

Comp 411 Notes

System F_ω

Scribe: Roumen Kaiabachev

May 1, 2005

In today's lecture, we combine type operators with the polymorphism of System F to yield a system called F_ω .

The syntax and typing rules after combining System F and λ_ω are presented below. We add kinding annotations in places where type variables are bound (i.e. in type abstractions and later in quantifiers). For now, please ignore the premises in boxes.

$$t ::= X \mid \lambda X :: K.t \mid t t \mid t \rightarrow t$$

$$e ::= x \mid \lambda x : t.e \mid e e$$

$$\frac{\Gamma(x) = t \quad \Gamma \vdash e_1 : t_1 \rightarrow t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash x : t} \quad \frac{\Gamma, x : t \vdash e : t' \quad \boxed{\Gamma \vdash t : *}}{\Gamma \vdash \lambda x : t.e : t \rightarrow t'}$$

We also add a rule that allows to perform computation at level of types. This is bad as we have to guess t_1 or t_2 depending on which direction we are going:

$$\frac{\Gamma \vdash e : t_1 \quad t_1 \equiv t_2 \quad \boxed{\Gamma \vdash t_1 : *}}{\Gamma \vdash e : t_2}$$

The well-formedness rules for types are:

$$\frac{\Gamma(X) = K}{\Gamma \vdash X :: K} \quad \frac{\Gamma \vdash t_1 :: K_1 \rightarrow K_2 \quad \Gamma \vdash t_2 :: K_1}{\Gamma \vdash t_1 t_2 :: K_2} \quad \frac{\Gamma, X :: K \vdash t : K'}{\Gamma \vdash \lambda X :: K.t :: K \rightarrow K'}$$

λ_ω allows us to introduce type variables into our types (we can have $list(\alpha)$ without saying what the α is). System F allows us to introduce a new type variable to use in rest of terms (we can write things like *double* and have polymorphism). We want to put these together to get system F_ω . So what do we need to do at the type and kind levels? At the type level we add:

$$t ::= \dots \mid \forall X :: * t$$

The \forall gives you the ability to take a polymorphic function and get a polymorphic function (recall the *double* function that was applying itself twice). At the level of kinds we now have:

$$\mathsf{K} ::= * \mid \mathsf{K} \rightarrow \mathsf{K}$$

The $*$ says that we don't want to quantify over all types, but only over proper ones. We also add to the typing rules the premises in the boxes above. The result is that now, in our typing rules for expressions, lambda only qualifies over proper types. The same goes for the computation rule.

We also need an introduction rule for \forall . At the term level we add the following rule:

$$\frac{\Gamma, X :: * \vdash e : t}{\Gamma \vdash \lambda X :: *. e : \forall X :: * t}$$