**CSE340 SPRING 2017**

**HOMEWORK 4: Type Systems and Hindley-Milner type checking**

Assigned 4/1/2017

Due 4/8/2017 by 11:59 pm

**Problem 1.** 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 : pointer to intger ; b : pointer to long; }

T4 : structure { b : pointer to long; a : pointer to 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 T8; }

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

T8 : structure { a : pointer to T5; 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;

**Determine which types are structurally equivalent. Show your work.**

(Note: the definition of structural equivalence that I gave in class is a permissive definition that few languages support and is different from the one in the textbook. You should use the definition I gave).

**Answer (work below):**

T5 and T6 are structurally equivalent.

T7 and T8 are structurally equivalent.

T9 and T10 are structurally equivalent.

**Work:**

**Note:** I did this problem before the in-class anouncement that we did not have to do the entire 14 × 14 table. I apologize, as I know it’s more work for you as a grader, but this problem took forever to do, and I am not going to redo it.

Initially:

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

**Note:** In order to save space, when more than one item in a row was false for the same reason (e.g. A1 is not equivalent to T1 through T8 because A1 is an integer, and T1 – T8 are all structures), I noted the subsequent types that were not equivalent in a sub-bullet point of the same level as the explanation for the first non-equivalence (see entry for “A1 ≡ T1?” for example).

Round 1:

* A1 ≡ A2?
  + False, integer ≢ pointer
* A1 ≡ A3?
  + False, integer ≢ pointer
* A1 ≡ T1?
  + False, integer ≢ structure
  + False for T2 – T8 for same reason
* A1 ≡ T9?
  + False, integer ≢ array
  + False for T10 for same reason
* A2 ≡ A3?
  + pointer ≡ pointer
    - False, pointer to float ≢ pointer to integer
  + False
* A2 ≡ T1?
  + False, pointer ≢ structure
  + False for T2 – T8 for same reason
* A2 ≡ T9?
  + False, integer ≢ array
  + False for T10 for same reason
* A3 ≡ T1?
  + False, pointer ≢ structure
  + False for T2 – T8 for same reason
* A3 ≡ T9?
  + False, integer ≢ array
  + False for T10 for same reason
* T1 ≡ T2?
  + False, different number of fields
  + False for T3 – T8 for same reason
* T1 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T2 ≡ T3?
  + Same number of fields
    - A1 ≡ pointer to integer?
      * False, integer ≢ pointer
    - False
  + False
* T2 ≡ T4?
  + Same number of fields
    - A1 ≡ pointer to long?
      * False, integer ≢ pointer
    - False
  + False
* T2 ≡ T5?
  + False, different number of fields
  + False for T6 – T8 for same reason
* T2 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T3 ≡ T4?
  + Same number of fields
    - pointer ≡ pointer
      * False, pointer to integer ≢ pointer to long
    - False
  + False
* T3 ≡ T5?
  + False, different number of fields
  + False for T6 – T8 for same reason
* T3 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T4 ≡ T5?
  + False, different number of fields
  + False for T6 – T8 for same reason
* T4 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T5 ≡ T6?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T7 ≡ pointer to T8
    - True
  + **True**
* T5 ≡ T7?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T7 ≡ pointer to T9
    - True
  + **True**
* T5 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T7 ≡ pointer to T10
    - True
  + **True**
* T5 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T6 ≡ T7?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T8 ≡ pointer to T9
    - True
  + **True**
* T6 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T8 ≡ pointer to T10
    - True
  + **True**
* T6 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T7 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T9 ≡ pointer to T10
    - True
  + **True**
* T7 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T8 ≡ T9?
  + False, structure ≢ array
  + False for T10 for same reason
* T9 ≡ T10?
  + array ≡ array
    - Same number of dimensions
      * rangeT9,1 ≡ rangeT10,1
      * rangeT9,2 ≡ rangeT10,2
        + True, T8 ≡ T7
      * True
    - True
  + **True**

After First Pass:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** | **T9** | **T10** |
| **A1** | **T** | F | F | F | F | F | F | F | F | F | F | F | F |
| **A2** | F | **T** | F | F | F | F | F | F | F | F | F | F | F |
| **A3** | F | F | **T** | F | F | F | F | F | F | F | F | F | F |
| **T1** | F | F | F | **T** | F | F | F | F | F | F | F | F | F |
| **T2** | F | F | F | F | **T** | F | F | F | F | F | F | F | F |
| **T3** | F | F | F | F | F | **T** | F | F | F | F | F | F | F |
| **T4** | F | F | F | F | F | F | **T** | F | F | F | F | F | F |
| **T5** | F | F | F | F | F | F | F | **T** | **T** | **T** | **T** | F | F |
| **T6** | F | F | F | F | F | F | F | **T** | **T** | **T** | **T** | F | F |
| **T7** | F | F | F | F | F | F | F | **T** | **T** | **T** | **T** | F | F |
| **T8** | F | F | F | F | F | F | F | **T** | **T** | **T** | **T** | F | F |
| **T9** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |
| **T10** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |

Things changed, so check true things again.

Round 2:

* T5 ≡ T6?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T7 ≡ pointer to T8
    - True
  + **True**
* T5 ≡ T7?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * False, pointer to T7 ≢ pointer to T9
    - False
  + False
* T5 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T6
    - pointer ≡ pointer
      * False, pointer to T7 ≢ pointer to T10
    - False
  + False
* T6 ≡ T7?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T5
    - pointer ≡ pointer
      * False, pointer to T8 ≢ pointer to T9
    - False
  + False
* T6 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * False, pointer to T8 ≢ pointer to T10
    - False
  + False
* T7 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T9 ≡ pointer to T10
    - True
  + **True**
* T9 ≡ T10?
  + array ≡ array
    - Same number of dimensions
      * rangeT9,1 ≡ rangeT10,1
      * rangeT9,2 ≡ rangeT10,2
        + True, T8 ≡ T7
      * True
    - True
  + **True**

After Second Pass:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** | **T9** | **T10** |
| **A1** | **T** | F | F | F | F | F | F | F | F | F | F | F | F |
| **A2** | F | **T** | F | F | F | F | F | F | F | F | F | F | F |
| **A3** | F | F | **T** | F | F | F | F | F | F | F | F | F | F |
| **T1** | F | F | F | **T** | F | F | F | F | F | F | F | F | F |
| **T2** | F | F | F | F | **T** | F | F | F | F | F | F | F | F |
| **T3** | F | F | F | F | F | **T** | F | F | F | F | F | F | F |
| **T4** | F | F | F | F | F | F | **T** | F | F | F | F | F | F |
| **T5** | F | F | F | F | F | F | F | **T** | **T** | F | F | F | F |
| **T6** | F | F | F | F | F | F | F | **T** | **T** | F | F | F | F |
| **T7** | F | F | F | F | F | F | F | F | F | **T** | **T** | F | F |
| **T8** | F | F | F | F | F | F | F | F | F | **T** | **T** | F | F |
| **T9** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |
| **T10** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |

Things changed, so check true things again.

Round 3:

* T5 ≡ T6?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T7 ≡ pointer to T8
    - True
  + **True**
* T7 ≡ T8?
  + Same number of fields
    - pointer ≡ pointer
      * True, pointer to T6 ≡ pointer to T5
    - pointer ≡ pointer
      * True, pointer to T5 ≡ pointer to T6
    - pointer ≡ pointer
      * True, pointer to T9 ≡ pointer to T10
    - True
  + **True**
* T9 ≡ T10?
  + array ≡ array
    - Same number of dimensions
      * rangeT9,1 ≡ rangeT10,1
      * rangeT9,2 ≡ rangeT10,2
        + True, T8 ≡ T7
      * True
    - True
  + **True**

After Third Pass:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** | **T9** | **T10** |
| **A1** | **T** | F | F | F | F | F | F | F | F | F | F | F | F |
| **A2** | F | **T** | F | F | F | F | F | F | F | F | F | F | F |
| **A3** | F | F | **T** | F | F | F | F | F | F | F | F | F | F |
| **T1** | F | F | F | **T** | F | F | F | F | F | F | F | F | F |
| **T2** | F | F | F | F | **T** | F | F | F | F | F | F | F | F |
| **T3** | F | F | F | F | F | **T** | F | F | F | F | F | F | F |
| **T4** | F | F | F | F | F | F | **T** | F | F | F | F | F | F |
| **T5** | F | F | F | F | F | F | F | **T** | **T** | F | F | F | F |
| **T6** | F | F | F | F | F | F | F | **T** | **T** | F | F | F | F |
| **T7** | F | F | F | F | F | F | F | F | F | **T** | **T** | F | F |
| **T8** | F | F | F | F | F | F | F | F | F | **T** | **T** | F | F |
| **T9** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |
| **T10** | F | F | F | F | F | F | F | F | F | F | F | **T** | **T** |

Nothing changed, so done.

**Problem 2**. Consider the following variable declarations in conjunction to the type declarations in problem 1

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)

Give your answers in the form of a table where each entry is a YES or a NO

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

**Explain your work.**

* s = t;
  + **Name:** Yes, both are of the same programmer-declared type (T9).
  + **Internal Name:** Yes, because they are name equivalent.
  + **Structural:** Yes, because T9 ≡ T9.
* t = u;
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** Yes, because T9 ≡ T10.
* u = v;
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** No, because rangeu1 ≢ rangev1.
* v = w;
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** No, because array ≢ structure.
* w = z;
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** Yes, because they are part of the same declaration.
  + **Structural:** Yes, because they are part of the same declaration, which means they are both structs, they both have the same number of fields, and each field in w’s struct is structurally equivalent to each field in z’s struct.
* z = x;
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** Yes, because they are both structs, they both have the same number of fields, and each field in z’s struct is structurally equivalent to each field in x’s struct.
* m = f(s)
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** No, because f(s) is a function and m is an int.
* n = f(u)
  + **Name:** No, they are not the same built-in or programmer-declared type.
  + **Internal Name:** No, because not name equivalent or part of same declaration.
  + **Structural:** No, because f(u) is a function and n is of type A1.

**Problem 3.** Consider the type declaration

A = struct {

a : int;

next : A;

}

1. Assume that for a struct T, the size of an object of type T is equal to the sum of the sizes of the fields of the structure. Write an equation that needs to be satisfied by objects of type A above assuming the size of int is 4.

**Answer:** sizeof(A) = 4 + sizeof(A.next)

1. Explain why if an object of type A exists, it must be infinite

**Answer:** The size of A depends on both the size of A.a and the size of A.next. The size of A.a is easily determined to be 4, because it is an int. A.next, however, is of type A, which means the size of A defined recursively.

As such,

sizeof(A.next) = 4 + sizeof(A.next.next)

Therefore,

sizeof(A) = 4 + sizeof(A.next) = 4 + 4 + sizeof(A.next.next)

This repeats itself over and over again, with the size of A depending on the size of A.next.next, A.next.next.next, and so on and so on.

1. Explain how it is possible to support type A in a programming language without having to allocate infinite memory for objects of type A (Hint: consider how an object of type A can be used and how the fields of A are accessed)

**Answer:** An object of type A could be supported in a programming language by requiring that structs that are defined recursively to make the recursive part of the definition be a pointer. That is, we would create a requirement that would force A.next to point to an object of type A, rather than actually *being* of type A.

Because A.next is a pointer, it stores an address. Since addresses are integers, that means A.next is an int. This means

sizeof(A.next) = 4

Which in turn means

sizeof(A) = 4 + sizeof(A.next) = 4 + 4 = 8.

Thus, the programming language only requires an allocation of 8 bytes for an object of type A.

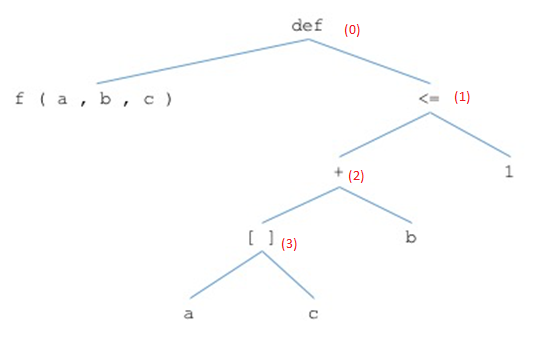
**Problem 4.** Consider the following definition

fun f(a, b, c) = (a[c] + b) <= 1

which has the following abstract syntax tree



Using Hindley-Milner type inference, determine the type of f. **Fully annotate the abstract syntax tree**



**Answer:**

(Note: Assuming types on either side of arithmetic operator or relational operator must be of same numeric type.)

Tf: bool (\*) (array of int, int, int)

**Work:**

fun f(a, b, c) = (a[c] + b) <= 1

|  |  |  |
| --- | --- | --- |
| **Visited** | **Node** | **Learned from visit:** |
| (0) | def-node | Tf = T(1) (\*) (Ta, Tb, Tc) |
| (1) | relop-node | T(1) = bool  T(2) = type of 1 = int  Tf = bool (\*) (Ta, Tb, Tc) |
| (2) | plus-node | T(2) = T(3) = Tb = int (because of visit to node (1))  Tf = bool (\*) (Ta, int, Tc) |
| (3) | array-node | Ta = array of T(3) = array of int (because of visit to node (2))  Tc = int  Tf = bool (\*) (array of int, int, int) |

Tf: bool (\*) (Ta, Tb, Tc)

Ta: array of int

Tb: int

Tc: int

T(0): Tf = T(1) (\*) (Ta, Tb, Tc)

T(1): bool

T(2): int

T(3): int

**Problem 5.** Consider the following definition

fun f(a, b, c, d, e, f) = if (a[c] + b[c]) <= f

d(a,e)

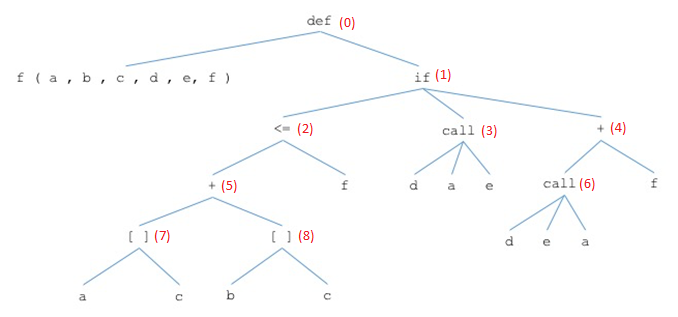
else

d(e,a) + f

which has the following abstract syntax tree



Using Hindley-Milner type inference, determine the type of f. **Fully annotate the abstract syntax tree**



**Answer:**

(Note: Assuming types on either side of arithmetic operator or relational operator must be of same numeric type, and that if must return the same type no matter what.)

Tf: bool (\*) (array of T, array of T, int, T (\*) (array of T, array of T), array of T, T)

**Work:**

fun f(a, b, c, d, e, f) = if (a[c] + b[c]) <= f

d(a,e)

else

d(e,a) + f

|  |  |  |
| --- | --- | --- |
| **Visited** | **Node** | **Learned from visit:** |
| (0) | def-node | TF = T(1) (\*) (Ta, Tb, Tc, Td, Te, Tg) |
| (7) | array-node | Ta = array of T(7)  Tc = int  Tf = T(1) (\*) (array of T(7), array of T(8), int, Td, Te, Tg) |
| (8) | array-node | Tb = array of T(8)  Tc = int (type match) |
| (5) | plus-node | T(5) = T(7) = T(8) = T (numeric)  Ta = array of T  Tb = array of T  Tf = T(1) (\*) (array of T, array of T, int, Td, Te, Tg) |
| (2) | relop-node | T(2) = bool  T(5) = Tg = T  Tf = T(1) (\*) (array of T, array of T, int, Td, Te, T) |
| (4) | plus-node | T(4) = T(6) = Tg = T  Tf = T(1) (\*) (array of T, array of T, int, Td, Te, T) |
| (6) | call-node | Td = T(6) (\*) (Te, Ta) = T (\*) (Te, array of T)  Tf = T(1) (\*) (array of T, array of T, int, T (\*) (Te, array of T), Te, T) |
| (3) | call-node | Te = Ta = array of T  (because visiting node (6) had them called in the opposite order)  Td = T(6) (\*) (Ta, Te) = T (\*) (array of T, array of T)  Tf = T(1) (\*) (array of T, array of T, int, T (\*) (array of T, array of T), array of T, T) |
| (1) | if-node | T(1) = bool  T(3) = T(4) = Tg = T(6) = T Tf = bool (\*) (array of T, array of T, int, T (\*) (array of T, array of T), array of T, T) |

Tf: T(1) (\*) (array of T, array of T, int, Td, Te, T)

Ta: array of T

Tb: array of T

Tc: int

Td: T (\*) (array of T, array of T)

Te: array of T

Tg: T

T(0): Tf = T(1) (\*) (Ta, Tb, Tc, Td, Te, Tg)

T(1): bool

T(2): bool

T(3): T

T(4): T

T(5): T (numeric)

T(6): T

T(7): T

T(8): T