(191)

Name: *Entry:* *Gp:*

Indian Institute of Technology Delhi
Department of Computer Sclence and Engineering

CSL302 Programming Languages Major Exam

May 4, 2007 15:30–17:30 Maximum Marks: 100

Open book and notes. Write your name, entry number and group at the top of each sheet in the blanks provided. Answer all questions in the space provided, in blue or black ink (no pencils, no red pens). Budget your time according to the marks. Do rough work on separate sheets.

Q1. (4x3 marks) **Unification.** For each of the following pairs of terms, if the *most general unifier* exists, present it as a *simultaneous substitution*, or else state why it does not exist:

1. $g(h(X,a),h(b,X))$ and $g(h(b,a),h(X,Z))$.

2. $g(h(X,a),Y)$ and $g(Z,h(b,X))$.

3. $g(h(X,a),Y)$ and $g(Y,h(b,X))$.

4. $g(h(a,b),X)$ and $g(Y,h(a,X))$.

Q2. (3+3+4 marks) **Prolog.** A list L_1 (which may even be empty) is a *prefix* of L_2 if some initial part of L_2 is exactly L_1. Likewise list L_3 is a *suffix* of L_4, if some final part of L_4 is L_3. L_5 is a *sublist* of L_6 if it appears somewhere within L_6. Suppose we are given a Prolog program append(L_1, L_2, L_3) for appending lists. Write Prolog programs for

1. `prefix(L1,L2) :-`

2. `suffix(L1,L2) :-`

3. `sublist(L1,L2) :-`

Q3. (16 marks) **Subtyping.** Type τ_1 is called a *subtype* of type τ_2 if wherever a value of type τ_2 is expected, a value of type τ_1 can safely be used. (This notion is at the core of the idea of subclass used for inheritance in Object-oriented programming). Suppose we have the following abstract grammar for types.

$$\tau \ ::= \ \textbf{unit} \mid \textbf{int} \mid \textbf{bool} \mid \textbf{real} \mid \tau_1 * \tau_2 \mid \tau_1 \rightarrow \tau_2$$

and wish to define a subtype relation between types defined inductively using the following rules: (a) unit is a subtype of any type; (b) int is a subtype of real; (c) τ_1 is a subtype of $\tau_1 * \tau_2$; (d) if τ_1' is a subtype of τ_1 and τ_2 is a subtype of τ_2', then $\tau_1 \rightarrow \tau_2$ is a subtype of $\tau_1' \rightarrow \tau_2'$; (e) the subtyping relation is reflexive and transitive.

Define a Prolog program subtype(T_1, T_2) to express the subtyping relation.

Q4. (8 marks) **Σ-homomorphisms.** Consider signature $\Sigma = \{0^{(0)}, 1^{(0)}, +^{(2)}\}$. Let $\mathcal{A}$ be the Σ-algebra with natural numbers as the carrier set, 0 interpreted as *zero*, 1 as *one* and $+$ as addition on natural numbers. What should the Σ-algebra $\mathcal{B}$ be, if the function *odd* : $\mathbb{N} \rightarrow \mathbb{B}$ is to be a Σ-homomorphism?
Carrier Set =

Interpretation of 0 is

Interpretation of 1 is

Interpretation of + is

Q5. (8+6 marks) **Call-by-value-result.** The *call-by-reference* parameter passing mechanism is necessary for writing procedures (such as swap) in which the procedure is intended to change the contents of its arguments. However, this mechanism is not a"clean abstraction" in that it suffers from the problem that any assignment to a reference argument immediate changes the corresponding global variable *before the completion of the execution of the procedure.* A better variant of this method is *call-by-value-result*, in which the contents of the argument variables are copied into fresh storage in the called procedure, the assignments are made to these locations, and finally, just before exit, the contents of

these new locations are copied back into the argument variables. Suppose P is a procedure with a single call-by-value-result parameter y and whose body is command c.

1. Provide big-step operational semantics for calling procedure P with (global) variable x as argument.

$$\frac{}{\beta \vdash \langle P(x), \sigma \rangle \implies \_} \quad \beta(P) = \ll \beta_1, \lambda y.c \gg, \beta(x) = l,$$

2. Suppose we have an implementation of procedure calls in an imperative language for the *call by value* parameter-passing mechanism. How will you modify this by adding some code before and after call-by-value procedure calls so that call-by-value-result can be simulated? *Explain precisely by providing pseudocode.*

Q6. (4x3 marks) **Lifetime and scope.** Suppose during the runtime execution of a program, an object o is created at time t_1 and is destroyed at time t_2, whereas it is associated via a *binding* to name x from time t_3 (when the binding is created) to t_4 (when the binding is destroyed). Express the relation that should hold (in terms of some of t_1, t_2, t_3, t_4) if

1. object o is garbage:

2. object o is always available (via name x):

3. name x is uninitialized for some time:

4. name x is a dangling reference:

Q7. (2+10 marks) **Definite iterations.** In mathematics, we have expressions such as $\sum_{i=a}^{i=b} f(i)$ and $\prod_{i=a}^{i=b} f(i)$. By which principle should we consider a special iterator command **foreach** i **from** a **to** b **do** C **od**,

where C is a command in which i appears as an integer variable?

Provide Big-step semantics for the construct **foreach** i **from** e_1 **to** e_2 **do** C **od**, assuming that (i) e_1 and e_2 are evaluated first to values v_1 and v_2 respectively; (ii) if $v_2 < v_1$, the command C is not executed, (iii) otherwise, C is serially executed $v_2 - v_1 + 1$ times with a fresh variable i taking successive values from v_1 to v_2 for each iteration of C; (iv) assume that i is only read but not otherwise changed within C, and that (v) variable i is not accessible after the end of the loop.

Q8. (8+8 marks) **λ-terms and their types.** Recall that Church numerals (encodings of the natural numerals in the λ-calculus) are $\lambda f.\lambda x.x, \lambda f.\lambda x.(f\ x), \lambda f.\lambda x.(f\ (f\ x)), \ldots$. If all these λ-terms were given the same type according to the typing rules given for abstraction and application, what would that type be?

Type of Church Numerals: _____

Any Y combinator satisfies the equation $Y\ f\ =_\beta\ f(Y\ f)$ for any function f. Assuming that the left and right sides of the equations must have the same type, what type should any Y combinator have? (*Make a suitable assumption about the type of f, and solve the constraints*).

Type of Y should be: _____