

CSL 765: Introduction to (Logic and Functional) Programming

I semester 2011-12

Tue 22 Nov 2011

13:00-15:00

Major

VI LT1

Max Marks 60

1. Please answer in the space provided on the question paper. The other sheets are only for rough work and will not be collected.
2. You may use any paper-based material including your class notes and any other text books.
3. You are not allowed to share reference material or rough pages during the exam.
4. You are not allowed to bring into the exam hall any electronic gadgets such as computers, mobile phones or calculators.
5. Please keep your identity card with you. You may be asked for it at any time for verification.

1. **(8 marks)** Use first-order resolution to prove the following argument where b is a unary predicate symbol and s is a binary predicate symbol.

$$\forall x[b(x) \rightarrow \forall y[\neg s(y, y) \leftrightarrow s(x, y)]] \models \neg \exists x[b(x)]$$

Solution. We need to show that the set $\{\forall x[b(x) \rightarrow \forall y[\neg s(y, y) \leftrightarrow s(x, y)]], \exists x[b(x)]\}$ is unsatisfiable. We first convert each formula into PCNF and then skolemize if necessary. $\exists x[b(x)]$ gets skolemized to $b(a)$. The PCNF of $\forall x[b(x) \rightarrow \forall y[\neg s(y, y) \leftrightarrow s(x, y)]]$ is obtained by moving the quantifier $\forall y$ and then converting the quantifier-free body of the formula into CNF. This gives us the logically equivalent formula $\forall x, y[\neg b(x) \vee ((\neg s(y, y) \vee \neg s(x, y)) \wedge (s(x, y) \vee s(y, y)))]$ which on distributing $\neg b(x)$ over the conjunction and standardizing the variables apart yields the following clauses.

$$\begin{aligned} C_1 &= \{\neg b(x_1), \neg s(y_1, y_1), \neg s(x_1, y_1)\} \\ C_2 &= \{\neg b(x_2), s(x_2, y_2), s(y_2, y_2)\} \\ C_3 &= \{b(a)\} \end{aligned}$$

Resolving on the atom b with the mgu $\{a/x_2, a/x_1\}$ yields

$$\begin{aligned} C_1 &= \{\neg s(y_1, y_1), \neg s(a, y_1)\} \\ C_2 &= \{s(a, y_2), s(y_2, y_2)\} \end{aligned}$$

Again resolving the two clauses with the mgu $\{a/y_2, a/y_1\}$ yields the empty clause.

2. (4+4+4 = 12 marks) Write a logic program (not a Prolog program, so no cuts and no extra controls)
- (a) which defines a binary tree,
 - (b) which defines a preorder traversal of a binary tree
 - (c) which transforms a given binary tree to a min-heap (under a less-than-or equal-to ordering) which preserves the structure of the original tree.

Solution.

We give the solution in the syntax of Prolog without using any of the extra control features of Prolog.

- (a) `bintree(empty).`
`bintree(node(Elt, Left, Right)) :-`
`bintree(Left), bintree(Right).`
- (b) `preorder(empty, []).`
`preorder(bintree(E, L, R), Pre):-`
`preorder(L, PreL), preorder(R, PreR), append([E|PreL], PreR, Pre).`
- (c) `heapify(empty, empty).`
`heapify(bintree(E, L, R), Heap) :-`
`heapify(L, HeapL), heapify(R, HeapR), addtoheap(E, HeapL, HeapR, Heap).`

`addtoheap(E, HeapL, HeapR, bintree(E, HeapL, HeapR)):-`
`lesseq(E, HeapL), lesseq(E, HeapR).`
`addtoheap(E, bintree(E1, L, R), HeapR, bintree(E1, HeapL, HeapR)) :-`
`E > E1, lesseq(E, HeapR), addtoheap(E, L, R, HeapL).`
`addtoheap(E, HeapL, bintree(E2, L, R), bintree(E2, HeapL, HeapR)) :-`
`E > E2, lesseq(E, HeapL), add2heap(E, L, R, HeapR).`
`lesseq(X, empty).`
`lesseq(X, bintree(E, L, R)) :- X ≤ E.`

3. **(10 marks)** The following algorithm was used by Russian peasants to multiply two positive integers. Prove the correctness of the algorithm i.e. prove that $russian(x, y) = xy$ for all integers $x, y > 0$.

$$russian(x, y) = \begin{cases} 0 & \text{if } y = 0 \\ russian(2x, y \operatorname{div} 2) & \text{if } y > 0 \wedge y \bmod 2 = 0 \\ x + russian(2x, y \operatorname{div} 2) & \text{if } y > 0 \wedge y \bmod 2 \neq 0 \end{cases}$$

Solution. We prove the correctness of the algorithm by induction on y .

Basis. $y = 0$. Then by the definition above we have $russian(x, 0) = 0 = x \cdot 0$

Induction Hypothesis (IH).

For all positive integers x and all integers z , $0 \leq k \leq z$, $russian(x, k) = x \cdot k$.

Induction Step. Let $y = z + 1$. Then $y > 0$ and we then have two cases to consider.

- Case $y \bmod 2 = 0$. Then since $y > 0$ we have $0 \leq y \operatorname{div} 2 < y$, $y = 2 \times (y \operatorname{div} 2)$ and $russian(x, y) = russian(2x, y \operatorname{div} 2) = 2x \times (y \operatorname{div} 2)$ (by the induction hypothesis) which in turn yields $x \cdot y$.
- Case $y \bmod 2 \neq 0$. Then again since $y > 0$ we have $0 \leq y \operatorname{div} 2 < y$, $y = 2 \times (y \operatorname{div} 2) + 1$ and $russian(x, y) = x + russian(2x, y \operatorname{div} 2) = x + 2x \times (y \operatorname{div} 2)$ (by the induction hypothesis) which yields $x \cdot (1 + 2 \times (y \operatorname{div} 2)) = x \cdot y$.

4. **(10 marks)** Prove that if f is an associative binary operation on a type α and e is the identity element of α , then $\text{foldl } f \ e = \text{foldr } f \ e$.

5. (10 marks) Consider the applied lambda-calculus built on the datatype of integers defined by

`datatype int = z | s of int | p of int | a of int*int`

where **z** stands for the number 0, **s** for the *successor* function on integers, **p** for the *predecessor* function and **a** for the addition operation on numbers. Define the function **fib** which defines the fibonacci numbers satisfying the equations $fib\ n = 0$, for all $n \leq 0$, $fib\ 1 = 1$, and for all $n > 1$, $fib\ n = fib(n-1) + fib(n-2)$.

Solution. An applied lambda calculus over a datatype with signature Σ is defined by the language

$$L, M ::= t \mid \lambda x[L] \mid (L\ M)$$

where $t \in T_{\Sigma}(X)$ for a set X of variables.

We first define the normal forms through rewrite rules. Let **Int** be the set of terms of the datatype and let the set of normal forms **IntNF** $\subset$ **Int** be defined as the set

$$\mathbf{IntNF} = \mathbf{IntNeg} \cup \{\mathbf{z}\} \cup \mathbf{IntPos}$$

where $\mathbf{IntNeg} = \{\mathbf{p}^n(\mathbf{z}) \mid n > 0\}$ and $\mathbf{IntPos} = \{\mathbf{s}^n(\mathbf{z}) \mid n > 0\}$. The set δ of reduction rules for the data-type **int** is defined by the following.

$$\begin{array}{l|l} \mathbf{p}(\mathbf{s}(x)) \longrightarrow_{\delta} x & \mathbf{s}(\mathbf{p}(x)) \longrightarrow_{\delta} x \\ \mathbf{a}(\mathbf{z}, \mathbf{z}) \longrightarrow_{\delta} \mathbf{z} & \\ \mathbf{a}(\mathbf{p}(x), y) \longrightarrow_{\delta} \mathbf{p}(\mathbf{a}(x, y)) & \mathbf{a}(\mathbf{s}(x), y) \longrightarrow_{\delta} \mathbf{s}(\mathbf{a}(x, y)) \\ \mathbf{a}(x, \mathbf{p}(y)) \longrightarrow_{\delta} \mathbf{p}(\mathbf{a}(x, y)) & \mathbf{a}(x, \mathbf{s}(y)) \longrightarrow_{\delta} \mathbf{s}(\mathbf{a}(x, y)) \end{array}$$

Let $\longrightarrow_{\delta}^*$ be the compatible-reflexive-transitive closure of $\longrightarrow_{\delta}$ (where compatibility is over all applied lambda terms as well), and let $=_{\delta}$ denote the equality relation generated by $\longrightarrow_{\delta}^*$. Further since δ -reduction is strongly normalizing and every term has a (unique) δ -normal form, we define $\longrightarrow_{\delta}^*!$ to be the maximal δ -reduction which yields a normal form.

Further we define the irreflexive-transitive-compatible relation $<_{\delta} \subseteq T_{\Sigma}(X) \times T_{\Sigma}(X)$ defined by the rules $\mathbf{p}(\mathbf{z}) <_{\delta} \mathbf{z}$, $\mathbf{z} <_{\delta} \mathbf{s}(\mathbf{z})$, $t <_{\delta} u \Leftrightarrow t \longrightarrow_{\delta}^*! m \wedge u \longrightarrow_{\delta}^*! n \wedge m <_{\delta} n$. We define $\leq_{\delta} = <_{\delta} \cup =_{\delta}$. We are then ready to define the the function $fib \stackrel{df}{=} (\mathbf{Y}\ fibfn)$ in the applied lambda calculus using the conditional expression $(L\ M\ N)$

$$fibfn \stackrel{df}{=} \lambda f\ n[((n \leq_{\delta} \mathbf{z})\ \mathbf{z}\ ((n =_{\delta} \mathbf{s}(\mathbf{z}))\ \mathbf{s}(\mathbf{z})\ (\mathbf{a}\ (f\ \mathbf{p}(n), f\ \mathbf{p}(\mathbf{p}(n))))))]]$$

6. (10 marks) For any binary relation ρ on the terms of a Σ -algebra, let

- ρ^c denote the compatible closure of ρ
- ρ^* denote the reflexive-symmetric-transitive closure of ρ .

Prove that $(\rho^c)^* = (\rho^*)^c$.

Solution.

Let $T_X(\Sigma)$ be the set of all terms of the Σ -algebra. We have the following definitions of $(\rho^c)^*$ and $(\rho^*)^c$ for any given relation ρ . We prove the following claims.

- Claim $(\rho^c)^* \subseteq (\rho^*)^c$. For any $t(\rho^c)^*u$ we have that for some $n \geq 0$, $t(\rho^c)^n u$. We proceed by induction on n to show that $(\rho^c)^n \subseteq (\rho^*)^c$ for all $n \geq 0$.

Basis. $n = 0$. $(\rho^c)^0 = \{(t, t) \mid t \in T_X(\Sigma)\} = \rho^0 = (\rho^0)^c \subseteq (\rho^*)^c$.

Induction Hypothesis (IH).

$$(\rho^c)^m \subseteq (\rho^*)^c \text{ for all } m, 0 \leq m \leq n$$

Induction Step. Assume $t(\rho^c)^{n+1}u$. Then there exists s such that $t(\rho^c)s(\rho^c)^n u$. By the induction hypothesis we have $s(\rho^*)^c u$. Since we have $t(\rho^c)s$ we proceed by induction on the structure of the terms t and s . If $t \rho s$ then there is nothing to prove. Otherwise, there exists a k -ary root operator o and terms $t_1, \dots, t_k, s_1, \dots, s_k$ such that for all $1 \leq j \leq k$, $t_j(\rho^c)s_j$. By the induction hypothesis we have $t_j(\rho^*)^c s_j$ for each j . Hence $t(\rho^*)^c s$, from which the claim follows.

- Claim $(\rho^c)^* \supseteq (\rho^*)^c$. Assume $t(\rho^*)^c u$. We proceed by induction on the structure of t and u to prove this claim. If $t \rho^* u$ then there must be some $n \geq 0$ such that $t \rho^n u$. We may show by induction on n that for all $n \geq 0$, $\rho^n \subseteq (\rho^c)^n \subseteq (\rho^c)^*$. Otherwise there exists a k -ary operator o for some $k \geq 0$ such that $t \equiv o(t_1, \dots, t_k)$, $u \equiv o(u_1, \dots, u_k)$ with $t_j(\rho^*)^c u_j$ for each j , $1 \leq j \leq k$. By the induction hypothesis we have that each $t_j(\rho^c)^* u_j$ from which it follows that $t(\rho^c)^* u$.