1 Type System

The type system is based on an extension of the ML type system with (multi-parameter) type classes and type constructors with monomorphic type kinds.

This is extended by adding a limited form of subtyping. In the new type system, the types are a result of the traditional typing together with added subtype dimensions and their subtype constraints.

Definition 1 (Typing dimension)**.** *The typing dimension is a semilattice with type unification serving as the meet operation.*

Definition 2 (Subtype dimension)**.** *The subtype dimension $\mathbb{S}$ is required to be a bounded lattice independent on the type-inferred program. We then define a transitive and reflective binary relation on types $\leq_S$ ($\tau \leq_S \sigma \iff \tau_S \wedge \sigma_S = \tau_S$ and $\tau_S \vee \sigma_S = \sigma_S$, where τ_S and σ_S are the types' respective subtype dimensions) which states that one type is less general in the given dimension than another type.*

We extend this definition to other comparison operators as well. Note that $\tau =_S \sigma \not\Rightarrow \tau = \sigma$, but indeed $\tau = \sigma \Rightarrow \tau =_S \sigma$.

We use the operator $\leq_S$ as a type constraint and we allow one of the operands be an element of $\mathbb{S}$.

In the type system specific to this thesis, we use two subtype dimensions: data kinds $\mathbb{K}$ and constnesses $\mathbb{C}$.

Observation 1 (Subtype dimension constraint combining)**.** *If a type τ is upper-bounded in a subtype dimension by multiple upperbounds: $\tau \leq_S s_1$, $\tau \leq_S s_2$, then we can replace such constraints with a single $\tau \leq_S (s_1 \wedge s_2)$. Similarly for lowerbounds and the join operation.*

Note that given multiple upperbounds and lowerbounds, the set of viable types can be empty.

Definition 3 (Types)**.** *Let $\mathbb{S}_1, \ldots \mathbb{S}_n$ be all subtype dimensions recognized by the system. Then for two types τ, σ with the same typing: $\tau = \sigma \iff \forall i \in \{1, \ldots n\}. \tau =_{S_n} \sigma$.*

We then formally define types as a composition of their dimensions: $\tau = \tau_T + \tau_{S_1} + \cdots \tau_{S_n}$, where τ_T is the typing of τ.

Definition 4 (Type variables)**.** *$\mathbb{V}$ is the space of type variables given as follows, each (sub)type considered by the inference algorithm will be assigned a type variable representing it:*
$$\begin{aligned} \mathbb{V} \Rightarrow\ & v \\ \mid\ & \mathbb{V}' \end{aligned}$$

The variables follow the obvious lexical ordering (based on their length), we use this ordering when choosing fresh variables.

Definition 5 (Typings)**.** *$\mathbb{T}$ is the space of typings given as follows, they specify that a type follows the same typing as a different type:*

$$
\begin{array}{lll}
\mathbb{T} \Rightarrow & & \text{Typing} \\
& \mathbb{V} & \text{Variable} \\
& [\mathbb{T}] \to \mathbb{T} & \text{Function} \\
& \mid [\mathbb{T}] & \text{Tuple} \\
& \mid \mathbb{T}\ \mathbb{T} & \text{Application} \\
& \ldots &
\end{array}
$$

Definition 6 (Types)**.** *Types use the same language as typings and thus the space $\mathbb{T}$ is considered the space of types as well as the space of typings.*