Knowledge-based systems in bioinformatics – 1MB602

Solutions to exam 2006-12-21

Please note that other solutions may exist for the problems given. The solutions provided are merely suggestions on how they can be solved.

1)

a)

```
(define (my-reverse lst)
  (accumulate (lambda (x y) (append y (list x)))
              '()
              lst))
```

b)

```
(define (my-reverse2 lst)
  (accumulate (lambda (x y)
                (if (pair? x)
                    (append y (list (my-reverse2 x)))
                    (append y (list x))))
              '()
              lst))
```

2)

```
(define (init-matrix nrow ncol init)
  (define (init-row count)
    (if (> count ncol)
        '()
        (cons init (init-row (+ count 1)))))
  (define (build count row)
    (if (> count nrow)
        '()
        (cons row (build (+ count 1) row))))
  (build 1 (init-row 1)))

(define (nth lst n)
      (if (= n 1)
          (car lst)
          (nth (cdr lst) (- n 1))))
```

```
(define (matrix nrow ncol init)
  (let ((data (init-matrix nrow ncol init)))
    (define (replace-nth lst n val)
      (if (= n 1)
          (cons val (cdr lst))
          (cons (car lst)
                (replace-nth (cdr lst) (- n 1) val))))
    (define (replace-val row col val lst)
      (if (= row 1)
          (cons (replace-nth (car lst) col val)
                (cdr lst))
          (cons (car lst)
                (replace-val (- row 1) col val (cdr lst)))))
    (define (set-value! row col val)
      (set! data (replace-val row col val data)))
    (define (get-value row col)
      (nth (nth data row) col))
    (define (get-row row)
      (nth data row))
    (define (get-col col)
      (accumulate (lambda (x y) (cons (nth x col) y))
                  '()
                  data))
    (define (print lst)
      (if (not (null? lst))
          (begin
            (display (car lst))
            (newline)
            (print (cdr lst)))))
    (define (dispatch m)
      (cond ((eq? m 'set) set-value!)
            ((eq? m 'get) get-value)
            ((eq? m 'get-col) get-col)
            ((eq? m 'get-row) get-row)
            ((eq? m 'print) (print data))
            (else (error "bad message sent to MATRIX" m))))
    dispatch))
```

3)

a)

$\models (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

A	**B**	**C**	**(A**	**→**	**(B → C))**	**→**	**((A → B)**	**→**	**(A → C))**
1	1	1		**1**	1	**1**	1	**1**	1
1	1	0		**0**	0	**1**	1	**0**	0
1	0	1		**1**	1	**1**	0	**1**	1
1	0	0		**1**	1	**1**	0	**1**	0
0	1	1		**1**	1	**1**	1	**1**	1
0	1	0		**1**	0	**1**	1	**1**	1
0	0	1		**1**	1	**1**	1	**1**	1
0	0	0		**1**	1	**1**	1	**1**	1

Implication true in all possible situations: valid and satisfiable.

$A \rightarrow \neg B, B \vee C, C \rightarrow A \models A \vee C$

A	**B**	**C**	**A → ¬B**	**B ∨ C**	**C → A**	**|=**	**A ∨ C**
1	1	1	0	1	1	**1**	1
1	1	0	0	1	1	**1**	1
1	0	1	1	1	1	**1**	1
1	0	0	1	0	1	**1**	1
0	1	1	1	1	0	**1**	1
0	1	0	1	1	1	**0**	0
0	0	1	1	1	0	**1**	1
0	0	0	1	0	1	**1**	0

Logical implication false in one case (row 6), otherwise true: satisfiable, contingent.

(A → B) ∧ (C → D) |= (A ∨ C) → (B ∨ D)

A	**B**	**C**	**D**	**(A → B)**	**∧**	**(C → D)**	**|=**	**(A ∨ C)**	**→**	**(B ∨ D)**
1	1	1	1	1	**1**	1	**1**	1	**1**	1
1	1	1	0	1	**0**	0	**1**	1	**1**	1
1	1	0	1	1	**1**	1	**1**	1	**1**	1
1	1	0	0	1	**1**	1	**1**	1	**1**	1
1	0	1	1	0	**0**	1	**1**	1	**1**	1
1	0	1	0	0	**0**	0	**1**	1	**0**	0
1	0	0	1	0	**0**	1	**1**	1	**1**	1
1	0	0	0	0	**0**	1	**1**	1	**0**	0
0	1	1	1	1	**1**	1	**1**	1	**1**	1
0	1	1	0	1	**0**	0	**1**	1	**1**	1
0	1	0	1	1	**1**	1	**1**	0	**1**	1
0	1	0	0	1	**1**	1	**1**	0	**1**	1
0	0	1	1	1	**1**	1	**1**	1	**1**	1
0	0	1	0	1	**0**	0	**1**	1	**0**	0
0	0	0	1	1	**1**	1	**1**	0	**1**	1
0	0	0	0	1	**1**	1	**1**	0	**1**	0

Logical implication true in all possible situations: valid and satisfiable.

b) (both tautologies shown)

|= (A → (B → C)) → ((A → B) → (A → C))

1.	A → (B → C)	P (extra)
2.	A → B	P (extra)
3.	A	P (extra)
4.	B → C	1, 3, →E
5.	B	2, 3, →E
6.	C	4, 5, →E
7.	A → C	3-6, →I
8.	(A → B) → (A → C)	2-7, →I
9.	(A → (B → C)) → ((A → B) → (A → C))	1-8, →I

$(A \rightarrow B) \wedge (C \rightarrow D) \models (A \vee C) \rightarrow (B \vee D)$

1.	$(A \rightarrow B) \wedge (C \rightarrow D)$	P
2.	$A \vee C$	P (extra)
3.	A	P (extra)
4.	$A \rightarrow B$	1, ∧E
5.	B	3, 4, →E
6.	$B \vee D$	5, ∨I
7.	$A \rightarrow (B \vee D)$	3-6, →I
8.	C	P (extra)
9.	$C \rightarrow D$	1, ∧E
10.	D	8, 9, →E
11.	$B \vee D$	10, ∨I
12.	$C \rightarrow (B \vee D)$	8-11, →I
13.	$B \vee D$	2, 7, 12, ∨E
14.	$(A \vee C) \rightarrow (B \vee D)$	2-13, →I

4)

1. **Expressivity**; a measure of the range of constructs that can be formally, flexibly, explicitly and accurately used to describe the components of an ontology.
2. **Rigour**; a measure of the satisfiability and consistency of the representation within the ontology. A model is satisfiable if none of the statements within contradict each other. Consistency within an ontology is a matter of encoding or conceptualising the knowledge in the same manner throughout the ontology.
3. **Semantics**. Clearly defined and well-understood semantics are essential if the ontology is to be used within the bioinformatics community for exchange of information.

5)

Use DFS in Iterative Deepening Search. Strategy: use DFS to a given depth, increment depth if no solution found and so on. Behaves like BFS, complete and optimal if finite branching factor.

6)

One solution is to use Iterative Deepening A* search which uses space linear in the length of the solution.

7)

a) see lecture notes
b) If examining a large hypothesis space it is a bad approach (infinite: impossible).
c) Gibbs classifier. Choose a hypothesis h from H at random according to the posterior distribution over H. Use h to predict the classification of the next instance x.

8)

A solution is said to be dominated by another solution if all its objective values are worse than the other solution. This principle could be used in an optimization problem where one wants to optimize different aspects, e.g., maximize profit of a company and minimize the company's toxic waste.

9)

A generational approach is often the better choice when the objective function has many local optima, a situation where the steady state approach often converges earlier at some local optima. Though, if one wants to find a good solution after a small number of iterations, the steady state approach would be better. More issues possible to consider here…

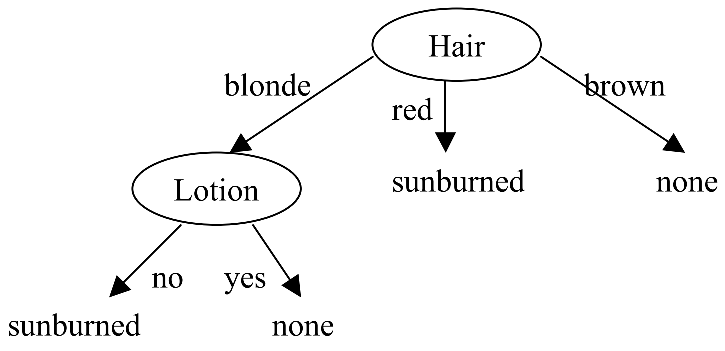
10)

$In_{hair}(T) =$
$4/8\ (-\ 2/4\ \log2(2/4) - 2/4\ \log2(2/4)) + 1/8\ (-\log2(1)) + 3/8\ (-\log2(1)) =$
$1/2$

$In_{height}(T) =$
$3/8\ (-\ 1/3\ \log2(1/3) - 2/3\ \log2(2/3)) + 3/8\ (-\ 1/3\ \log2(1/3) - 2/3\ \log2(2/3)) + 2/8\ (-\log2(1)) =$
$-\ 1/2\ \log2(2/3) - 1/4\ \log2(1/3) \approx$
0.69

$In_{weight}(T) = \ldots \approx 0.94$
$In_{lotion}(T) = \ldots \approx 0.61$

Choose attribute hair as the first test as this attribute will maximize the gain. Splitting on hair generates 3 subsets, one (blonde) which needs to be split further. After some calculation one can see that lotion is the best attribute to split on. Now we're done. Final tree


Hair
blonde
red
brown
Lotion
sunburned
none
no
yes
sunburned
none