RECURSION

Alex Kavvos

Reading: PFPL, §19

1 Termination for the simply-typed λ -calculus

The simply-typed λ -calculus (STLC) has a property that is very unusual for a programming language.

Theorem 1 (Termination). For every $\vdash e : \tau$ there exists a v val such that $e \mapsto^* v$.

This may be proven using the technique of logical relations; see e.g. here.

In other words, every program written in the STLC terminates with a value. However, we intuitively know that any realistic programming language allows **infinite loops**. This theorem says that it is impossible to write a term with infinite behaviour in the STLC, so there is room to increase its expressivity.

2 Recursion and fixed points

We want to add **general recursion** to the STLC; this will enable the writing of recursive programs, as in Haskell. Consider the following recursive definition of the factorial function:

$$fact(n) = if n = 0 then 1 else n * fact(n - 1)$$

First we use (informal) λ -notation to abstract away the argument:

$$fact = \lambda n$$
. if $n = 0$ then 1 else $n * fact(n - 1)$

Then we use λ -notation again to abstract away the **recursive call**:

$$\mathsf{fact} = \underbrace{(\lambda f.\ \lambda n.\ \mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * f(n-1))}_F(\mathsf{fact})$$

This is an equation of the form fact = F(fact), which is to say that fact is a **fixed point** of the higher-order function given by $F(f) \stackrel{\text{def}}{=} \lambda n$. if n = 0 then 1 else n * f(n - 1). The types here are

fact :
$$\mathbb{N} \to \mathbb{N}$$
 $F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$

Therefore one way to add recursion to a programming language is to include a construct that computes the fixed point of any function $F: \sigma \to \sigma$. If we have fixed points at all types then we have them for $\mathbb{N} \to \mathbb{N}$ as well.

Curiously, this may be achieved within Haskell itself.

```
fix :: (a → a) → a
fix f = f (fix f)

h :: (Integer → Integer) → (Integer → Integer)
h f n = if n = 0 then 1 else n * f (n-1)

fact :: Integer → Integer
fact = fix h
```

3 PCF

PCF (= Programming Computable Functions) = (some version of) the STLC + fixed points. Syntax chart:

The **statics** of PCF are given by the following typing rules.

What has been removed: products, sums (can be added back at will). What has been replaced: numbers and strings (by natural numbers, with an "if zero" test). What has been added: fixed points. The **dynamics** are

$$\begin{array}{c} \text{Val-Zero} & \begin{array}{c} \text{Val-Succ} \\ \underline{e \ val} \\ \text{zero val} \end{array} & \begin{array}{c} \text{Val-Lam} \\ \underline{e \ \mapsto e'} \\ \text{succ}(e) \text{ val} \end{array} & \begin{array}{c} \text{D-Succ} \\ \underline{e \ \mapsto e'} \\ \text{succ}(e) \mapsto \text{succ}(e') \end{array} \\ \\ \begin{array}{c} \text{D-App-1} \\ \underline{e_1 \mapsto e_1'} \\ \underline{e_1(e_2) \mapsto e_1'(e_2)} \end{array} & \begin{array}{c} \text{D-Beta} \\ \overline{(\lambda x : \tau. e_1)(e_2) \mapsto e_1[e_2/x]} \end{array} \\ \\ \begin{array}{c} \text{D-Fix} \\ \overline{\text{fix}(x : \tau. e) \mapsto e[\text{fix}(x : \tau. e)/x]} \end{array} & \begin{array}{c} \text{D-Ifz-1} \\ \underline{e \mapsto e'} \\ \overline{\text{ifz}(e; e_0; x. e_1) \mapsto \text{ifz}(e'; e_0; x. e_1)} \end{array} \\ \\ \begin{array}{c} \text{D-Ifz-Succ} \\ \underline{\text{ord} \ \text{ifz}(\text{succ}(e); e_0; x. e_1) \mapsto e_1[e/x]} \end{array}$$

For example, the following terms are well-typed.

$$\vdash \mathsf{pred} \stackrel{\scriptscriptstyle\mathsf{def}}{=} \lambda n : \mathsf{Nat}.\,\mathsf{ifz}(n;\mathsf{zero};x.\,x) : \mathsf{Nat} \rightharpoonup \mathsf{Nat} \\ \vdash \mathsf{fix}(n:\mathsf{Nat}.\,\mathsf{succ}(n)) : \mathsf{Nat}$$

We have the following transition sequences.

```
\begin{array}{l} \operatorname{pred}(\operatorname{zero}) \longmapsto \operatorname{ifz}(\operatorname{zero};\operatorname{zero};x.\,x) \longmapsto \operatorname{zero} \\ \operatorname{pred}(\operatorname{succ}(\operatorname{zero})) \longmapsto \operatorname{ifz}(\operatorname{succ}(\operatorname{zero});\operatorname{zero};x.\,x) \longmapsto \operatorname{zero} \\ \operatorname{pred}(\operatorname{succ}(\operatorname{succ}(\operatorname{zero}))) \longmapsto \operatorname{ifz}(\operatorname{succ}(\operatorname{succ}(\operatorname{zero}));\operatorname{zero};x.\,x) \longmapsto \operatorname{succ}(\operatorname{zero}) \\ \operatorname{pred}(\operatorname{succ}(\operatorname{succ}(\operatorname{succ}(\operatorname{succ}(\operatorname{succ}(\operatorname{succ}(\operatorname{succ}(\operatorname{zero})));\operatorname{zero};x.\,x) \longmapsto \operatorname{succ}(\operatorname{succ}(\operatorname{zero})) \\ \vdots \end{array}
```