Currying and the Lambda Calculus

CS 350

Dr. Joseph Eremondi

Last updated: July 19, 2024

Overview

Objectives

- To learn how multi-argument functions can be desugared into single-argument functions

Objectives

- To learn how multi-argument functions can be desugared into single-argument functions
    - Curly-Curry

Objectives

- To learn how multi-argument functions can be desugared into single-argument functions
  - Curly-Curry
- To see that *everything* can be desugared into single-argument functions

Objectives

- To learn how multi-argument functions can be desugared into single-argument functions
  - Curly-Curry
- To see that *everything* can be desugared into single-argument functions
  - by learning about the Lambda Calculus

Details

Executing a multiple-argument function

- Say we allow `{lambda {x y z} {+ x {* y z}}}` in Curly

Executing a multiple-argument function

- Say we allow `{lambda {x y z} {+ x {* y z}}}` in Curly
- How can we interpret a call to this function?

Executing a multiple-argument function

- Say we allow `{lambda {x y z} {+ x {* y z}}}` in Curly
- How can we interpret a call to this function?
    - Evaluate the body with either

Executing a multiple-argument function

- Say we allow `{lambda {x y z} {+ x {* y z}}}` in Curly
- How can we interpret a call to this function?
  - Evaluate the body with either
    - `x`, `y`, `z` replaced by concrete argument values (substitution)

Executing a multiple-argument function

- Say we allow `{lambda {x y z} {+ x {* y z}}}` in Curly
- How can we interpret a call to this function?
  - Evaluate the body with either
    - `x`, `y`, `z` replaced by concrete argument values (substitution)
    - `x`, `y`, `z` bound to concrete values in an environment

Achieving this: Currying

- We can achieve this with nested lambda expressions

Achieving this: Currying

- We can achieve this with nested lambda expressions

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

```
{{{f 1} 2} 3}
```

- Step by step:

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

```
{{{f 1} 2} 3}
```

- Step by step:
  - `f 1` produces `{fun {y} {fun {z} {+ 1 {* y z}}}}`

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

```
{{{f 1} 2} 3}
```

- Step by step:
  - `f 1` produces `{fun {y} {fun {z} {+ 1 {* y z}}}}`
  - Calling that on 2 produces `{fun {z} {+ 1 {* 2 z}}}`

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

```
{{{f 1} 2} 3}
```

- Step by step:
  - `f 1` produces `{fun {y} {fun {z} {+ 1 {* y z}}}}`
  - Calling that on 2 produces `{fun {z} {+ 1 {* 2 z}}}`
  - Calling that on 3 produces `{+ 1 {* 2 3}}`

Achieving this: Currying

- We can achieve this with nested lambda expressions

```
{let f {fun {x} {fun {y} {fun {z} {+ x {* y z}}}}}
  ....}
```

- To call, we do nested calls

```
{{{f 1} 2} 3}
```

- Step by step:
  - `f 1` produces `{fun {y} {fun {z} {+ 1 {* y z}}}}`
  - Calling that on 2 produces `{fun {z} {+ 1 {* 2 z}}}`
  - Calling that on 3 produces `{+ 1 {* 2 3}}`
    - Exactly what we want for `{f 1 2 3}`