# Type Systems

John Skaller

October 7, 2020

# Contents

# Chapter 1

# Subtyping Kernel

## 1.1 Set Inclusions

Let $U$ be some finite set and let $C$ be the power set of $U$, the set of all subsets of $U$. Then $C$ is a category with an arrow $A \to B$ if $A \subseteq B$. We can provide two maps $C \times C \to C$ called intersection and union with the usual binary operators $\cap$ and $\cup$ thereby constructing a lattice. These maps can be shown to be bifunctors.

An concretised version of this kernel abstraction replaces the subtyping judgment by set inclusion arrows with a witness function which provides an actual embedding.

## 1.2 Abstract Subtyping Kernel

An abstract subtyping kernel is the interface of an implementation. It has the same basic structure as the set inclusion lattice. However the concretised version, which uses actual functions, requires an strong constraint: these functions must be monic.

## 1.3 Generating Graph

We will specify the subtyping kernel as the category generated by a finite graph we present. In Felix notation we will write for example:

```
    type int = "int";
    type long = "long";
    type A= "A";
    type B= "B";
```

to specify the vertices of the graph as primitive monomorphic types. The code on the RHS is the representation in C++.

The edges of the graph are specified like:

```
    supertype long : int = "(long)$1";
    supertype B : A = "(B)$1";
```

which specifies the second type is a subtype of the first and provides an implementation called a coercion. The two categories generated by this graph consist of the subtyping judgement category and the more concrete coercion category which consists of all compositions of coercions.

It is important to note that no constraint is imposed on the set of coercions provided so that it is possible for the graph to be cyclic. In this case two primitives may be judged equivalent and can be interconverted using any path without loss of information: the objects are isomorphic and the coercion paths are isomorphisms.

There is a cruicial constraint on the representations and coercions functions: the functions must be monic. This causes a serious problem as we shall soon see but the requirement is essential to ensure coherence. We have already met one of the reasons: to be able to interconvert between any set of equivalent types the coercions must be isomorphisms, and this is ensured by the fact that all coercions are monic.

Saunders MacLean says an arrow $m : a \to b$ is monic in a category $C$ if when for any two parallel arrows $f, g : d \to a$, we have that $f \odot m = g \odot m$ implies $f = g$. Note that we are using reverse composition in OO style, so that above the first function is executed first, and the second applied to its result.

Intersection and union types are provided with binary operators like:

```
     a & b \| c
```

we note both these operators are symmetric and associative. The type 0 or `void` is an unit of the union and zero of the intersection, whilst the universal type is called `any` and is the unit of the intersection and zero of the union. An empty intersection is universal, whilst an empty union void.

In order to ensure coherence, no representation can be specified for intersections and unions, instead it is generated.

For the subtyping judgement category the judgements follow automatically from the set inclusion model: an intersection of two types is a subtype of each of them, any two types are subtype of their union.

However we need to be able to construct the coercions.