CPE/CSC 481 - W04

Knowledge-Based Systems

Final Exam

Instructor: Franz J. Kurfess

This is the final exam for CPE/CSC 481-W04 *Knowledge-Based Systems*. It is a take-home exam, and you may use textbooks, course notes, or other material, but you must formulate the text for your answers yourself. Please provide references if you use material other than the textbooks or course notes. You are not allowed to discuss the questions and answers with other students or anybody else.
If you need clarifications about questions, you can contact me via email, or see me during my office hours on Monday from 4-6 pm and Tuesday from 3-6 pm. The deadline for the exam is Tuesday, March 16, 2004, at 6:00 p.m.. You must submit a printed and signed copy of the exam, which you can either leave in the drop box in front of the CSC department office (room 14-154), or give it to me on Tuesday, March 16, 2004 during my office hours.
The number of points is indicated on the margin, with subtotals for the different tasks.

Student Name: **Date:**

Signature:

Task I – Multiple Choice Questions
Mark the correct answers (only one per question).

1. Which of the following statements is the best characterization of *conditional probability* for two events A and B in probability theory? 3

   ☐ the probability that both of two independent events occur
   ☐ the probability that one of two independent events occurs
   ☐ for two *dependent* events A and B, the probability that event A occurs given that B has already occurred
   ☐ for two *independent* events A and B, the probability that event A occurs given that B has already occurred

2. In which situations is a Bayesian approach to uncertain reasoning frequently used? 3

   ☐ In diagnostic systems, when the most likely cause for given symptoms needs to be determined.
   ☐ In prognostic systems, where the most likely future event is determined.
   ☐ In reactive systems, where a quick estimate is often more important than a lengthy but precise inference chain.
   ☐ In situations where only a small sample data set is available.

3. What is the historical background for *certainty factors*? 3

   ☐ It was developed by Thomas Bayes, an eighteenth-century British clergyman.
   ☐ It was initially developed in the 1960s by Lotfi Zadeh as an attempt to formalize common-sense reasoning
   ☐ The initial development was an ad-hoc attempt at dealing with uncertainty in expert system.
   ☐ It is a relatively recent development that subsumes some of the other approaches to uncertainty as special cases.

4. In which of the following approaches to uncertainty and approximate reasoning does the term *linguistic variable* play an important role? 3

   ☐ certainty factors
   ☐ fuzzy logic
   ☐ Bayesian reasoning
   ☐ Dempster-Shafer theory

5. What is the role of the *membership function* for fuzzy sets? 3

   ☐ its value indicates how strongly a particular element is affiliated with the set
   ☐ it returns `true` or `false` to indicate if an element is a member of the set
   ☐ it indicates the probability that an element is a member of the set
   ☐ it is one of several inference rules for fuzzy logic

Subtotal Task 1: 15

Task II – Short Questions

1. What are the main sources of uncertainty in knowledge-based systems? List at least five, and briefly discuss their relevance to your team project! 10

(a)

(b)

(c)

(d)

(e)

2. Explain the difference between *possibility* and *probability* in fuzzy logic. 10

3. Compare and Contrast the *spiral model* from Software Engineering, and the *linear model* used for knowledge-based systems. If possible, use an example such as your term project to illustrate important aspects. 10

Aspect	*Spiral Model*	*Linear Model*
Features	• • • • •	• • • • •
Differences	• • • • •	• • • • •
Advantages	• • • • •	• • • • •
Problems	• • • • •	• • • • •

4. Identify two possible sources of inefficiency in pattern matching for rule-based systems. For at least one of them, relate it to your own experiences in homework assignments or your team project. 5

   (a) Inefficiency 1:

   (b) Inefficiency 2:

Subtotal Task 2: 35

Task III – CLIPS Program: Factorial Forwards and Backwards

The purpose of this task is to examine the capabilities and limitations of CLIPS and JESS with respect to forward and backward chaining. Various strategies can be used to calculate the factorial from a given number:

1. Calculation in a function: All calculations are performed through a (recursive) function.
2. Pure rule-based programming: Forward chaining based on rules without procedural constructs (like loops), and without storing intermediate results via `assert`.
3. Rule-based programming with assert: Forward chaining based on rules without procedural constructs (like loops); asserting intermediate results is permitted.
4. Rule-based programming with procedural elements: Forward chaining based on rules with procedural constructs.
5. Backward chaining: Variations with or without `Assert` and loops.

These strategies may differ considerably with respect to memory usage and computation time. Your task is to outline the rules for the strategies listed below, and to discuss their respective advantages and problems. For some cases, example code is provided. Please not that the code does not necessarily represent excellent programming style, but it performs the required calculations (within limits). You can use it as a reference, but you can also write your own programs. The use of computers to run CLIPS or JESS is permitted. You need to explain the rules, and give an indication of the memory usage (e.g. in terms of number of facts asserted) and the computation time (e.g. in seconds, or in terms of number of rules fired, iterations through loops, recursive calls).

Calculation in a Function

```
; Factorial as function
;   From http://pesona.mmu.edu.my/~ytbau/tutor06.htm
;   Minor modifications by Franz J. Kurfess

(deffunction factorial (?a)
      (if (or (not (integerp ?a)) (< ?a 0)) then
              (printout t "Factorial Error!" crlf)
  else
              (if (= ?a 0) then
                   1
               else
                      (* ?a (factorial (- ?a 1))))))

; To run type the following at the prompt:
; CLIPS> (factorial 5)
```

a) *General Strategy*: Description of the approach.

b) *Code*: Explanation of the main elements. 3

c) *Properties*: Memory space, computation time, advantages, problems. 4

Forward Chaining, No Assert, No Loops

For this case, no sample code is provided, and you need to come up with your own solution. Ideally it should be working code, but you will get most points for a conceptually correct, but not necessarily executable code.Please note that the number of points for this section is higher.

a) *General Strategy*: Description of the approach. 5

b) *Code*: Explanation of the main elements, (partial) code. 7

c) *Properties*: Memory space, computation time, advantages, problems. 5

Forward Chaining, Assert, No Loops

```
; Factorial with Rules and Assert
;   from http://pesona.mmu.edu.my/~ytbau/rule_factorial.clp
;   Minor modifications by Franz J. Kurfess, March 2004

(deftemplate start)

(deftemplate  factorial
(slot n) (slot fn))

(deftemplate find-fact
(slot n))

(defrule  start
?s <- (start)
=>
(retract ?s)
(assert (factorial (n  0) (fn  1)))
(assert (find-fact (n  20))))

(defrule  stop
(find-fact (n  ?n))
?f <- (factorial (n ?n) (fn ?fn))
=>
(retract ?f)
(printout t "factorial " ?n " is "  ?fn  crlf))

(defrule  next
(find-fact  (n  ?n))
?f  <-  (factorial  (n    ?i&~?n)  (fn   ?fn))
=>
(retract  ?f)
(assert   (factorial  (n  (+  ?i  1))
                      (fn    (*  ?fn  (+  ?i  1))))))

; To run type the following commands at the prompt.
; You can also uncomment them and run it in a batch file.
;
; (reset)
; (assert (start))
; (run)
```

a) *General Strategy*: Description of the approach.

b) *Code*: Explanation of the main elements. 4

c) *Properties*: Memory space, computation time, advantages, problems. 4

Backward Chaining, Assert, Loops in JESS

This JESS program calculates the factorial of a number. The version given here contains constructs specific to JESS that enable the use of backward chaining in a less cumbersome way than CLIPS. The function `do-backward-chaining` marks a template as being eligible for backward chaining; in this case `factorial`. In addition to some other internal modifications to enable backward chaining, upon `(reset)` JESS may insert a fact with the prefix `need-` into working memory to trigger the backward chaining. In this case, it is `(need-factorial 20 nil)`. Such facts are also called *goal-seeking* or *trigger* facts.

```
; Factorial as backward chaining demo
;   Author: Ernest Friedman-Hill
;   Minor modifications by Franz J. Kurfess, March 2004

(do-backward-chaining factorial)

(defrule print-factorial-20
    (factorial 20 ?r1)
    =>
    (printout t "The factorial of 20 is " ?r1 crlf))

(defrule do-factorial
    (need-factorial ?x ?)
    =>
     ;; compute the factorial of ?x in ?r
    (bind ?r 1)
    (bind ?n ?x)
    (while (> ?n 1)
        (bind ?r (* ?r ?n))
        (bind ?n (- ?n 1)))
    (assert (factorial ?x ?r)))
(reset)
(run)
```

a) *General Strategy*: Description of the approach. 4

b) *Code*: Explanation of the main elements. 3

c) *Properties*: Memory space, computation time, advantages, problems. 4

Subtotal Task 3: 50

Total Points: 100