lec14: Polymorphism

Jana Dunfield

April 12, 2022

1 Introduction

Polymorphic code is code that can operate on different types of data. For example, in many languages, a function that concatenates lists can work on lists of any element type: concatenating two lists of integers into one list of integers, two lists of booleans into one list of booleans, and so on.

Polymorphic data is data that may take many forms. Most typed languages that include lists allow lists of any element type: lists of integers, lists of pairs of integers, lists of trees, and so on.

("Genericity" or "generics" are less fancy-sounding names for polymorphism.)

In our language, the expression

$$(\texttt{Lam}\ x\ x)$$

can be given the types (1) $\mathsf{int} \to \mathsf{int}$, (2) $\mathsf{bool} \to \mathsf{bool}$, (3) $\mathsf{int} + \mathsf{bool} \to \mathsf{int} + \mathsf{bool}$, and so on. That is, for any type α, it has type $\alpha \to \alpha$.

To add polymorphism to our language in this course, we will extend the grammar of types with $\forall\alpha.\,T$, read "for all types α, T". We will also extend it with *type variables* α, because we need to be able to mention α in T.

Types	$S, T ::=$	unit	unit type
		$\mid \mathsf{int}$	type of integers
		$\mid \mathsf{bool}$	type of booleans
		$\mid S \to T$	type of functions on S that produce T
		$\mid S \times T$	type of pairs of one S and one T
		$\mid S + T$	*disjoint union* or *sum* type: contains either an S or a T
		$\mid \forall\alpha.\,T$	polymorphic type
Typing contexts	$\Gamma ::=$	$\emptyset$	empty context
		$\mid \Gamma, x : S$	x has type S
		$\mid \Gamma, \alpha\ \mathsf{type}$	α is a type variable

Then the expression $(\texttt{Lam}\ x\ x)$ should have type

$$\forall\alpha.\,\alpha \to \alpha$$

To get the typing rules for this new type, we can "translate" the rules of sequent calculus.

2 From universal quantification in sequent calculus to typing rules

In our definition of sequent calculus, these were the rules for $\forall$:

$$\frac{\Gamma, x\big[a : \mathsf{nat}\big] \vdash B\ \mathsf{true}}{\Gamma \vdash (\forall a : \mathsf{nat}.\ B)\ \mathsf{true}}\,\forall\text{Intro} \qquad \frac{\Gamma \vdash (\forall a : \mathsf{nat}.\ B)\ \mathsf{true} \qquad \Gamma \vdash n : \mathsf{nat}}{\Gamma \vdash [n/a]B\ \mathsf{true}}\,\forall\text{Elim}$$

That $\forall$ quantified over natural numbers: "for all natural numbers a, ...". We will change this to quantify over types. Changing B to T, a to α, n to S and $x[a : \mathsf{nat}]$ to α type, we get rules that almost have the right form:

$$\frac{\Gamma, \alpha\ \mathsf{type} \vdash \qquad : T}{\Gamma \vdash \qquad : (\forall\alpha.\, T)}\,\forall\text{Intro} \qquad \frac{\Gamma \vdash \qquad : (\forall\alpha.\, T) \qquad \Gamma \vdash S\ \mathsf{type}}{\Gamma \vdash \qquad : [S/\alpha]T}\,\forall\text{Elim}$$

These don't have the right form because they're missing the expression. Here we have a design choice: we could add new expression forms, say `(All` e`)` and `(Instantiate S` e`)`. Or we could not: when called with an integer as an argument, the expression `(Lam x x)` returns an integer; when called with a boolean as an argument, the expression `(Lam x x)` returns a boolean; the type $\forall\alpha.\, \alpha \to \alpha$ accurately describes the behaviour of the expression, so why not let `(Lam x x)` have that type? We will follow this latter choice, which means that the blank spaces in the rules can all be replaced with e:

$$\frac{\Gamma, \alpha\ \mathsf{type} \vdash e : T}{\Gamma \vdash e : (\forall\alpha.\, T)}\,\forall\text{Intro} \qquad \frac{\Gamma \vdash e : (\forall\alpha.\, T) \qquad \Gamma \vdash S\ \mathsf{type}}{\Gamma \vdash e : [S/\alpha]T}\,\forall\text{Elim}$$

Let's see if these rules (1) let us give the expected type to `(Lam x x)`, and (2) let us pass arguments of different types to `(Lam x x)`.

$$\cfrac{\cfrac{\cfrac{(x : \alpha) \in \emptyset, \alpha\ \mathsf{type}, x : \alpha}{\emptyset, \alpha\ \mathsf{type}, x : \alpha \vdash x : \alpha}\,\text{type-assum}}{\emptyset, \alpha\ \mathsf{type} \vdash \mathtt{(Lam\ x\ x)} : \alpha \to \alpha}\,{\to}\text{Intro}}{\emptyset \vdash \mathtt{(Lam\ x\ x)} : (\forall\alpha.\, \alpha \to \alpha)}\,\forall\text{Intro}$$

$$\cfrac{\cfrac{\emptyset \vdash \mathtt{(Lam\ x\ x)} : (\forall\alpha.\, \alpha \to \alpha) \qquad \emptyset \vdash \mathsf{int}\ \mathsf{type}}{\begin{array}{c}\emptyset \vdash \mathtt{(Lam\ x\ x)} : [\mathsf{int}/\alpha](\alpha \to \alpha)\\ = \mathsf{int} \to \mathsf{int}\end{array}}\,\forall\text{Elim} \qquad \cfrac{}{\emptyset \vdash 3 : \mathsf{int}}\,\mathsf{intIntro}}{\emptyset \vdash \mathtt{(Call\ (Lam\ x\ x)\ 3)} : \mathsf{int}}\,{\to}\text{Elim}$$

And now with a boolean:

$$\cfrac{\cfrac{\emptyset \vdash \mathtt{(Lam\ x\ x)} : (\forall\alpha.\, \alpha \to \alpha) \qquad \emptyset \vdash \mathsf{bool}\ \mathsf{type}}{\begin{array}{c}\emptyset \vdash \mathtt{(Lam\ x\ x)} : [\mathsf{bool}/\alpha](\alpha \to \alpha)\\ = \mathsf{bool} \to \mathsf{bool}\end{array}}\,\forall\text{Elim} \qquad \cfrac{}{\emptyset \vdash \mathsf{False} : \mathsf{bool}}\,\text{type-false}}{\emptyset \vdash \mathtt{(Call\ (Lam\ x\ x)\ False)} : \mathsf{bool}}\,{\to}\text{Elim}$$

This leaves open the question of defining the new judgment form

$$\Gamma \vdash S\ \mathsf{type}$$

which I may do during lecture.