Programming Language Design 2022
Types

Hans Hüttel

28 February 2022

Where nothing else is mentioned, chapters and exercises are from the notes "Programming Language Design and Implementation", which are available on the Absalon page for the course.

The video podcasts and the slides

You can either watch the podcasts via the media library or download everything. The podcast is yours to keep. All files, podcasts and slides alike, have the same name as the topic of the session. Remember that the podcasts come in several parts.

If you just to watch the podcasts online, you can find them in the Media section and watch them there. If you want to download the podcast and watch everything off-line, visit the Video folder in the Files area. The corresponding slides can be found in the Slides folder along with their LaTeX sources in the Slide sources' folder. You can cut and paste from these to your assignment reports.

Many of the exercises have no single correct answer, so you are expected to prepare for discussing these in the exercise classes.

Monday 28 February – *Types*

The podcast and slides will cover Chapter 8 up to and including Section 8.3 of "Programming Language Design and Implementation".

Learning goals for the session *Types*

The learning goals for this session are

- To understand why the notion of type exists and is useful
- To be able to distinguish between static and dynamic type checking and to understand the rationales behind these two approaches and their limitations
- To understand the various other ways of classifying type systems and apply them to distinguish between different type systems
- To understand commonly occurring type constructs
- To understand the notion of slack

How you should prepare before we meet on 28 February

Before we meet on 28 February, watch the podcast (all parts, please!) and read the text. You can do this in any order you like.

Also see if you can solve the following two small problems.

1. Explain the difference between an atomic type and a composite type as precisely as you can. One gets the impression that simple values always have atomic/simple types and composite values always have composite types. But is that really true?
2. Look up the general syntax of the `switch` expression in C#. What steps would you carry out in order to check if a `switch` statement is well-typed?

What happens on 28 February?

When we meet, we will discuss the small problems that I mentioned above along with a bunch of other small problems that I will make available to you on the day. The goal of this is to help you understand the text better and to prepare you for the exercise session on Tuesday 1 March.

I will introduce the small problems along the way and ask you to discuss them in small groups of two or three people. Every now and then I will discuss the problems with everyone.

Along the way, there will of course also be time for you to ask me any questions that you may have. I will do my best to answer them.

Tuesday 1 March – *Exercises*

For the exercise class, prepare the following exercises for presentation and discussion in class:

1. 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. 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. 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. 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.