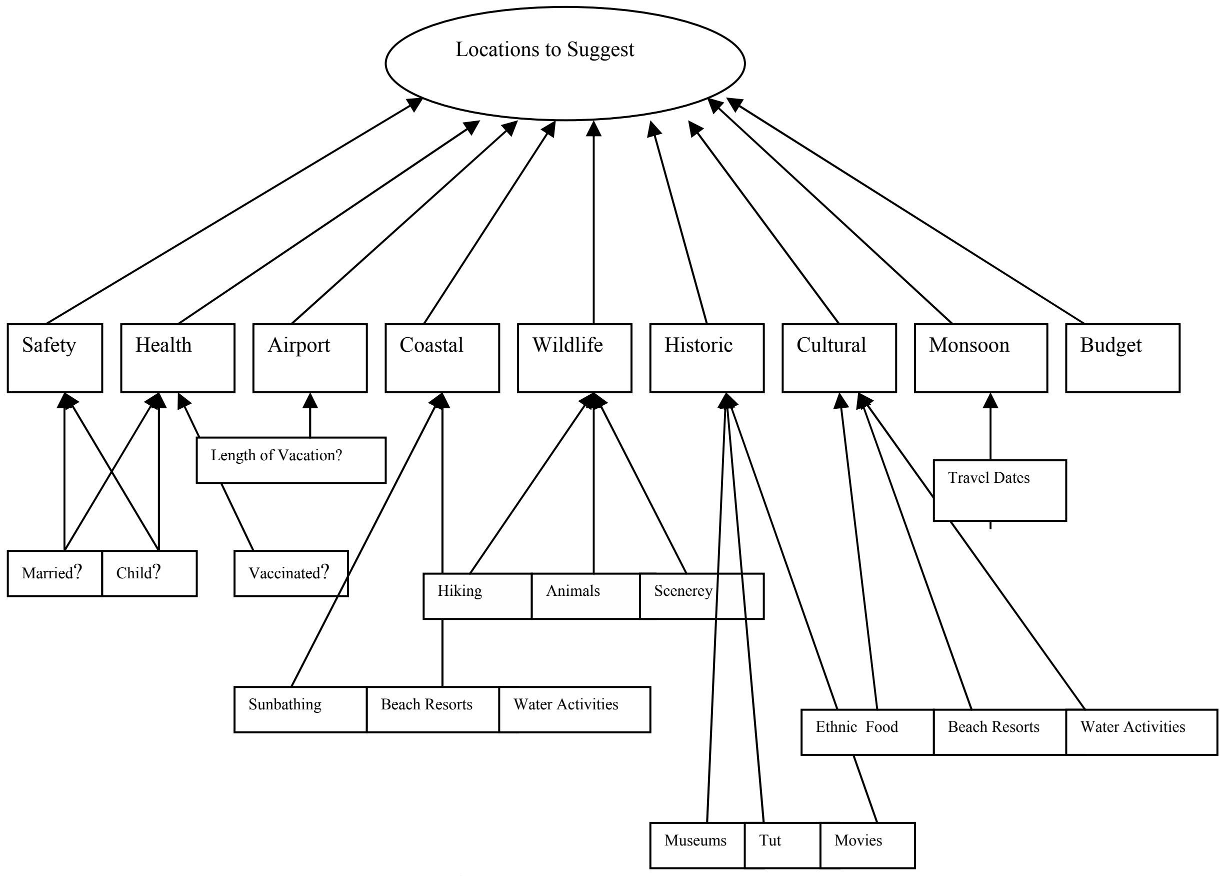

Figure 3: The tree-structure of the backward chaining rules-based system

For more details about the knowledge base, refer to Appendix A

How does it work?

The Travel Advisor was built based upon the KBS that was used for Problem Set 2. It works by backtracking on multiple rules that suggest locations. When it comes to a predicate that cannot be inferred from any other rule, it prompts the user for input. The user input is considered true and assigned a certainty of 1.0.

Most of the questions put to the user required either numeric or Boolean answers. We designed the system that way for better usability. Initial testing of the system showed that requiring the user to input one of a specified set of values was tedious.

Many rules could come to the conclusion that the suggested location should be X. But each rule has a specific certainty and the more rules that conclude that the location X should be suggested, the higher its certainty will become. In the end, the locations will the highest certainty will be the location that is most suited to the traveler's needs.

What went well?

** Well defined result*

The Travel Advisor was always successful in assigning the highest certainty to the most appropriate destination given the user's interests and demographics. Moreover, if the traveler is interested in visiting multiple destinations, this system gives additional knowledge about which locations are preferable over others for a particular user.

** Experience with Joshua from PS2*

The fact that we were somewhat familiar with Joshua and had used a KBS in that environment prior to this project made it easier for us to focus on the actual knowledge engineering instead of dealing with the logistics concerned with learning a new software tool.

** Attributes that were covered*

The way we categorized interest and other attributes worked well to quantify information and appropriately capture the knowledge in the domain we focused on.

** Realistic Scope*

Defining a realistic scope made reaching our target result more feasible. Restricting our system to work with 12 possible destinations meant that we could be more accurate in our inference. In an academic sense, including more locations would have mostly been a repetitive exercise The interesting aspect of the project lies in how we captured and categorized the knowledge and how we used it in the inference.

What went badly?

** Parenthesis error at bit 478204*

There were usability issues with Joshua that hindered us as we continues to get familiar with the system. Sometimes it was difficult to determine exactly what was going wrong. However, this was a good learning experience.

** Assigning certainty values, ranking importance*

Due to time constraints we were unable to do a full analysis on the optimal way to assign certainty and importance values. Improving this may make the overall system perform better.

** Significant last minute changes (that turned out to be useless and had to be undone…at 5am)*

A “brilliant” idea at 3am prompted us to restructure our rules, and change the certainty assignments and importance ranking for most of our rules. After over 2 hours of labor, we realized that the previous system we had, worked better and met our specification quite well.

***** *What did the system do badly?*

We would have preferred that the certainty values assigned to the different destinations were more widely spread. We were not sure how to accomplish this. In a future system, we envision that the certainty values will be more dispersed and the list will be ranked according to sorted certainty.

However, our current system was always successful in assigning the highest certainty (albeit sometimes different from others only at the 2nd decimal place) to the most appropriate destination according to our model.

What else did you learn from this?

One of the main things I learnt from this project was to have an early “code-freeze” or at least, a “feature-freeze”. It is unwise to do major changes to a system right before a deadline and is often better to just analyze the existing system and comment on its strengths and weaknesses.

I enjoyed this opportunity to think critically about what will comprise a knowledge-base and how inference can be done on it, given the representation that was chosen. For instance, it was not immediately clear how a traveler should be matched with a particular destination. After the model for our knowledge was chosen, it was also not obvious how rules should infer the best possible destination, given that both the traveler and the destination had multiple attributes that could be matched up.

Appendix A – *srilanka.lisp*

```
;;; Trip Planner Knowledge Base for Joshua
;;; 6.871 Final Project, Spring 2005
;;; Sarah Mirza, Sumudu Watugala

(in-package :ju)

; (ask [category-of-fund matt money-market] #'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 *srilanka-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 ((srilanka-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 *srilanka-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 #'srilanka-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 (vacation-length value-is-numeric-mixin)
:possesive-suffix "'s" :prompt1 "vacation length" :prompt2 "is")

(define-predicate-with-ancillary-info (group-size value-is-numeric-mixin)
:possesive-suffix "'s" :prompt1 "size of group" :prompt2 "is")

```
(define-predicate-with-ancillary-info (appx-budget value-is-numeric-mixin)
  :possesive-suffix "'s" :prompt1 "budget for the trip" :prompt2 "approximately is")

;;; Predicates that take one of a set of values
(define-predicate-with-ancillary-info (health-requirement value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "health requirement" :prompt2 "is"
  :possible-values (high medium low))

(define-predicate-with-ancillary-info (budget value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "budget category" :prompt2 "is"
  :possible-values (economy standard deluxe))

(define-predicate-with-ancillary-info (airport-proximity-required value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "airport proximity requirement" :prompt2 "is"
  :possible-values (high medium low))


(define-predicate-with-ancillary-info (safety-requirement value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "safety requirement" :prompt2 "is"
  :possible-values (high medium low))
(define-predicate-with-ancillary-info (suggested-location value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "recommended location in Sri Lanka" :prompt2 "is"
  :possible-values (COLOMBO KANDY TRINCOMALEE GALLE UNA-WATUNA ANURADHAPURA
RATNAPURA KATARAGAMA UDA-WALAWE HIKKADUWA JAFFNA BADULLA))
(define-predicate-with-ancillary-info (wildlife-interest value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "wildlife interest" :prompt2 "is"
  :possible-values (averse indifferent essential))
(define-predicate-with-ancillary-info (history-interest value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "history interest" :prompt2 "is"
  :possible-values (averse indifferent essential))
(define-predicate-with-ancillary-info (urban-interest value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "urban interest" :prompt2 "is"
  :possible-values (averse indifferent essential))
(define-predicate-with-ancillary-info (coast-interest value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "coastal interest" :prompt2 "is"
  :possible-values (averse indifferent essential))
(define-predicate-with-ancillary-info (culture-interest value-is-option-mixin)
  :possesive-suffix "'s" :prompt1 "cultural interest" :prompt2 "is"
  :possible-values (averse indifferent essential))


;;(define-predicate-with-ancillary-info (reason-for-travel value-is-option-mixin)
;;  :possesive-suffix "'s" :prompt1 "reason for taking a vacation" :prompt2 "is"
;;  :possible-values (relax sight-see both))


;;; boolean valued predicates

;;say has visited particular location

;;(define-predicate-with-ancillary-info (visited-before-colombo value-is-boolean-mixin)
;;  :possesive-suffix "" :prompt1 "has" :prompt2 "hasn't" :prompt3 "visted colombo before")



(define-predicate-with-ancillary-info (vaccinated value-is-boolean-mixin)
  :possesive-suffix "'s" :prompt1 "group has" :prompt2 "hasn't" :prompt3 "been properly vaccinated")

(define-predicate-with-ancillary-info (visited-before value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "has" :prompt2 "hasn't" :prompt3 "visted before")
```

```
(define-predicate-with-ancillary-info (has-budget value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "has" :prompt2 "hasn't" :prompt3 "a budget for the trip.")

(define-predicate-with-ancillary-info (date-constrained value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "has" :prompt2 "hasn't" :prompt3 "a date constraint")


(define-predicate-with-ancillary-info (travel-during-south-west-monsoons value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "is" :prompt2 "isn't" :prompt3 "travelling during the months of:January-
March")
(define-predicate-with-ancillary-info (travel-during-north-east-monsoons value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "is" :prompt2 "isn't" :prompt3 "travelling during the months of July-
September")

(define-predicate-with-ancillary-info (has-children value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "has" :prompt2 "doesn't have" :prompt3 "children")
(define-predicate-with-ancillary-info (is-married value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "is" :prompt2 "isn't" :prompt3 "married")


;;;indications of interest in urban life
(define-predicate-with-ancillary-info (shopping-interest value-is-boolean-mixin)
    :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "enjoy going shopping.")

(define-predicate-with-ancillary-info (theater-interest value-is-boolean-mixin)
    :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "enjoy going to theater.")

(define-predicate-with-ancillary-info (club-interest value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "going to clubs, restaurants.")

;;;indications of interest in history
(define-predicate-with-ancillary-info (museum-interest value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "enjoy going to museums.")

(define-predicate-with-ancillary-info (king-tut-opinion value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "have an opnion whether
Tutankhamen  was murdered.")

(define-predicate-with-ancillary-info (history-movies value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "did" :prompt2 "didn't" :prompt3 "enjoy any of the movies: The Mummy,
Troy, Anna and the King, Elizabeth.")


;;;indications of interest in culture
(define-predicate-with-ancillary-info (culture-curiosity value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "enjoy reading about different
cultures.")

(define-predicate-with-ancillary-info (cultural-movies value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "did" :prompt2 "didn't" :prompt3 "enjoy any of the movies: Seven Years in
Tibet, Monsoon Wedding, Whale Rider, Ghandi.")

(define-predicate-with-ancillary-info (ethnic-food value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "enjoy ethnic cuisine.")


(define-predicate-with-ancillary-info (like-scenery value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "like scenery (you know the green
stuff you don't have at MIT)")
```

```
(define-predicate-with-ancillary-info (goes-hiking value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "like to go hiking")

(define-predicate-with-ancillary-info (like-endangered-species value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "care about the plight of endangered
animals")

(define-predicate-with-ancillary-info (like-water-activities value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "have an interest in water related
activities e.g snorkelling, jet-skiing")

(define-predicate-with-ancillary-info (like-sunbathing value-is-boolean-mixin)
   :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "have an interest in sunbathing
(mmmm warm sand, lazing around")

(define-predicate-with-ancillary-info (like-beachresorts value-is-boolean-mixin)
  :possesive-suffix "" :prompt1 "does" :prompt2 "doesn't" :prompt3 "have an interest in going to a beach
resort")



;; 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: adequacy of Basic Insurance Coverage          ;;;
;;;                                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; RULES ABOUT: whether you need safety
(defrule need-for-safety1 (:backward :certainty 1.0 :importance 99)
  if [has-children ?who yes]
  then [safety-requirement ?who high])

(defrule need-for-safety2 (:backward :certainty 1.0 :importance 99)
  if [and [is-married ?who yes]
       [has-children ?who no]]
  then [safety-requirement ?who medium])

(defrule need-for-safety3 (:backward :certainty 1.0 :importance 99)
  if [and [is-married ?who no]
       [has-children ?who no]]
  then [safety-requirement ?who low])



;;; RULES ABOUT: whether you need health

(defrule need-for-health1 (:backward :certainty 1.0 :importance 98)
  if [vaccinated ?who no]
  then [health-requirement ?who high])

(defrule need-for-health2 (:backward :certainty 1.0 :importance 98)
  if [has-children ?who yes]
  then [health-requirement ?who high])

(defrule need-for-health3 (:backward :certainty 1.0 :importance 98)
  if [and [is-married ?who yes]
       [has-children ?who no]
       [vaccinated ?who yes]]
  then [health-requirement ?who medium])

(defrule need-for-health4 (:backward :certainty 1.0 :importance 98)
  if [and [vaccinated ?who yes]
  [has-children ?who no]
  [is-married ?who no]]
  then [health-requirement ?who low])


;;RULES ABOUT the monsoons

;; ;;; RULES ABOUT: safety requirement

(defrule safety-requirement-adequacy1 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
          [safety-requirement ?who medium]
          [safety-requirement ?who high]]
  then [suggested-location ?who KANDY])

(defrule safety-requirement-adequacy2 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]]
  then [suggested-location ?who TRINCOMALEE])

(defrule safety-requirement-adequacy3 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]
  [safety-requirement ?who high]]
  then [suggested-location ?who GALLE])

(defrule safety-requirement-adequacy4 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]]
  then [suggested-location ?who KATARAGAMA])

(defrule safety-requirement-adequacy5 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]]
  then [suggested-location ?who UNA-WATUNA])

(defrule safety-requirement-adequacy6 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]]
  then [suggested-location ?who HIKKADUWA])

(defrule safety-requirement-adequacy7 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
       [safety-requirement ?who medium]]
  then [suggested-location ?who COLOMBO])

(defrule safety-requirement-adequacy8 (:backward :certainty 0.5 :importance 95)
  if [safety-requirement ?who low]
  then [suggested-location ?who JAFFNA])
```

```
(defrule safety-requirement-adequacy9 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
       [safety-requirement ?who medium]]
  then [suggested-location ?who ANURADHAPURA])

(defrule safety-requirement-adequacy10 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]
  [safety-requirement ?who high]]
  then [suggested-location ?who BADULLA])

(defrule safety-requirement-adequacy11 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]
  [safety-requirement ?who high]]
  then [suggested-location ?who RATNAPURA])

(defrule safety-requirementurban-interest-adequacy12 (:backward :certainty 0.5 :importance 95)
  if [or [safety-requirement ?who low]
  [safety-requirement ?who medium]
  [safety-requirement ?who high]]
  then [suggested-location ?who UDA-WALAWE])


;; ;;; RULES ABOUT: health requirement

(defrule health-requirement-adequacy1 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who low]
       [health-requirement ?who medium]
       [health-requirement ?who high]]
  then [suggested-location ?who KANDY])

(defrule health-requirement-adequacy2 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who medium]
  [health-requirement ?who low]]
  then [suggested-location ?who TRINCOMALEE])

(defrule health-requirement-adequacy3 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who medium]
  [health-requirement ?who low]
  [health-requirement ?who high]]
  then [suggested-location ?who GALLE])

(defrule health-requirement-adequacy4 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who low]
  [health-requirement ?who medium]]
  then [suggested-location ?who KATARAGAMA])

(defrule health-requirement-adequacy5 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who low]
  [health-requirement ?who medium]]
  then [suggested-location ?who UNA-WATUNA])

(defrule health-requirement-adequacy6 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who low]
  [health-requirement ?who medium]]
  then [suggested-location ?who HIKKADUWA])

(defrule health-requirement-adequacy7 (:backward :certainty 0.5 :importance 94)
  if [or [health-requirement ?who low]
       [health-requirement ?who medium]
```

```
     [health-requirement ?who high]]
 then [suggested-location ?who COLOMBO])

(defrule health-requirement-adequacy8 (:backward :certainty 0.5 :importance 94)
 if [or [health-requirement ?who medium]
 [health-requirement ?who low]]
 then [suggested-location ?who JAFFNA])

(defrule health-requirement-adequacy9 (:backward :certainty 0.5 :importance 94)
 if  [health-requirement ?who low]
 then [suggested-location ?who ANURADHAPURA])

(defrule health-requirement-adequacy10 (:backward :certainty 0.5 :importance 94)
 if [or [health-requirement ?who low]
 [health-requirement ?who medium]]
 then [suggested-location ?who BADULLA])

(defrule health-requirement-adequacy11 (:backward :certainty 0.5 :importance 94)
 if [or [health-requirement ?who medium]
 [health-requirement ?who low]
 [health-requirement ?who high]]
 then [suggested-location ?who RATNAPURA])

(defrule health-requirementurban-interest-adequacy12 (:backward :certainty 0.5 :importance 94)
 if [or [health-requirement ?who medium]
 [health-requirement ?who low]]
 then [suggested-location ?who UDA-WALAWE])


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; RULES ABOUT DATE CONSTRAINTS



(defrule one-monsoon-season (:backward :certainty 0.5 :importance 97)
 if [and [date-constrained ?who yes]
      [travel-during-south-west-monsoons ?who yes]]
      then  [travel-during-north-east-monsoons ?who no])

(defrule date-contraint-south1 (:backward :certainty 0.5 :importance 96)
 if [and [date-constrained ?who yes]
      [travel-during-south-west-monsoons ?who yes]]
      then [suggested-location ?who JAFFNA])

(defrule date-contraint-south2 (:backward :certainty 0.5 :importance 96)
 if [and [date-constrained ?who yes]
      [travel-during-south-west-monsoons ?who yes]]
 then [suggested-location ?who TRINCOMALEE])

(defrule date-contraint-south3 (:backward :certainty 0.5 :importance 96)
 if [and [date-constrained ?who yes]
      [travel-during-south-west-monsoons ?who yes]]
      then [suggested-location ?who ANURADHAPURA])

(defrule date-contraint-north1 (:backward :certainty 0.5 :importance 96)
 if [and [date-constrained ?who yes]
      [travel-during-north-east-monsoons ?who yes]]
      then [suggested-location ?who COLOMBO])

(defrule date-contraint-north2 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who GALLE])

(defrule date-contraint-north3 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who UNA-WATUNA])

(defrule date-contraint-north4 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who UDA-WALAWE])

(defrule date-contraint-north5 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who HIKKADUWA])

(defrule date-contraint-north6 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who KANDY])

(defrule date-contraint-north7 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who KANDY])

(defrule date-contraint-north8 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who KATARAGAMA])

(defrule date-contraint-north9 (:backward :certainty 0.5 :importance 96)
  if [and [date-constrained ?who yes]
        [travel-during-north-east-monsoons ?who yes]]
        then [suggested-location ?who BADULLA])

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;RULES determining vacationer interests:


;;Interested in wildlife?

(defrule wildlife-preference1 (:backward :certainty 1.0 :importance 95)
  if [and [goes-hiking ?who yes]
    [like-scenery ?who yes]
    [like-endangered-species ?who yes]]
    then [wildlife-interest ?who essential])

(defrule wildlife-preference2 (:backward :certainty 1.0 :importance 95)
  if [and [goes-hiking ?who no]
    [like-scenery ?who no]
    [like-endangered-species ?who no]]
    then [wildlife-interest ?who averse])

(defrule wildlife-preference3 (:backward :certainty 1.0 :importance 95)
  if [and
     [or [goes-hiking ?who yes]
    [like-scenery ?who yes]
    [like-endangered-species ?who yes]]
    [or [goes-hiking ?who no]
    [like-scenery ?who no]
    [like-endangered-species ?who no]]]
    then [wildlife-interest ?who indifferent])

;;Interested in beaches/coastal areas?

(defrule coast-preference1 (:backward :certainty 1.0 :importance 95)
  if [and [like-sunbathing ?who yes]
    [like-water-activities ?who yes]
    [like-beachresorts ?who yes]]
    then [coast-interest ?who essential])

(defrule coast-preference2 (:backward :certainty 1.0 :importance 95)
  if [and [like-sunbathing ?who no]
    [like-water-activities ?who no]
    [like-beachresorts ?who no]]
    then [coast-interest ?who averse])

(defrule coast-preference3 (:backward :certainty 0.7 :importance 94)
  if [and  [or [like-sunbathing ?who yes]
    [like-water-activities ?who yes]
    [like-beachresorts ?who yes]]
   [or [like-sunbathing ?who no]
    [like-water-activities ?who no]
    [like-beachresorts ?who no]]]
    then [coast-interest ?who indifferent])

;;Interested in urban areas?

(defrule urban-preference1 (:backward :certainty 1.0 :importance 95)
  if [and [shopping-interest ?who yes]
    [club-interest ?who yes]
    [theater-interest ?who yes]]
  then [urban-interest ?who essential])

(defrule urban-preference2 (:backward :certainty 1.0 :importance 95)
  if [and [shopping-interest ?who no]
    [club-interest ?who no]
    [theater-interest ?who no]]
    then [urban-interest ?who averse])

(defrule urban-preference3 (:backward :certainty 1.0 :importance 90)
  if [and
   [or [shopping-interest ?who yes]
    [club-interest ?who yes]
    [theater-interest ?who yes]]
   [or [shopping-interest ?who no]
    [club-interest ?who no]
    [theater-interest ?who no]]]
    then [urban-interest ?who indifferent])

;;Interested in cultural venues?

(defrule culture-preference1 (:backward :certainty 0.9 :importance 95)
  if [and [culture-curiosity ?who yes]
    [ethnic-food ?who yes]
    [cultural-movies ?who yes]]
  then [culture-interest ?who essential])
```

```
(defrule culture-preference2 (:backward :certainty 0.9 :importance 95)
  if [and [culture-curiosity ?who no]
    [ethnic-food ?who no]
    [cultural-movies ?who no]]
    then [culture-interest ?who averse])

(defrule culture-preference3 (:backward :certainty 0.9 :importance 94)
  if [and
    [or [culture-curiosity ?who yes]
        [ethnic-food ?who yes]
        [cultural-movies ?who yes]]
    [or [culture-curiosity ?who no]
        [ethnic-food ?who no]
        [cultural-movies ?who no]]]
  then [culture-interest ?who indifferent])

;;Interested in history/civilization?

(defrule history-preference1 (:backward :certainty 0.9 :importance 95)
  if [and [museum-interest ?who yes]
    [king-tut-opinion ?who yes]
    [history-movies ?who yes]]
  then [history-interest ?who essential])

(defrule history-preference2 (:backward :certainty 0.9 :importance 95)
  if [and [museum-interest ?who no]
    [king-tut-opinion ?who no]
    [history-movies ?who no]]
  then [history-interest ?who averse])

(defrule history-preference3 (:backward :certainty 0.9 :importance 94)
  if [and
   [or [museum-interest ?who yes]
    [king-tut-opinion ?who yes]
    [history-movies ?who yes]]
  [or [museum-interest ?who no]
    [king-tut-opinion ?who no]
    [history-movies ?who no]]]
    then [history-interest ?who indifferent])

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Given a particular interest, suggesting appropriate locations

;;URBAN INTEREST

(defrule urban-interest-adequacy1 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who essential]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who KANDY])

(defrule urban-interest-adequacy2 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who TRINCOMALEE])

(defrule urban-interest-adequacy3 (:backward :certainty 0.5 :importance 90)
  if  [urban-interest ?who indifferent]
  then [suggested-location ?who GALLE])

(defrule urban-interest-adequacy4 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who KATARAGAMA])

(defrule urban-interest-adequacy5 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who UNA-WATUNA])

(defrule urban-interest-adequacy6 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who HIKKADUWA])

(defrule urban-interest-adequacy7 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who essential]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who COLOMBO])

(defrule urban-interest-adequacy8 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who JAFFNA])

(defrule urban-interest-adequacy9 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who ANURADHAPURA])

(defrule urban-interest-adequacy10 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who BADULLA])

(defrule urban-interest-adequacy11 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who RATNAPURA])

(defrule urban-interest-adequacy12 (:backward :certainty 0.5 :importance 90)
  if [or [urban-interest ?who averse]
  [urban-interest ?who indifferent]]
  then [suggested-location ?who UDA-WALAWE])


;;WILDLIFE INTEREST


(defrule wildlife-interest-adequacy1 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who essential]
  [wildlife-interest ?who indifferent]]
  then [suggested-location ?who KANDY])

(defrule wildlife-interest-adequacy2 (:backward :certainty 0.5 :importance 90)
  if [wildlife-interest ?who indifferent]
  then [suggested-location ?who TRINCOMALEE])

(defrule wildlife-interest-adequacy3 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who essential]
  [wildlife-interest ?who indifferent]]
  then [suggested-location ?who GALLE])
```

```
(defrule wildlife-interest-adequacy4 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who essential]
  [wildlife-interest ?who indifferent]]
  then [suggested-location ?who KATARAGAMA])

(defrule wildlife-interest-adequacy5 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who essential]
  [wildlife-interest ?who indifferent]]
  then [suggested-location ?who UNA-WATUNA])

(defrule wildlife-interest-adequacy6 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who averse]]
  then [suggested-location ?who HIKKADUWA])

(defrule wildlife-interest-adequacy7 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who averse]]
  then [suggested-location ?who COLOMBO])

(defrule wildlife-interest-adequacy8 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who averse]]
  then [suggested-location ?who JAFFNA])

(defrule wildlife-interest-adequacy9 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who essential]]
  then [suggested-location ?who ANURADHAPURA])

(defrule wildlife-interest-adequacy10 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who essential]]
  then [suggested-location ?who BADULLA])

(defrule wildlife-interest-adequacy11 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who essential]]
  then [suggested-location ?who RATNAPURA])

(defrule wildlife-interest-adequacy12 (:backward :certainty 0.5 :importance 90)
  if [or [wildlife-interest ?who indifferent]
  [wildlife-interest ?who essential]]
  then [suggested-location ?who UDA-WALAWE])

;;CULTURE INTEREST

(defrule culture-interest-adequacy1 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who essential]
  [culture-interest ?who indifferent]]
  then [suggested-location ?who KANDY])

(defrule culture-interest-adequacy2 (:backward :certainty 0.5 :importance 90)
  if [culture-interest ?who indifferent]
  then [suggested-location ?who TRINCOMALEE])

(defrule culture-interest-adequacy3 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who essential]
  [culture-interest ?who indifferent]]
  then [suggested-location ?who GALLE])
```

```
(defrule culture-interest-adequacy4 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who essential]
  [culture-interest ?who indifferent]]
  then [suggested-location ?who KATARAGAMA])

(defrule culture-interest-adequacy5 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who averse]
  [culture-interest ?who indifferent]]
  then [suggested-location ?who UNA-WATUNA])

(defrule culture-interest-adequacy6 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who indifferent]
  [culture-interest ?who averse]]
  then [suggested-location ?who HIKKADUWA])

(defrule culture-interest-adequacy7 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who indifferent]
  [culture-interest ?who essential]]
  then [suggested-location ?who COLOMBO])

(defrule culture-interest-adequacy8 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who indifferent]
  [culture-interest ?who essential]]
  then [suggested-location ?who JAFFNA])

(defrule culture-interest-adequacy9 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who indifferent]
  [culture-interest ?who essential]]
  then [suggested-location ?who ANURADHAPURA])

(defrule culture-interest-adequacy10 (:backward :certainty 0.5 :importance 90)
  if [or [culture-interest ?who indifferent]
  [culture-interest ?who essential]]
  then [suggested-location ?who BADULLA])

(defrule culture-interest-adequacy11 (:backward :certainty 0.5 :importance 90)
  if [culture-interest ?who indifferent]
  then [suggested-location ?who RATNAPURA])

(defrule culture-interest-adequacy12 (:backward :certainty 0.5 :importance 90)
  if [culture-interest ?who indifferent]
  then [suggested-location ?who UDA-WALAWE])

;;HISTORICAL INTEREST

(defrule history-interest-adequacy1 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who essential]
  [history-interest ?who indifferent]]
  then [suggested-location ?who KANDY])

(defrule history-interest-adequacy2 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who averse]]
  then [suggested-location ?who TRINCOMALEE])

(defrule history-interest-adequacy3 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who essential]
  [history-interest ?who indifferent]]
  then [suggested-location ?who GALLE])
```

```
(defrule history-interest-adequacy4 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who essential]
  [history-interest ?who indifferent]]
  then [suggested-location ?who KATARAGAMA])

(defrule history-interest-adequacy5 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who averse]
  [history-interest ?who indifferent]]
  then [suggested-location ?who UNA-WATUNA])

(defrule history-interest-adequacy6 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who averse]]
  then [suggested-location ?who HIKKADUWA])

(defrule history-interest-adequacy7 (:backward :certainty 0.5 :importance 90)
  if [history-interest ?who indifferent]
  then [suggested-location ?who COLOMBO])

(defrule history-interest-adequacy8 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who averse]]
  then [suggested-location ?who JAFFNA])

(defrule history-interest-adequacy9 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who essential]]
  then [suggested-location ?who ANURADHAPURA])

(defrule history-interest-adequacy10 (:backward :certainty 0.5 :importance 90)
  if [history-interest ?who indifferent]
  then [suggested-location ?who BADULLA])

(defrule history-interest-adequacy11 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who averse]]
  then [suggested-location ?who RATNAPURA])

(defrule history-interest-adequacy12 (:backward :certainty 0.5 :importance 90)
  if [or [history-interest ?who indifferent]
  [history-interest ?who averse]]
  then [suggested-location ?who UNA-WATUNA])


;; COAST INTEREST

(defrule coast-interest-adequacy1 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who KANDY])

(defrule coast-interest-adequacy2 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who indifferent]
  [coast-interest ?who essential]]
  then [suggested-location ?who TRINCOMALEE])

(defrule coast-interest-adequacy3 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who essential]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who GALLE])
```

```
(defrule coast-interest-adequacy4 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who KATARAGAMA])

(defrule coast-interest-adequacy5 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who essential]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who UNA-WATUNA])

(defrule coast-interest-adequacy6 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who essential]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who HIKKADUWA])

(defrule coast-interest-adequacy7(:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who essential]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who COLOMBO])

(defrule coast-interest-adequacy8 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who essential]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who JAFFNA])

(defrule coast-interest-adequacy9 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who ANURADHAPURA])

(defrule coast-interest-adequacy10 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who BADULLA])

(defrule coast-interest-adequacy11 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who RATNAPURA])

(defrule coast-interest-adequacy12 (:backward :certainty 0.5 :importance 90)
  if [or [coast-interest ?who averse]
  [coast-interest ?who indifferent]]
  then [suggested-location ?who UDA-WALAWE])


;;PROXIMIT TO AIRPORT

;;Determining required proximity

(defrule required-proximity-high (:backward :certainty 1.0 :importance 92)
  if [and [vacation-length ?who ?x]
       (<= ?x 2)]
  then [airport-proximity-required ?who high])

(defrule required-proximity-medium (:backward :certainty 1.0 :importance 92)
  if [and [vacation-length ?who ?x]
         (and (> ?x 2)
          (<= ?x 5))]
  then [airport-proximity-required ?who medium])
```

```
(defrule required-proximity-low (:backward :certainty 1.0 :importance 92)
  if [and [vacation-length ?who ?x]
       (> ?x 5)]
  then [airport-proximity-required ?who low])

;;;;;;;;;;;;;

(defrule proximity-airport-adequacy1 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who KANDY])

(defrule proximity-airport-adequacy2 (:backward :certainty 0.5 :importance 93)
  if [airport-proximity-required ?who low]
  then [suggested-location ?who TRINCOMALEE])

(defrule proximity-airport-adequacy3 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who GALLE])

(defrule proximity-airport-adequacy4 (:backward :certainty 0.5 :importance 93)
  if [airport-proximity-required ?who low]
  then [suggested-location ?who KATARAGAMA])

(defrule proximity-airport-adequacy5 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who UNA-WATUNA])

(defrule proximity-airport-adequacy6 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who HIKKADUWA])

(defrule proximity-airport-adequacy7 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]
 [airport-proximity-required ?who high]]
  then [suggested-location ?who COLOMBO])

(defrule proximity-airport-adequacy8 (:backward :certainty 0.5 :importance 93)
  if [airport-proximity-required ?who low]
  then [suggested-location ?who JAFFNA])

(defrule proximity-airport-adequacy9 (:backward :certainty 0.5 :importance 93)
  if [airport-proximity-required ?who low]
  then [suggested-location ?who ANURADHAPURA])

(defrule proximity-airport-adequacy10 (:backward :certainty 0.5 :importance 93)
  if [airport-proximity-required ?who low]
  then [suggested-location ?who BADULLA])

(defrule proximity-airport-adequacy11 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who RATNAPURA])

(defrule proximity-airport-adequacy12 (:backward :certainty 0.5 :importance 93)
  if [or [airport-proximity-required ?who low]
  [airport-proximity-required ?who medium]]
  then [suggested-location ?who UDA-WALAWE])


;;BUDGET


;;Determining available budget

(defrule no-budget (:backward :certainty 1.0 :importance 96)
  if [has-budget ?who no]
  then [budget ?who deluxe])

(defrule available-budget-economy (:backward :certainty 1.0 :importance 95)
  if [and [has-budget ?who yes]
  [appx-budget ?who ?x]
        [group-size ?who ?y]
      (<= ?x (* ?y 2000))]
  then [budget ?who economy])

(defrule available-budget-standard (:backward :certainty 1.0 :importance 95)
  if [and  [has-budget ?who yes]
        [appx-budget ?who ?x]
        [group-size ?who ?y]
      (and (> ?x (* ?y 2000))
          (<= ?x (* ?y 5000)))]
  then [budget ?who standard])

(defrule available-budget-deluxe (:backward :certainty 1.0 :importance 95)
  if [and  [has-budget ?who yes]
       [appx-budget ?who ?x]
        [group-size ?who ?y]
      (> ?x (* ?y 5000))]
  then [budget ?who deluxe])

;;;;;;;;;;;;;;;;;;
(defrule budget-adequacy1 (:backward :certainty 0.5 :importance 94)
  if [budget ?who deluxe]
  then [suggested-location ?who KANDY])

(defrule budget-adequacy2 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who economoy]
  [budget ?who standard]]
  then [suggested-location ?who TRINCOMALEE])

(defrule budget-adequacy3 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who economoy]
  [budget ?who standard]]
  then [suggested-location ?who GALLE])

(defrule budget-adequacy4 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who economoy]
  [budget ?who standard]]
  then [suggested-location ?who KATARAGAMA])

(defrule budget-adequacy5 (:backward :certainty 0.5 :importance 94)
  if [budget ?who economoy]
  then [suggested-location ?who UNA-WATUNA])

(defrule budget-adequacy6 (:backward :certainty 0.5 :importance 94)
  if [budget ?who economoy]
  then [suggested-location ?who HIKKADUWA])
```

```
(defrule budget-adequacy7 (:backward :certainty 0.5 :importance 94)
  if [budget ?who deluxe]
  then [suggested-location ?who COLOMBO])

(defrule budget-adequacy8 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who standard]
  [budget ?who deluxe]]
  then [suggested-location ?who JAFFNA])

(defrule budget-adequacy9 (:backward :certainty 0.5 :importance 94)
  if [or[budget ?who deluxe]
  [budget ?who standard]
  [budget ?who economoy]]
  then [suggested-location ?who ANURADHAPURA])

(defrule budget-adequacy10 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who economoy]
  [budget ?who standard]
  [budget ?who deluxe]]
  then [suggested-location ?who BADULLA])

(defrule budget-adequacy11 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who economoy]
  [budget ?who standard]
  [budget ?who deluxe]]
  then [suggested-location ?who RATNAPURA])

(defrule budget-adequacy12 (:backward :certainty 0.5 :importance 94)
  if [or [budget ?who standard]
  [budget ?who deluxe]]
  then [suggested-location ?who UDA-WALAWE])


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(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)))




;;-----------------------------------------------------------------------------------------------------------------------------------------------
```