Comp 411
Principles of Programming Languages
Lecture 14
Eliminating Lambda Using Combinators

Corky Cartwright

February 14, 2022

How to Eliminate `lambda` (`map` in Jam)

Goal: devise a few combinators (functions expressed as λ-abstractions with no free variables) that enable us to express all **λ**-expressions without explicitly using **λ**.

Notation: let $\lambda^* x.M$ denote $\lambda x.M$ *converted* to an equivalent syntactic form that eliminates the starred **λ**. Then

$\lambda^* x.x \rightarrow I$ (where $I = \lambda x.x$)

$\lambda^* x.y \rightarrow Ky$ (where $K = \lambda y.\lambda x.y$)

$\lambda^* x.(M\ N) \rightarrow S(\lambda^* x.M)(\lambda^* x.N)$

(where $S = \lambda x.\lambda y.\lambda z.((x\ z)(y\ z))$)

How to Eliminate `lambda` (`map` in Jam) cont.

Question: Where did **S** come from?

- Intuition: it falls out when we formulate the translation to combinatory form using structural recursion on the abstract syntax of **λ**-expressions.
- The first two cases on the preceding slide do not involve recursion.
- In the third case, the form of the “magic” **S** combinator is determined by structural recursion! It is simply the pure **λ**-abstraction that works when plugged in for λ^*.

How Can We Systematically Eliminate All **λ**s?

Strategy:

- Eliminate **λ**-abstractions from inside out, one-at-a-time. This process terminates because it strictly reduces the sum of the *depth*s* of every **λ**-abstraction. The definition of *depth** is a bit tricky because each reduction rule (on slide 2) must strictly lower it for all **λ**-abstractions. The rule involving **S** must be handled delicately.
- Warning: this transformation can (and usually does) cause exponential blow-up because the third rule replaces one **λ**-abstraction by two of them. Note that the *depth** function grows exponentially with tree depth because the *depth** must add the depths of both subtrees of an application.

Final Observations

- Checking the **`App`** case

$$\mathtt{S\ (\lambda x.M)\ (\lambda x.N)}$$

$$= \mathtt{(\lambda x.\lambda y.\lambda z.(x\ z)(y\ z))\ (\lambda x.M)\ (\lambda x.N)}$$

$$= \mathtt{(\lambda y.\lambda z.((\lambda x.M)\ z)(y\ z))\ (\lambda x.N)}$$

$$= \mathtt{(\lambda z.((\lambda x.M)\ z)((\lambda x.N)\ z))}$$

$$= \mathtt{(\lambda z.(M_{x\leftarrow z})\ ((\lambda x.N)\ z))}$$

$$= \mathtt{(\lambda z.(M_{x\leftarrow z})\ (N_{x\leftarrow z}))} = \mathtt{\lambda x.(M\ N)}\ \text{(by } \alpha\text{-conversion)}$$

Note: the names $\mathtt{x\ y\ z}$ are fresh and arbitrary, distinct from any free names in $\mathtt{\lambda x.M\ \lambda x.N}$