CSE 340 Spring B 2022  
HOMEWORK 2: Type Checking  
Claudio Rodriguez Rodriguez

Solution must be typed.

**You should show your work for all problems.**

**Problem 1 (20 points + 40 points).** Consider the following type declarations

TYPE

A1 : integer;

A2 : pointer to float; A3 : pointer to integer;

T1 : structure { x : integer; }

T2 : structure { x : A1; next : pointer to integer; } T3 : structure { a : integer; b : float; }

T4 : structure { b : float; a : integer; }

T5 : structure { a : pointer to T5; b : pointer to T6; c : pointer to T7; } T6 : structure { a : pointer to T6; b : pointer to T5; c : pointer to T5; } T7 : structure { a : pointer to T6; b : pointer to T7; c : pointer to T9; } T8 : structure { a : pointer to T7; b : pointer to T6; c : pointer to T10; }

T9 : array [4][5] of T8; // array 4 rows 5 columns T10 : array [4][5] of T7;

Assuming the most permissive definition of structural equivalence, which types are structurally equivalent? Draw the table step by step.

Step 1 – Initialize everything to true

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A2 | A3 | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| A1 | true | true | true | true | true | true | true | true | true | true | true | true | true |
| A2 |  | true | true | true | true | true | true | true | true | true | true | true | true |
| A3 |  |  | true | true | true | true | true | true | true | true | true | true | true |
| T1 |  |  |  | true | true | true | true | true | true | true | true | true | true |
| T2 |  |  |  |  | true | true | true | true | true | true | true | true | true |
| T3 |  |  |  |  |  | true | true | true | true | true | true | true | true |
| T4 |  |  |  |  |  |  | true | true | true | true | true | true | true |
| T5 |  |  |  |  |  |  |  | true | true | true | true | true | true |
| T6 |  |  |  |  |  |  |  |  | true | true | true | true | true |
| T7 |  |  |  |  |  |  |  |  |  | true | true | true | true |
| T8 |  |  |  |  |  |  |  |  |  |  | true | true | true |
| T9 |  |  |  |  |  |  |  |  |  |  |  | true | true |
| T10 |  |  |  |  |  |  |  |  |  |  |  |  | true |

Step 2 – First pass

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A2 | A3 | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| A1 | true | false | false | false | false | false | false | false | false | false | false | false | false |
| A2 |  | true | false | false | false | false | false | false | false | false | false | false | false |
| A3 |  |  | true | false | false | false | false | false | false | false | false | false | false |
| T1 |  |  |  | true | false | false | false | false | false | false | false | false | false |
| T2 |  |  |  |  | true | false | false | false | false | false | false | false | false |
| T3 |  |  |  |  |  | true | false | false | false | false | false | false | false |
| T4 |  |  |  |  |  |  | true | false | false | false | false | false | false |
| T5 |  |  |  |  |  |  |  | true | true | false | false | false | false |
| T6 |  |  |  |  |  |  |  |  | true | false | false | false | false |
| T7 |  |  |  |  |  |  |  |  |  | true | true | false | false |
| T8 |  |  |  |  |  |  |  |  |  |  | true | false | false |
| T9 |  |  |  |  |  |  |  |  |  |  |  | true | true |
| T10 |  |  |  |  |  |  |  |  |  |  |  |  | true |

Structurally Equivalent (SE)

* A1 not SE to any
* A2 not SE to any
* A3 not SE to any
* T1 not SE to any because it’s the only struct with 1 property
* T2 has 2 properties and one of them is a pointer, T3 and T4 don’t, so it’s not SE to any
* T3 only other possible is T4, but first property is different, so not SE to any
* T4 possible SE are T2 and T3 and they are already false
* T5 seems SE to T6, but not to T7 or T8 because of the property c
* T6 is like T5, they seem SE, so we leave them equivalent. But it is not SE to T7 and T8
* T7 and T8 seem SE, but both not equivalent to T9 and T10
* T9 and T10 seem SE

Step 3 – Second pass

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A2 | A3 | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| A1 | true | false | false | false | false | false | false | false | false | false | false | false | false |
| A2 |  | true | false | false | false | false | false | false | false | false | false | false | false |
| A3 |  |  | true | false | false | false | false | false | false | false | false | false | false |
| T1 |  |  |  | true | false | false | false | false | false | false | false | false | false |
| T2 |  |  |  |  | true | false | false | false | false | false | false | false | false |
| T3 |  |  |  |  |  | true | false | false | false | false | false | false | false |
| T4 |  |  |  |  |  |  | true | false | false | false | false | false | false |
| T5 |  |  |  |  |  |  |  | true | false | false | false | false | false |
| T6 |  |  |  |  |  |  |  |  | true | false | false | false | false |
| T7 |  |  |  |  |  |  |  |  |  | true | false | false | false |
| T8 |  |  |  |  |  |  |  |  |  |  | true | false | false |
| T9 |  |  |  |  |  |  |  |  |  |  |  | true | false |
| T10 |  |  |  |  |  |  |  |  |  |  |  |  | true |

Structurally Equivalent (SE)

* T5 and T6 are compared again. The property c is not SE so we flip the result to false.
* T7 and T8 are compared again. The property a and b are flipped, and because of that they are not SE. (T6 and T7 are not Structurally equivalent because of property c).
* T9 and T10 are compared again. T8 and T7 are not Structurally Equivalent, and we set it to false as well.

Consider the following variable declarations in conjunction to the above type declarations

VAR // var declaration section s : T9;

t : T9;

u : T10;

v : array [5][4] of T8; w, z : struct {

int a;

struct T5\* next;

};

x, y : struct {

int a;

struct T5\* next;

};

f : function of T9 returns int; g : function of T9 returns A1; m : int;

n : A1;

Assume that assignments between variables are allowed if the types of the variables are equivalent. For each of the following, list all type equivalence schemes under which the expression is valid. Consider name equivalence, internal name equivalence, and structural equivalence for each case. Assume that if two variables are equivalent under name equivalence, they are also equivalent under internal name equivalence.

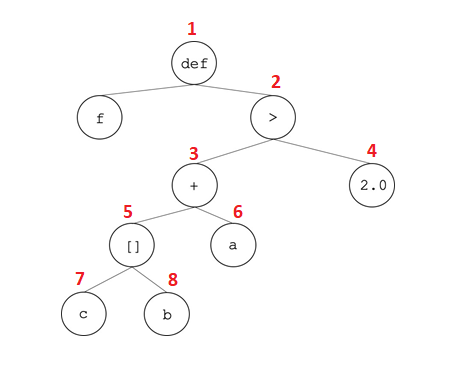
* s = t;
* t = u;
* u = v;
* v = w;
* w = z;
* z = x;
* m = f(s)
* n = f(u)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name Equivalence | Internal Name Equivalence | Structural Equivalence |
| s = t; | YES | YES | YES |
| t = u; | NO | NO | NO |
| u = v; | NO | NO | NO |
| v = w; | NO | NO | NO |
| w = z; | NO | YES | YES |
| z = x; | NO | NO | YES |
| m = f(s) | NO | YES | YES |
| n = f(u) | NO | NO | YES |

**Problem 2 (10 points).** Consider the following definition

fun f(a, b, c) = c[b] + a > 2.0

Using Hindley-Milner type inference, determine the type of f.



**Type of f is Boolean per table below**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| f | (T1, T2, T3) -> T4 | (T1, Int, Array of T5) -> T4 | (Float, Int, Array of Float) -> Bool |
| a | T1 | T1 | Float |
| b | T2 | Int | Int |
| c | T3 | Array of T5 | Array of Float |
| 1 |  |  |  |
| 2 | Bool | Bool | Bool |
| 3 |  | Float | Float |
| 4 | Float | Float | Float |
| 5 |  | T5 | Float |
| 6 |  | Infer float | Float |
| 7 | Array of T5 | Array of T5 | Array of Float |
| 8 | Int | Int | Int |

**Problem 3 (15 points).** Consider the following definition:

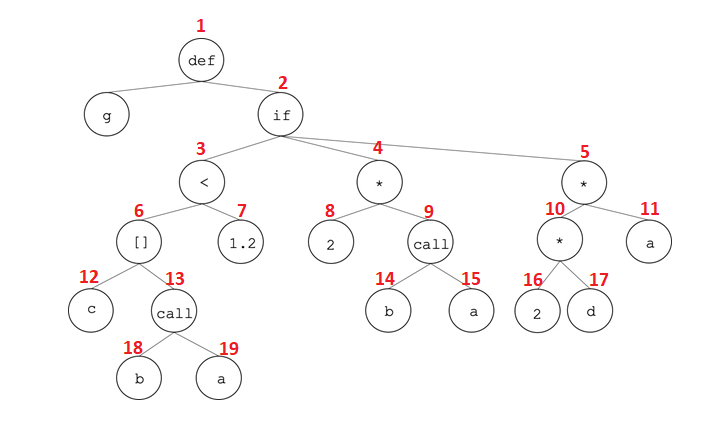
fun g(a, b, c, d) = if c[b(a)] < 1.2 then

2 \* b(a)

else

2 \* d \* a

Using Hindley-Milner type inference, determine the type of g.



**Type of g is Int per table below**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| g | (T1,T2,T3,T4) -> T5 | (T1, T1->int, Array of Float, int) -> T5 | (int, int -> int, Array of Float, int) -> T5 | (int, int -> int, Array of Float, int) -> int |
| a | T1 | T1 | Int | Int |
| b | T2 | T1 -> int | Int -> int | Int -> int |
| c | T3 | Array of Float | Array of Float | Array of Float |
| d | T4 | int | Int | Int |
| 1 |  |  |  |  |
| 2 |  |  |  | Int |
| 3 | Bool | Bool | Bool | Bool |
| 4 | int | int | Int | Int |
| 5 |  |  | Int | Int |
| 6 | Float | Float | Float | Float |
| 7 | Float | Float | Float | Float |
| 8 | int | int | Int | Int |
| 9 | int | int | Int | Int |
| 10 | Infer int | Int | Int | Int |
| 11 | T1 | Infer int | Int | Int |
| 12 | Array of T6 | Array of Float | Array of Float | Array of Float |
| 13 | Int | Int | int | int |
| 14 | T1 -> int | T1 -> int | Int -> int | Int -> int |
| 15 | T1 | T1 | Int | Int |
| 16 | int | Int | Int | Int |
| 17 | T4 | Int | Int | Int |
| 18 | T1 -> int | T1 -> int | Int -> int | Int -> int |
| 19 | T1 | T1 | Int | Int |

**Problem 4 (15 points).** Consider the following definition:

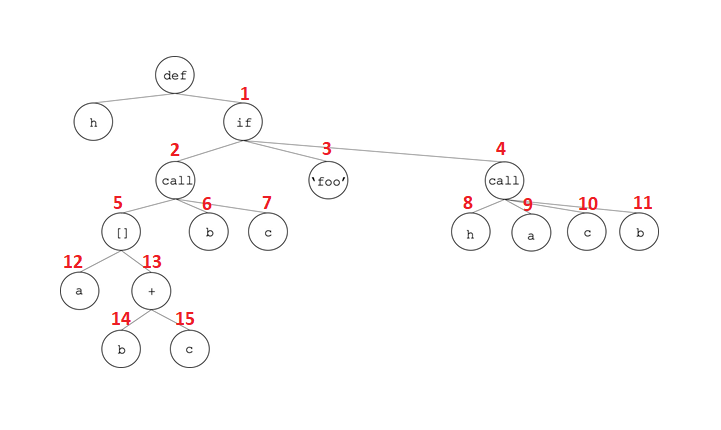
fun h(a, b, c) = if a[b + c](b, c) then

“foo”

else

h(a, c, b)

Using Hindley-Milner type inference, determine the type of h.



**Type of h is String per table below**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| h | (T1,T2,T3) -> T4 | (Array of (Int,Int) ->Bool,Int,Int) -> String |
| a | T1 | Array of (Int,Int) ->Bool |
| b | T2 | Int |
| c | T3 | Int |
| 1 |  | String |
| 2 | (T2,T3) ->Bool | (Int, Int) -> Bool |
| 3 | String | String |
| 4 | Infer String (because of 3) | String |
| 5 | (T2,T3)->Bool | (int,int)->Bool |
| 6 | T2 | Int |
| 7 | T3 | Int |
| 8 | (T1,T2,T3) -> T4 | (Array of (Int,Int) ->Bool,Int,Int) -> String |
| 9 | T1 | Array of (Int,Int) ->Bool |
| 10 | T2 | Int |
| 11 | T3 | Int |
| 12 | Array of (T2,T3)->Bool | Array of (Int,Int) ->Bool |
| 13 | Int | Int |
| 14 | Int | Int |
| 15 | Int | Int |