Aufgabe 4

a)

Sei der Zustand gegeben durch das Tupel status(Raum in dem der Staubsauger ist, Raum 1 ist sauber, Raum 2 ist sauber).
Hierbei ist "Raum in dem der Staubsauger ist" ein Element aus {r1, r2},
"Raum 1, bzw Raum 2 ist sauber" als Element aus {wahr, falsch}.

Das bedeutet es gibt folgende 8 Kombinationsmöglichkeiten:
(r1, falsch, falsch)
(r2, falsch, falsch)
(r1, falsch, wahr)
(r2, falsch, wahr)
(r1, wahr, falsch)
(r2, wahr, falsch)
(r1, wahr, wahr)
(r2, wahr, wahr)

b)

status(r1, falsch, falsch) -> (S, status(r1, wahr, falsch))
status(r2, falsch, falsch) -> (S, status(r2, falsch, wahr))
status(r1, falsch, wahr) -> (S, status(r1, wahr, wahr))
status(r2, falsch, wahr) -> (R1, status(r1, falsch, wahr))
status(r1, wahr, falsch) -> (R2, status(r2, wahr, falsch))
status(r2, wahr, falsch) -> (S, status(r2, wahr, wahr))

c)

In unserem Regelsystem ist, solange mindestens ein Raum verschmutzt ist, stets genau eine Regelinstanz anwendbar. Es sind also nie mehrere Regelinstanzen in der Konfliktmenge und daher ist keine Konfliktlösungsstrategie (außer Refraktion) nötig.
Anzahl der Aktionen abhängig von Startkonfiguration:
- 3, wenn beide Räume verschmutzt sind (saugen, fahren, saugen)
- 2, wenn nur der entfernte Raum verschmutzt ist (fahren, saugen)
- 1, wenn nur der aktuelle Raum verschmutzt ist (saugen)

Aufgabe 5

a)

Es ist ineffizient, in jeder Iteration die Konfliktmenge komplett neu zu berechnen. Denn es können beim Matching genau zwei Fälle eintreten, die eine Änderung der Konfliktmenge nötig machen:

1) die erweiterte Faktenbasis erfüllt den kompletten Prämissenteil einer Regel, die noch nicht in der KM ist

2) die erweiterte Faktenbasis macht den Prämissenteil einer Regel aus der KM unerfüllbar (negative Prämisse).

Aufgabe 16

c)

Programm:

```
;;;======================================================
;;;   Self Learning Decision Tree Program
;;;
;;;     This program tries to determine the animal you are
;;;     thinking of by asking questions.
;;;
;;;     Jess Version 4.2 Example
;;;
;;;     To execute, merely load the file.
;;;======================================================


(deftemplate node
   (slot name)
   (slot type)
   (slot question)
   (slot yes-node)
   (slot no-node)
   (slot answer)
   (slot additional))                    Zusätzliches Feld für Verweis zu der
                                         Zusatzfragen node

(defrule initialize-1
   (not (node (name root)))
   =>
   (load-facts "animal.dat")
   (assert (current-node root)))

(defrule initialize-2
  (declare (salience 100))
  ?fact <- (next-gensym-idx ?idx)
  =>
  (retract ?fact)
  (setgen ?idx))


(defrule ask-decision-node-question
   ?node <- (current-node ?name)
   (node (name ?name)
         (type decision)
         (question ?question))
   (not (answer ?))
   =>
   (printout t ?question " (yes, dnk or no) ")    Do not know Ausgabe
   (assert (answer (read))))

(defrule bad-answer
```

```
  ?answer <- (answer ~yes&~no&~dnk)
  =>
  (retract ?answer))
```

Do not know Antwortmöglichkeit

```
(defrule proceed-to-yes-branch
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (yes-node ?yes-branch))
  ?answer <- (answer yes)
  =>
  (retract ?node ?answer)
  (assert (current-node ?yes-branch)))
```

```
(defrule proceed-to-get-dnk-branch
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (additional nil))
  ?answer <- (answer dnk)
  =>
  (retract ?answer)
  (assert (ask-additional-question)))
```

Falls Do not know gewählt wurde und kein zusätzlicher Frageknoten verfügbar ist soll eine neue Abfrage stattfinden

```
(defrule proceed-to-dnk-branch
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (additional ?dnk-branch))
  ?answer <- (answer dnk)
  =>
  (retract ?node ?answer)
  (assert (current-node ?dnk-branch)))
```

Falls Do not know gewählt wurde und ein zusätzlicher Frageknoten verfügbar ist soll mit diesem weitergemacht werden

```
(defrule ask-additional-question-rule
   (ask-additional-question)
   =>
   (printout t "Create new additional question?
(yes, no)")
   (assert (addQuestAnswer (read))))
```

Es wird gefragt ob eine neue Frage hinzugefügt werden soll und die Antwort validiert.

```
(defrule bad-addQuestAnswer
  ?answer <- (addQuestAnswer ~yes&~no)
  =>
  (retract ?answer))
```

```
(defrule addQuestAnswerYes
        (addQuestAnswer yes)
        =>
        (printout t "Additional Question:")
        (assert (addQuestText (readline)))
        (printout t "Yes-Animal:")
        (assert (addQuestAnimalYes (read)))
        (printout t "No-Animal:")
```

Wenn eine neue Frage eingegeben werden soll wird diese erfragt, samt Ja und Nein Antworten. Die Antworten werden assertet um sie im nächsten Schritt dann zu verarbeiten

```
        (assert (addQuestAnimalNo (read)))
        (assert (replace-dnk))
)

(defrule replace-dnk-node
        ?replace <- (replace-dnk)
        ?quest <- (addQuestText ?quest-text)
        ?yes <- (addQuestAnimalYes ?yes-text)
        ?no <- (addQuestAnimalNo ?no-text)
        ?current <- (current-node ?name)
        ?data <- (node (name ?name))
        =>
        (retract ?replace)
        (retract ?current)
        (bind ?newnode1 (gensym*))
        (bind ?newnode2 (gensym*))
        (bind ?newparent (gensym*))
        (modify ?data (additional ?newparent))
        (assert (node (name ?newparent)
(question ?quest-text) (type decision)
        (yes-node ?newnode1) (no-node ?newnode2)))
      (assert (node (name ?newnode1)
            (type answer)
            (answer ?yes-text)))
      (assert (node (name ?newnode2)
            (type answer)
            (answer ?no-text)))
        (assert (ask-try-again))
)
```

Hier werden zunächst für die neue Frage und Antworten namen generiert und der Name der neuen Frage bei der aktuellen Frage als Zusatzfrage eingefügt.

Dann werden Frage und Antworten zur Faktenbasis hinzugefügt.

```
(defrule addQuestAnswerNo
        ?node <- (current-node ?name)
        ?quest <- (addQuestAnswer ?answ)
        (addQuestAnswer no)
        =>
        (retract ?quest)
        (retract ?node)
        (printout t "Bye" ))
```

Hier wird das Programm beendet, da der Nutzer keine Zusatzfrage eingeben möchte.

```
(defrule proceed-to-no-branch
  ?node <- (current-node ?name)
  (node (name ?name)
      (type decision)
      (no-node ?no-branch))
  ?answer <- (answer no)
  =>
  (retract ?node ?answer)
  (assert (current-node ?no-branch)))

(defrule ask-if-answer-node-is-correct
  ?node <- (current-node ?name)
  (node (name ?name) (type answer) (answer ?value))
  (not (answer ?))
  =>
```

```
   (printout t "I guess it is a " ?value crlf)
   (printout t "Am I correct? (yes or no) ")
   (assert (answer (read))))

(defrule answer-node-guess-is-correct
   ?node <- (current-node ?name)
   (node (name ?name) (type answer))
   ?answer <- (answer yes)
   =>
   (assert (ask-try-again))
   (retract ?node ?answer))

(defrule answer-node-guess-is-incorrect
   ?node <- (current-node ?name)
   (node (name ?name) (type answer))
   ?answer <- (answer no)
   =>
   (assert (replace-answer-node ?name))
   (retract ?answer ?node))

(defrule ask-try-again
   (ask-try-again)
   (not (answer ?))
   =>
   (printout t "Try again? (yes or no) ")
   (assert (answer (read))))

(defrule one-more-time
   ?phase <- (ask-try-again)
   ?answer <- (answer yes)
   =>
   (retract ?phase ?answer)
   (assert (current-node root)))

(defrule no-more
   ?phase <- (ask-try-again)
   ?answer <- (answer no)
   =>
   (retract ?phase ?answer)
   (bind ?g (gensym*))
   (assert (next-gensym-idx (sub-string 4 (str-length ?g) ?g)))
   (save-facts "./animal.dat" MAIN::node MAIN::next-gensym-idx))

(defrule replace-answer-node
   ?phase <- (replace-answer-node ?name)
   ?data <- (node (name ?name)
                  (type answer)
                  (answer ?value))
   =>
   (retract ?phase)
   ; Determine what the guess should have been
   (printout t "What is the animal? ")
   (bind ?new-animal (read))
   ; Get the question for the guess
   (printout t "What question when answered yes ")
   (printout t "will distinguish " crlf "   a ")
```

```
  (printout t ?new-animal " from a " ?value "? ")
  (bind ?question (readline))
  (printout t "Now I can guess " ?new-animal crlf)
  ; Create the new learned nodes
  (bind ?newnode1 (gensym*))
  (bind ?newnode2 (gensym*))
  (modify ?data (type decision)
          (question ?question)
          (yes-node ?newnode1)
          (no-node ?newnode2))
  (assert (node (name ?newnode1)
          (type answer)
          (answer ?new-animal)))
  (assert (node (name ?newnode2)
          (type answer)
          (answer ?value)))
  ; Determine if the player wants to try again
  (assert (ask-try-again)))


(reset)
(run)
```

Sitzungsbeispiel

```
lionis:~{ki07} java jess.Main aufgabe16.clp

Jess, the Java Expert System Shell
Copyright (C) 2001 E.J. Friedman Hill and the Sandia Corporation
Jess Version 6.1p2 5/21/2003

Is the animal warm-blooded? (yes, dnk or no) dnk
Create new additional question? (yes, no)yes
Additional Question:Is it extinct?
Yes-Animal:Dodo
No-Animal:Mouse
Try again? (yes or no) yes
Is the animal warm-blooded? (yes, dnk or no) dnk
Is it extinct? (yes, dnk or no) yes
I guess it is a Dodo
Am I correct? (yes or no) yes
Try again? (yes or no) no
```