

Bounded Quantified Types

Types and Programming Languages, Spring 2005, SoE UCSC

PRESENTER Jessica Gronski
SCRIBE Avik Chaudhuri

May 19, 2005

1 Looking back

We recalled the differences in programming styles when using objects versus ADTs, in particular the expressiveness problem with objects under pure existential types when defining (strong) binary operations. Such questions came up in Kim Bruce's talk on May 17 as well. We agreed that more information on the type variable in existential types is useful in order to write interesting programs; bounded existentials provide the means to pass such information.

2 On the board

The syntax of the pure lambda calculus with bounded quantified types has the following syntax.

Terms t	::=	x	(variable)
		$\lambda x : T.t$	(abstraction)
		$t t$	(application)
		$\Lambda X <: T.t$	(type abstraction)
		$t [T]$	(type application)
		$\{\star T, t\}$ as T	(packing)
		let $\{X, x\} = t$ in t	(unpacking)

Types T	::=	X	(type variable)
		Top	(maximum type)
		$T \rightarrow T$	(function type)
		$\forall X <: T. T$	(universal type)
		$\{\exists X <: T, T\}$	(existential type)

Environments Γ	::=	\emptyset	(empty)
		$\Gamma, x : T$	(variable type declaration)
		$\Gamma, X <: T$	(type bound declaration)

For a moment, the syntax looked funny because the bounding symbol $<$: occurred in three places (type abstraction, universal and existential type) instead of four (packing?). However it was observed that in the term $\{\star T_1, t\}$ as T_2 , T_2 would be of the form $\{\exists X <: T_3, T_4\}$, hence satisfying our intuitive alarm.

Before going further, we tried our hands at subtyping pure (unbounded) quantified types.

$$(<: \exists) \frac{\Gamma, X \vdash T <: S}{\Gamma \vdash \exists X. T <: \exists X. S} \quad (<: \forall) \frac{\Gamma, X \vdash T <: S}{\Gamma \vdash \forall X. T <: \forall X. S}$$

It was agreed that there was no need for rules with different bound variables, since these could be renamed without affecting the meaning of the quantified type. We then moved to writing some of the interesting type rules.

$$\begin{array}{l} (\text{Ax } <:) \frac{X <: T \in \Gamma}{\Gamma \vdash X <: T} \\ \\ (\exists <:^\circ) \frac{\Gamma, X <: T \vdash S_1 <: S_2}{\Gamma \vdash \{\exists X <: T, S_1\} <: \{\exists X <: T, S_2\}} \quad (\forall <:^\circ) \frac{\Gamma, X <: T \vdash S_1 <: S_2}{\Gamma \vdash \forall X <: T. S_1 <: \forall X <: T. S_2} \\ \\ (\exists <:) \frac{\Gamma \vdash T_1 <: T_2 \quad \Gamma, X <: T_1 \vdash S_1 <: S_2}{\Gamma \vdash \{\exists X <: T_1, S_1\} <: \{\exists X <: T_2, S_2\}} \quad (\forall <:) \frac{\Gamma \vdash T_2 <: T_1 \quad \Gamma, X <: T_2 \vdash S_1 <: S_2}{\Gamma \vdash \forall X <: T_1. S_1 <: \forall X <: T_2. S_2} \\ \\ (\forall E) \frac{\Gamma, X <: T \vdash t_2 : T_2}{\Gamma \vdash \Lambda X <: T. t_2 : \forall X <: T. T_2} \quad (\forall I) \frac{\Gamma \vdash t : \forall X <: T'. T'' \quad \Gamma \vdash T <: T'}{\Gamma \vdash t [T] : [X \mapsto T] T''} \end{array}$$

The first rule is straightforward. The second and third rules assume that the quantified type variables have the same bounds. The fourth and fifth rules are generalizations of these, where the bounds may be different. In such rules, the usual notions of covariance and contravariance apply. The sixth and seventh rules introduce and eliminate type abstractions. Similar rules can be written to introduce and eliminate packages.

Exercise Try encoding existentials with bounded universal types alone (*i.e.*, find encodings for existential types, packing and unpacking in terms of the rest of the calculus).