Unlambda

Specialisation Seminar – 99 Bottles of Beer

Institute of Computer Science
University of Innsbruck

Author: Franziska Rapp
Supervisor: Cynthia Kop

Outline

- Motivation
- Introduction
- Combinators
- Examples
- Currying
- Similarities to other languages
- Conclusion

Motivation

- Functional, esoteric programming language
- First functional Turing tarpit
- *"Unlambda is meant as a demonstration of very pure functional programming rather than for practical use.“*

  Author unknown
- *"Debugging or reading Unlambda programs is just about impossible."*

  David Madore
- Invented by David Madore 1999

Introduction (1)

- Based on cl (= combinatory logic)

$$I\ x \rightarrow x$$

$$K\ x\ y \rightarrow x$$

$$S\ x\ y\ z \rightarrow x\ z\ (y\ z)$$

- Prefix notation
- Without parenthesis → apply operator ` instead
- Example: `fx applies f to the argument x

Introduction (2)

- No variables
- No data structures
- No code structures
- User-defined functions
  - ✓ Creation
  - x Saving
  - x Naming

Introduction (3)

- s, k and apply operator → Turing completeness!
- Eager evaluation → arguments before applying the function
- Obfuscated programming language
  - built to make programming difficult/challenging
- Implementation of lambda calculus without λ

  → Unlambda

Combinators (1)

- s
  - ```sxyz → ``xz`yz
  - Equal to: (((S x) y) z) → ((x z)(y z)) in combinatory logic

    Sxyz → xz(yz)
- k
  - ``kxy → x
- i
  - `ix → x

Combinators (2)

- v
  - `vx → v
- .x
  - `.xy → y
  - Prints the character x as side effect
- r
  - Abbreviation for .<newline>

Combinators (3)

- d
  - ``dxy → `xy
  - Special effect: the second argument must be evaluated before the first!
  - If d has only one argument → the argument will not be evaluated!
- c
  - Takes 1 argument
  - Creates a continuation (<cont>) of the program's current state
  - Applies the argument to this continuation

Combinators (4)

- <cont>
  - Takes 1 argument
  - When applied to an argument
    - "goes back in time”
    - To the point where <cont> was created
    - Returns the argument of <cont>
  - This function is the most difficult one

Example 1 (99 bottles of beer)

```
````s``s``si`kk``s``s`ks``s`k`s`k``s``si`k`ki`kkk`k``s``s`ksk`k.9``s``s``si`k`ki`kk`k`d`.9.1````s``s``si`k`ki`kk`k`d
`.9.0``s``s``si`kk``s``s`k`s``s`ks``s``s`ksk`k`s`k.9``s``s`ks``s`k`s`ks``s``s`ks``s`kk``s`ks``s`k`s`k``si`kkk`k``s``s`ks
``s`kk``s`ksk`k``s`k`s``s`kskk`k``s``s`ks``s``s`ksk`k`s`k.1``s``s`ksk`k`s`k.0``s``si`k`ki`kk``s``s``s`k`s``si`k``s`k.9.1
``s``s`ks``s``s`ks``s`k`s`k``si`kkk`k``s``s`ksk`k``s``s`ksk`k.1`k``s``si`k``s`k.1.1``si`k``s`k.0.1``s``si`k`ki`kk``s``s``s
``si`k`ki`kk`k.9``s``s``si`kk``s``s`ks``s`k`s`k``s``si`k`ki`kkk`ki``s`k``s`kr``s`k..``s``s``si`k`ki`kk``s``s``si`kk`k`ki
``s`k.s``s``s``si`kk`kk``s`k`d`.r`.o`.m`.`.o`.N.e``s``s``si`k`ki`k`ki``s``s``si`kk`k`ki``s``s``si`kk`kk``s`k.1``s`kr``s`k
.,``s``s``si`k`ki`kk``s``s``si`kk`k`ki``s``s``si`kk`kk``s`k.1``s`kr``s`kr``s`k..``s``s``si`k`ki`kk``s``s``si`kk`k`ki``s``s
``si`kk`kk``s`k.1`ki``s``si`k`ki`k`ki```s``s``si`kk``s``s``si`kk``si`k`ki``s`k``s``s`ks``s`k`s`k``s``si`k`ki`k`kik``s``s
`ks``s`k`s`k``s``si`kk`k`kik``s`k`s`k.s``s``s`ks``s`k`s`k``s``si`kk`kkk``s`k`si``s`k`s`kr``s`k`s`k.,``s``s`ks``s`k`s`k
``s``si`k`ki`kkk`s``s`ks``s`k`s`k``s``si`kk`k`kik``s`k`s`k.s``s``s`ks``s`k`s`k``s``si`kk`kkk``s`k`si``s`k`s`kr``s`k`s
`kr``s`k`s`k.,``s``s`ks``s`k`s`k``s``si`k`ki`kkk`s``s`ks``s`k`s`k``s``si`kk`k`kik``s`k`s`k.s``s``s`ks``s`k`s`k``s``si`kk
`kkk`k``si`ki``si`k`ki``s``s`k``s``s``si`k`ki`k`ki``s``s``si`kk`k`ki``s`k.s``s``s``si`kk`kk``s`k.9``s`k.9``s`kr``s`k.,``s
``s``si`k`ki`kk``s``s``si`kk`k`ki``s`k.s``s``s``si`kk`kk``s`k.9``s`k.9`ki``si`k`ki`k``s``si`k.8``s``si`k.7``s``si`k.6``s``si
`k.5``s``si`k.4``s``si`k.3``si`k.2```s``si``s``si``s``si`k`d`.,`.d`.n`.u`.o`.r`.a`. `.t`.i`.`.s`.s`.a`.p`.`.,`.n`.w`.o`.d`.`.e`.n`.o`.
`.e`.k`.a`.T`rr`k`d`.l`.a`.w`. `.e`.h`.t`. `.n`.o`..l``s``si`k`d`.e`.e`.b`. `.f`.o`. .r`k`d`.l`.t`.t`.o`.b`..ei``s`k`s``s`ks``s`k`sik
``s`kkki
```

- It takes 38411 steps to evaluate this program
```

Example 2

``ci`k	(c creates a <cont>, applying i to this <cont>)
→ ``i<cont>k	(i returns <cont> as normal)
→ `<cont>k	(applying <cont> takes us "back in time"...)
→ `kk	(... and changes the original `ci to k)
	(`kk will not be evaluated → k needs two args)

Example 3

``````````.H.e.l.l.o. .w.o.r.l.dri

Output:
``````````

Example 3

``````````.H.e.l.l.o. .w.o.r.l.dri

→ `````````.e.l.l.o. .w.o.r.l.dri

Output:

H
``````````

Example 3

```
``````````.H.e.l.l.o. .w.o.r.l.dri
→ ``````````.e.l.l.o. .w.o.r.l.dri
→ ``````````.l.l.o. .w.o.r.l.dri
```

Output:

He
```

Example 3

``````````.H.e.l.l.o. .w.o.r.l.dri

→ ``````````.e.l.l.o. .w.o.r.l.dri

→ ``````````.l.l.o. .w.o.r.l.dri

→ ``````````.l.o. .w.o.r.l.dri

Output:

Hel
``````````

Example 3

```
``````````.H.e.l.l.o. .w.o.r.l.dri
→ ``````````.e.l.l.o. .w.o.r.l.dri
→ ```````````.l.l.o. .w.o.r.l.dri
→ ````````````.l.o. .w.o.r.l.dri
→ …
→ `ri
```

Output:

Hello world
```

Example 3

```
``````````.H.e.l.l.o. .w.o.r.l.dri
→ ``````````.e.l.l.o. .w.o.r.l.dri
→ ```````````.l.l.o. .w.o.r.l.dri
→ ````````````.l.o. .w.o.r.l.dri
→ …
→ `ri
→ i
```

Output:

Hello world
```

Currying

- Some functions need more than one argument
- But we can only apply one arg with ` operator
- For example s
  - `sx is a function waiting for 2 args
  - ``sxy is a function waiting for 1 arg
  - ```sxyz can be evaluated

Similarities to other languages (1)

- Unlambda combines functional and obfuscated programming languages
- Other functional programming languages
  - Scheme (a Lisp dialect)
  - Ocaml
  - Haskell
- Other obfuscated programming languages
  - INTERCAL
  - Befunge

Similarities to other languages (2)

- Currying
  - lambda calculus
  - Haskell
- Continuation
  - Scheme
  - (Exceptions in Java)
- Dynamically manipulating source code at runtime
  - Befunge

Similarities to other languages (3)

- Very similar languages
  - Jot (Iota)
  - Lazy K
- Turing tarpits
- Based on the cl
- Esoteric

Conclusion

- Very interesting continuation feature
  - But very hard to program with it
  - Not sensible for "real languages“
- Disadvantage: No user interaction possible

  → Unlambda 2 realizes that with 4 additional funtions
- The goal of unreadable code has been achieved ;)
- Challenging to write programs with this limited number of functions
- But there are some small cool progs like: ``r`cd`.*`cd

Thank you for your attention!

Questions?