Web Programming

Woche 3

FS 2019

Prof. D. König

Rückschau

Fragen zum letzten Quiz

Fragen zum letzten Homework

Sonstige Fragen

Berichtigung

var: hoisted to function
initialisiert
const, let: hoisted to local scope
nicht initialisiert

{ } als anonymer block is möglich

Heutiges Programm

Test setup: Einzeldateien, Verzeichnisse

Lambda Boolean Logic

Lambda Algebraische Datentypen

Quiz: Punkte sammeln

Live Coding Log

https://github.com/
Dierk/WebProgramming/
tree/fs19-live-coding

Goal

Becoming creative with

- Higher Order Functions
- Using the Lambda scope

Atomic Lambda Terms

```
// atoms
const id    = x =>       x;
const konst = x => y => x;


// derived true, false, and, or, equals, …
const F = …;
const T = …;
```

Pair, Product Type

```
const pair = x => y => f => f(x)(y);
const fst  = p => p(T);
const snd  = p => p(F);
```

the basic product type

Triple

Can you encode triples by following the same pattern as for pairs?

N-Tuples?

Pair encoding

```
const person =
              firstname =>
              lastname  =>
              age       =>
              pair (pair(firstname)(lastname)) (age);

const firstn = p => fst(fst(p));
const lastn  = p => snd(fst(p));
const age    = p => snd(p);
```

Pair, Triple, etc.

Note that our pattern leads to immutable values ("objects")!

Accessor functions are lazy until they are applied (beta reduced).

Either, Co-Product, Sum

```
// dual of the product
const pair = x => y => f => f(x)(y);    // one ctor
const fst  = p => p(T);                 // accessor 1
const snd  = p => p(F);                 // accessor 2
```

Either, Co-Product, Sum

```
// dual of the product
const pair = x => y => f => f(x)(y);     // one ctor
const fst  = p => p(T);                  // accessor 1
const snd  = p => p(F);                  // accessor 2

const Left   = x => …;                     // ctor 1
const Right  = x => …;                     // ctor 2
const either = e => f => g => …;           // accessor
```

Either, Co-Product, Sum

```
// dual of the product
const pair = x => y => f => f(x)(y);    // one ctor
const fst  = p => p(T);                 // accessor 1
const snd  = p => p(F);                 // accessor 2

const Left   = x => …;                    // ctor 1
const Right  = x => …;                    // ctor 2
const either = e => f => g => e(f)(g);    // accessor
```

Either, Co-Product, Sum

```
// dual of the product
const pair = x => y => f => f(x)(y);    // one ctor
const fst  = p => p(T);                 // accessor 1
const snd  = p => p(F);                 // accessor 2

const Left   = x => f => g => f(x);       // ctor 1
const Right  = x => …;                    // ctor 2
const either = e => f => g => e(f)(g);    // accessor
```

Either, Co-Product, Sum

```
// dual of the product
const pair = x => y => f => f(x)(y);    // one ctor
const fst  = p => p(T);                 // accessor 1
const snd  = p => p(F);                 // accessor 2

const Left   = x => f => g => f(x);       // ctor 1
const Right  = x => f => g => g(x);       // ctor 2
const either = e => f => g => e(f)(g);    // accessor
```

Either, Co-Product, Sum

```
const Left   = x => f => g => f(x);        // ctor 1
const Right  = x => f => g => g(x);        // ctor 2
const either = e => f => g => e(f)(g);     // accessor
```

the basic sum type

Special Case: Maybe

```
const Nothing  = Left ();
const Just     = Right  ;
const maybe    = either ;

maybe (expressionThatMightGoWrong)
      (handleBad)
      (handleGood);
```

go around null / undefined

To Do at Home

Allow grouping of Test Cases (suite)
Report OK when whole suite passes

JavaScript Scope Chains and Closures:
https://www.youtube.com/watch?v=zRZNb4GDOPI (InfoQ, 56 min)