Implementing Recursion via State

CS 350

Dr. Joseph Eremondi

Last updated: August 3, 2024

Recursion via State

Overview

- Goals

Overview

- Goals
  - To see how to implement an interpreter for a language with recursion

Overview

- Goals
  - To see how to implement an interpreter for a language with recursion
  - To see how recursion interacts with environments and stores

Overview

- Goals
  - To see how to implement an interpreter for a language with recursion
  - To see how recursion interacts with environments and stores
- Key Concepts

Overview

- Goals
  - To see how to implement an interpreter for a language with recursion
  - To see how recursion interacts with environments and stores
- Key Concepts
  - Landin's Knot

Recursion in Curly-Rec

- When we are allowed to refer to x while defining the value that is assigned to x

Recursion in Curly-Rec

- When we are allowed to refer to `x` while defining the value that is assigned to `x`
- We'll do this with a special `let` form

Recursion in Curly-Rec

- When we are allowed to refer to `x` while defining the value that is assigned to `x`
- We'll do this with a special `let` form
  - `{letrec x <expr> <expr>}`

Recursion in Curly-Rec

- When we are allowed to refer to `x` while defining the value that is assigned to `x`
- We'll do this with a special `let` form
  - `{letrec x <expr> <expr>}`
    - Gives `{letrec x e1 e2}` gives `x` the value `e1` then evaluates `e2` with `x` in scope

Recursion in Curly-Rec

- When we are allowed to refer to `x` while defining the value that is assigned to `x`
- We'll do this with a special `let` form
  - `{letrec x <expr> <expr>}`
    - Gives `{letrec x e1 e2}` gives `x` the value `e1` then evaluates `e2` with `x` in scope
    - Exactly like `letvar`, except **x is also in scope in e1**