Класификация на фериботни услуги

Константин Йовков

ф.н.25557, спец. ИИОЗ

Описание на проекта

Проектът представлява експертна система за класификация и предлагане на фериботни услуги, на базата на зададени характеристики на пътниците и багажа, който имат. Видовете фериботни услуги могат да се разделят на предложение за вид ферибот, в зависимост броя на пътниците и дали има деца или домашни любимци сред тях, предложение на най-скорошен ферибот при зададен максимум на цена и др. Характеристиките на един ферибот представляват съвкупности от инстанции на описания на каюти, помещения за превозни средства, наличие на ресторанти на ферибота и др.

Резултатът от търсенето представлява най-скорошен ферибот или най-евтин такъв, в завимост от зададените характеристики на търсене. Системата проверява валидността на зададените критерии за търсене - например, няма фериботи, на които се позволява едни възрастен да се качи с повече от едно дете, за което да отговаря, тоест - ако има двама възрастни с три деца, то тогава системата трябва да изведе съответното съобщение.

Системата е реализирана с помощта на платформата CLIPS.

Работа на системата

При стартиране системата работи по следната последователност:

1. Първоначално системата разполага с няколко начални факта, описващи различни типове фериботи и техните характеристики.
2. Системата задава последователно въпроси за всички характеристики за резервацията на ферибот. При отговор за всяка от тях се създава съответен факт, както и факт, че съответната характеристика е проверена.
3. След като всички характеристики са проверени се създава фактът, че всичко е проверено.

4. На базата на избрания тип ферибот (най-евтин или най-скорошен) се филтрират всички начални факти за фериботи и се връща най-подходящят (ако въобще има такъв).

5. Намереният факт се модифицира, като слотовете, в които са описани вместимост за коли, каравани и камиони се модифицират, спрямо критерия за търсене. Например, ако търсим място за автомобил и е намерен подходящ факт за ферибот, то броят за свободните места за автомобили се намаля с 1. Възможно е, когато всички тези слотове държи стойност 0, фактът за съответния ферибот да се премахне изцяло, тъй като това е индикатор, че местата на ферибота са напълно резервирани.

6. Извежда се съобщение, описващо най-скорошния или най-евтиния (в зависимост от търсените критерии) ферибот.

7. Системата задава въпрос дали потребителят би искал да направи нова резервация. В случай, че той желае, се изтриват всички създадени досега факти, с изключение на началните, защото тяхното състояние трябва да е консистентно.

Код на системата

Следва пълният код на системата. Реализиран и тестван е с помощта на езика CLIPS, версия 6.3. Освен тук, кодът може да се намери и във файла ferry.clp.

```
;===== ask questions functions
(deffunction ask-question (?question $?allowed-values)
    (printout t crlf)
    (printout t ?question)
    (printout t crlf)
    (bind ?answer (read))
    (if (lexemep ?answer) then
        (bind ?answer (lowcase ?answer)))
    (while (not (member ?answer ?allowed-values)) do
        (printout t crlf)
        (printout t ?question)
        (bind ?answer (read))
        (if (lexemep ?answer) then
            (bind ?answer (lowcase ?answer))))
    ?answer)

(deffunction ask-question-number-ans (?question)
    (printout t crlf)
    (printout t ?question)
```

```
    (printout t crlf)
    (bind ?answer (read))
    (while (not (numberp ?answer)) do
        (printout t crlf)
        (printout t ?question)
        (bind ?answer (read)))
    ?answer)

(deffunction ask-question-integer-ans (?question)
    (printout t crlf)
    (printout t ?question)
    (printout t crlf)
    (bind ?answer (read))
    (while (not (integerp ?answer)) do
        (printout t crlf)
        (printout t ?question)
        (bind ?answer (read)))
    ?answer)

(deffunction ask-question-integer-ans-in-bounds (?question ?lower-bound ?upper-bound)
    (bind ?answer
          (ask-question-integer-ans ?question))
    (while (not (<= ?lower-bound ?answer ?upper-bound))
        (bind ?answer
              (ask-question-integer-ans ?question)))
    ?answer)

(deffunction yes-or-no-p (?question)
    (bind ?response (ask-question ?question yes no y n))
    (if (or (eq ?response yes) (eq ?response y)) then
        TRUE
    else
        FALSE))

;===== templates

(deftemplate ferry
        (slot type)
        (slot from)
        (slot to)
        (slot departure-time (type INTEGER))
        (slot initial-price (type INTEGER))
        (slot children-per-person (type INTEGER))
        (slot animals-allowed)
        (slot restaurant)
        (slot cars (type INTEGER))
```

```
        (slot vans (type INTEGER))
        (slot trucks (type INTEGER))
        (slot rooms (type INTEGER))
)

;===== initial-facts

(deffacts initial
  (ferry
    (type catamaran)
    (from dover)
    (to calais)
    (departure-time 9)
    (initial-price 15)
    (animals-allowed yes)
    (children-per-person 2)
    (restaurant yes)
    (cars 50)
    (vans 20)
    (trucks 20)
    (rooms 100)
  )
  (ferry
    (type catamaran)
    (from dover)
    (to calais)
    (departure-time 10)
    (initial-price 5)
    (animals-allowed yes)
    (children-per-person 2)
    (restaurant yes)
    (cars 1)
    (vans 20)
    (trucks 20)
    (rooms 100)
  )
  (ferry
    (type ferry)
    (from liverpool)
    (to dublin)
    (departure-time 11)
    (initial-price 30)
    (animals-allowed yes)
    (children-per-person 1)
    (restaurant no)
    (cars 50)
```

```
    (vans 20)
  )
  (ferry
    (type car-ferry)
    (from hull)
    (to rotterdam)
    (departure-time 13)
    (initial-price 20)
    (children-per-person 4)
    (animals-allowed no)
    (restaurant no)
    (cars 100)
  )
)


;===== TODO: add more facts

;===== check for facts rules
(defrule check-from "From : Dover, Hull, Liverpool"
    =>
    (bind ?response (ask-question "Where do you depart from? (Dover, Hull, Liverpool)" dover
    (if (or (eq ?response dover) (eq ?response Dover)) then
        (assert (from dover))
    else
        (if (or (eq ?response hull) (eq ?response Hull)) then
            (assert (from hull))
        else
            (assert (from liverpool))))
    (assert (checked from)))


(defrule check-to "To : Calais, Rotterdam, Dublin"
    =>
    (bind ?response (ask-question "Where do you arrive in? (Calais, Rotterdam, Dublin)" cala
    (if (or (eq ?response calais) (eq ?response Calais)) then
        (assert (to calais))
    else
        (if (or (eq ?response Rotterdam) (eq ?response rotterdam)) then
            (assert (to rotterdam))
        else
            (assert (to dublin))))
    (assert (checked to)))

(defrule check-deparature-not-before "After: hour"
    =>
```

```
    (bind ?response (ask-question-number-ans "Earliest hour of departure? (between 6 and 20.
    (if (and (>= ?response 6) (<= ?response 20))
      then (assert (after ?response)))
    (assert (checked after)))

(defrule check-deparature-not-after "Before: hour"
    =>
    (bind ?response (ask-question-number-ans "Latest hour of departure? (between 6 and 20. T
    (if (and (>= ?response 6) (<= ?response 20))
      then (assert (before ?response)))
    (assert (checked before)))

(defrule check-adults "Adults count"
    =>
    (bind ?response (ask-question-integer-ans-in-bounds "Number of adults? (between 1 and 5)
    (assert (adults ?response))
    (assert (checked adults)))

(defrule check-children "Children count"
    =>
    (bind ?response (ask-question-number-ans "Number of children?"))
    (assert (children ?response))
    (assert (checked children)))

(defrule check-vehicle-type "Vehicle type: car, van or truck"
    =>
    (bind ?response (ask-question "What is your vehicle type? (car, van or truck)" car Car v
    (assert (vehicle ?response))
    (assert (checked vehicle)))

(defrule check-animals "Animals: yes/no"
    =>
    (if (yes-or-no-p "Do you bring animals with you? (yes/no)") then
        (assert (animals yes))
        else
                (assert (animals no)))
    (assert (checked animals)))

(defrule check-max-price "Maximum initial price"
    =>
    (bind ?response (ask-question-number-ans "Maximum initial price? (between 1 and 10000. T
    (if (and (>= ?response 1) (<= ?response 10000))
      then (assert (max-price ?response)))
    (assert (checked max-price)))

(defrule check-cheapest-or-earliest "Cheapest or earliest ferry wanted"
```

```
    =>
    (bind ?response (ask-question "Do you want the cheapest or the earliest ferry we have?"
    (if (or (eq ?response c) (eq ?response cheapest)) then
        (assert (c-o-r cheapest))
    else
       (assert (c-o-r earliest)))
    (assert (checked cheapest-or-earliest)))

(defrule check-room-need "Room needed: yes/no"
    =>
    (if (yes-or-no-p "Do you need a room on the ferry? (yes/no)") then
        (assert (room-needed yes))
        else
                (assert (room-needed no)))
    (assert (checked room-needed)))

(defrule proceed-with-another-reservation "Proceed: yes/no"
        (declare (salience -2))
  ?from <- (from ?)
  ?checked-from <- (checked from)

  ?to <- (to ?)
  ?checked-to <- (checked to)

  ?after <- (after ?)
  ?checked-after <- (checked after)

  ?before <- (before ?)
  ?checked-before <- (checked before)

  ?adults <- (adults ?)
  ?checked-adults <- (checked adults)

  ?children <- (children ?)
  ?checked-children <- (checked children)

  ?vehicle <- (vehicle ?)
  ?checked-vehicle <- (checked vehicle)

  ?animals <- (animals ?)
  ?checked-animals <- (checked animals)

  ?max-price <- (max-price ?)
  ?checked-max-price <- (checked max-price)

  ?cheapest-or-earliest <- (c-o-r ?)
```

```
    ?checked-cheapest-or-earliest <- (checked cheapest-or-earliest)

    ?room-needed <- (room-needed ?)
    ?checked-room-needed <- (checked room-needed)

    ?all-checked <- (all-checked)
          =>
          (if (yes-or-no-p "Do you want to proceed? (yes/no)") then
                  (retract ?from)
                  (retract ?checked-from)
                  (retract ?to)
                  (retract ?checked-to)
                  (retract ?after)
                  (retract ?checked-after)
                  (retract ?before)
                  (retract ?checked-before)
                  (retract ?adults)
                  (retract ?checked-adults)
                  (retract ?children)
                  (retract ?checked-children)
                  (retract ?vehicle)
                  (retract ?checked-vehicle)
                  (retract ?animals)
                  (retract ?checked-animals)
                  (retract ?max-price)
                  (retract ?checked-max-price)
                  (retract ?cheapest-or-earliest)
                  (retract ?checked-cheapest-or-earliest)
                  (retract ?room-needed)
                  (retract ?checked-room-needed)
                  (retract ?all-checked)
                  (refresh check-from)
                  (refresh check-to)
                  (refresh check-deparature-not-after)
                  (refresh check-deparature-not-before)
                  (refresh check-adults)
                  (refresh check-children)
                  (refresh check-vehicle-type)
                  (refresh check-animals)
                  (refresh check-cheapest-or-earliest)
                  (refresh check-max-price)
                  (refresh check-room-need)
                  (refresh proceed-with-another-reservation)
          )
)
```

```
(defrule all-checked "Assert that all characteristics were checked"
    (checked from)
    (checked to)
    (checked after)
    (checked before)
    (checked adults)
    (checked children)
    (checked vehicle)
    (checked animals)
    (checked max-price)
    (checked cheapest-or-earliest)
    (checked room-needed)
    =>
    (printout t crlf)
    (assert (all-checked)))

;===== validation rules

(defrule validate-children-per-person
        (all-checked)
        (children ?children)
        (adults ?adults)
        =>
        (if (> ?children ?adults) then
                (printout t "There are no ferries which allow more than one child per an adu
                (assert (error))))

(defrule validate-time-span
        (all-checked)
        (before ?before)
        (after ?after)
        =>
        (if (> ?after ?before) then
                (printout t "The time-span between the after and before departure time shoul
                (assert (error))))


;===== evaluation rules and functions


(deffunction earliest-predicate (?fact1 ?fact2)
   (< (fact-slot-value ?fact1 departure-time) (fact-slot-value ?fact2 departure-time))
)

(deffunction cheapest-predicate(?fact1 ?fact2)
   (< (fact-slot-value ?fact1 initial-price) (fact-slot-value ?fact2 initial-price))
```

```
)

(deffunction execute-find(?template ?from ?to ?max-price ?cars ?vans ?trucks ?after ?predica
   (bind ?min FALSE)
   (do-for-all-facts ((?f ?template))
                (and
                        (eq ?f:from ?from)
                        (eq ?f:to ?to)
                        (<= ?f:initial-price ?max-price)
                        (>= ?f:departure-time ?after)
                        (>= ?f:cars ?cars)
                        (>= ?f:vans ?vans)
                        (>= ?f:trucks ?trucks)
                )
                (if (or (not ?min) (funcall ?predicate ?f ?min))
                        then (bind ?min ?f))
        )
   (return ?min)
)

(deffunction calculate-price(?initial-price ?cars ?vans ?trucks ?rooms ?children ?pets)
        (bind ?result ?initial-price)
        (if (eq ?cars 1) then (bind ?result (+ ?result 20)))
        (if (eq ?vans 1) then (bind ?result (+ ?result 50)))
        (if (eq ?trucks 1) then (bind ?result (+ ?result 70)))
        (if (eq ?rooms 1) then (bind ?result (+ ?result 30)))
        (if (> ?children 0) then (bind ?result (+ ?result (* ?children 15))))
        (if (> ?pets 0) then (bind ?result (+ ?result (* ?pets 5))))
        (return ?result)
)

(defrule find-ferry
        (declare (salience -1))
        (all-checked)
        (from ?from)
        (to ?to)
        (after ?after)
        (max-price ?max-price)
        (c-o-r ?c-o-r)
        (vehicle ?vehicle)
        (children ?children)
        (room-needed ?room-needed)
        (animals ?animals)
        =>
        (bind ?result nil)
        (bind ?cars 0)
```

```
        (bind ?vans 0)
        (bind ?trucks 0)
        (bind ?rooms 0)
        (bind ?pets 0)
        (if (eq ?vehicle car) then (bind ?cars 1))
        (if (eq ?vehicle vans) then (bind ?vans 1))
        (if (eq ?vehicle truck) then (bind ?trucks 1))
        (if (eq ?room-needed yes) then (bind ?rooms 1))
        (if (eq ?animals yes) then (bind ?pets 1))
        (do-for-all-facts ((?f ferry))
                TRUE
                (if (eq ?c-o-r earliest) then
                        (bind ?result (execute-find ferry ?from ?to ?max-price ?cars ?vans ?
                else
                        (bind ?result (execute-find ferry ?from ?to ?max-price ?cars ?vans ?
                )
        )
        (if ?result then
                (printout t "We have booked a ferry for you. Here are the details:" crlf)
                (printout t "From: " (fact-slot-value ?result from) crlf)
                (printout t "To: " (fact-slot-value ?result to) crlf)
                (printout t "Departure time: " (fact-slot-value ?result departure-time) ":00
                (printout t "Price (in euros): " (calculate-price (fact-slot-value ?result i
                (printout t "See you on board!" crlf)
                (if (eq ?cars 1) then
                        (modify ?result (cars (- (fact-slot-value ?result cars) 1)))
                )
                (if (eq ?vans 1) then
                        (modify ?result (vans (- (fact-slot-value ?result vans) 1)))
                )
                (if (eq ?trucks 1) then
                        (modify ?result (trucks (- (fact-slot-value ?result trucks) 1)))
                )
        else
                (printout t "No ferry was found that matches your searching criteria..." crl
        (printout t "-----------------------------------------------" crlf)
)
```