Programming Language Design 2024
Types

Hans Hüttel

26 February 2024

1. *(20 minutes)* The Pascal language allows for variant records. Here is an example of this construct.

   ```
   type Three = 1..3;
   var worldrecord : record
   case selector : Three of
         1 : (a : integer);
         2 : (b : boolean);
         3 : (c : char)
   end
   ```

   Depending on the value of the field `selector`, a record will have either a field called `a`, `b` or `c`. We can read from and write to the fields of variant records using the familiar dot notation such as

   ```
   worldrecord.selector := 1;
   worldrecord.a := 123;
   ```

   Discuss the problems with static typing in the presence of variant records.

   There are also problems with dynamic typechecking for variant records. What are they?

2. *(20 minutes)* In part 3 of the podcast on types I claim that we can implement many other composite types as function types. Explain how to do this for tuples and records.

3. *(25 minutes)* In some languages we have range types that allow us to specify that a value must be within a certain interval of integers or characters. If we wanted to model the Danish grading scale, we might want to have variable declarations such as

   ```
   var dkgrade = -3...12
   ```

   If we wanted to model the ECTS grading scale, we might want to have variable declarations such as

   ```
   var ectsgrade = 'A'...'F'
   ```

   How easy is it to use static typing with types of this form? How about dynamic typing? Discuss the pros and cons; be as precise as possible.

4. *(30 minutes)* Functional programming languages support higher-order functions, but C-like languages do not.

   However, if we use C-style functions we can simulate this by using function pointers. Suppose we want to extend the syntax of C such that we can directly allow functions as parameters in C.

   Below is an example of a incomplete program with a function `applytoall` that applies a function to all elements of an array. We call `applytoall` with the `square` function and the array `myarray` as actual parameters and should afterwards have that every element in the array `myarray` has been squared.

   We would like the program to be well-typed, whereas we would like the modified program where `myarray` was a character array of type `char []` (and nothing else was changed) not to be well-typed.

   ```
   void applytoall(T f, *int arr)
   int n = sizeof(arr)
   for (int i = 0; i < n ; i++)
   ```

```
    {
    arr[i] = f (arr[i])
    }
...
int myarray[4] = {1,2,3,4}
...

T1 square(int x) ...

applytoall(square,myarray)
```

(a) Suggest new type constructs that we should add to the type system for C. What should `T` be in the above example?

(b) Suggest how we should type check function declarations and function calls if we extend the C language in this way.

The extended type system should be able to handle cases such as

```
T1 f(T2 g, T3 x) { return g(x); }
T4 h(T5 y) {...}
T6 j(T7 v) { return f(h,v));
```

and tell us what `T1,T2,T3,T4,T5,T6` and `T7` are.