CS 342 — Principles of Programming Languages May 15, 1991

Final

Please write your name here:

Before you begin, please take a moment to look over the entire test so that you can budget your time.

In this test the names "SASL, "CLU", etc. refer to the languages implemented in the textbook's interpreters. The "real" languages will always be referred to by the names "real SASL", "real CLU" etc.

Some global functions and clusters for the CLU and SASL questions are given on the next page.

Answers to essay questions will be judged on their completeness and clarity. However, please do not feel obligated to fill up all the empty space on the page. Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

Show any work for which you want partial credit.

problem	subject	points	score
1	CLU vs. Smalltalk	10	
2	Goals of Prolog	15	
3	Program `Set` in CLU	20	
4	Prolog: complete the program	10	
5	Prolog: write `rac`	10	
6	Prolog: trace queries	15	
7	SASL find the answer	20	
Total		100	

This exam (only) counts for 15% of your final grade.

Assume that the following CLU functions and cluster are in the global environment for all of the following CLU questions.

```
(define and (b1 b2) (if b1 1 b2))

(cluster List
    ; Exports: nil, null?, cons, car, cdr, rplaca, rplacd
    (rep type a d)
    (define nil () (List 0 0 0))
    (define null? (l) (= (type l) 0))
    (define cons (item l) (List 1 item l))
    (define car (l) (a l))
    (define cdr (l) (d l))
    (define rplaca (l a) (set-a l a))
    (define rplacd (l d) (set-d l d))
)
```

---

1. (10 points) Briefly describe two differences between CLU and Smalltalk.

2. (15 points) What are the goals of logic programming? What features in Prolog help achieve those goals?

3. (20 points) Write a mutable `Set` cluster in CLU by filling in the blanks in the following code.

```
(cluster Set
  ; Exports: empty-set, empty?, member?, insert, equal?
  (rep down)
  (define empty-set ()  ; TYPE: -> Set


  )
  (define empty? (s)    ; TYPE: Set -> Bool




  )
  (define member? (s x)  ; TYPE: Set * ElemType -> Bool




  )
  (define insert (s x)   ; MODIFIES: s   ; EFFECT: make x a member of s




  )
  (define equal? (s1 s2) ; TYPE: Set * Set -> Bool







  )










)
```

4. (10 points) Complete the following Prolog program by adding rules or facts as necessary so that the query `(infer? (studied me))` would be satisfied.

```
(infer (knows clu X)    from (thinks abstractly X) (hides information X))
(infer (knows lisp Everyone))
(infer (knows prolog X) from (logical X) (thinks abstractly X))
(infer (knows sasl X)   from (knows scheme X) (lazy X))
(infer (knows scheme X) from (knows lisp X) (knows lambda X))
(infer (studied X)      from (knows sasl X) (knows prolog X) (knows clu X))
```

5. (10 points) Define a predicate `rac` in Prolog that relates a non-empty list to a list without its last element. For example, the following queries should all be satisfied by your version of rac.

```
(infer? (rac (cons 27 nil) nil))
(infer? (rac (cons 7 (cons 99 (cons 86 nil))) (cons 7 (cons 99 nil))))
```

6. (20 points) Consider the following Prolog program.

```
(infer (rt X (app i X))
       from (print (app i X)))
(infer (rt X (app (app k X) Y))
       from (print (app (app k X) Y)))
(infer (rt W (app (app (app s X) Y) Z))
       from (print (app (app (app s X) Y) Z))
            (rt W (app (app X Y) (app X Z))))
```

For the following query give the output of the final query and any intermediate prints (in order). If the query gets into an infinite loop, say so (and give some of the output). If not, write "Satisifed" or "Not satisfied" at the end.

```
(infer? (rt Z (app (app (app s k) k) Z)) (print Z))
```

7. Give the output, if any, of each of the following SASL programs; that is, the output of the *last* expression in each part. If the expression denotes an infinite loop, say so. Write suspended thunks as "...".

   (a) (10 points)

```
(set mapcar (lambda (f l)
     (if (null? l) '()
      (cons (f (car l)) (mapcar f (cdr l))))))
(set octaveAbove (lambda (b) (+ b 7)))
(set thirdAbove (lambda (b) (+ b 2)))
(set octaves (cons 1 (mapcar octaveAbove octaves)))
(set thirds (mapcar thirdAbove octaves))
(car thirds)
(car (cdr thirds))
(car (cdr (cdr thirds)))
thirds
```

   (b) (10 points) In the following, the above definitions are also to be used.

```
(set fb (lambda (n1 n2 n3) (if (> (- n2 n1) 0) n2 n3)))
(set fb* (lambda (lst n)
   (fb n (car lst) (fb* (cdr lst) n))))

(fb* thirds 3)
```