Advanced Programming

Final Lecture

Troels Henriksen

Real World Haskell

How to Live the Rest of Your Life

Course Evaluation

The Exam

Leaving the ivory tower

AP	▪ Simple models. ▪ Clear specifications. ▪ No performance concerns.
Real world	▪ Arbitrary complexity. ▪ Unclear requirements. ▪ Performance requirements.

The ivory tower still exists in the real world

- My research is on parallel programming, compiler optimisations, high performance computing, etc.
- **Applied computer science**—I read and write many programs.
  - Haskell is my *pragmatic* tool of choice.

The ivory tower still exists in the real world

- My research is on parallel programming, compiler optimisations, high performance computing, etc.
- **Applied computer science**—I read and write many programs.
  - Haskell is my *pragmatic* tool of choice.

- Main applied project is *Futhark*.
- Almost 100k lines of Haskell.
- Haskell is *not the point*—feels like any other program.

`https://futhark-lang.org`

Artwork by Robert Schenck.

So what is it like to use Haskell for real programs?

`String` considered harmful

```
type String = [Char]

"DIKU" == 'D' : 'I' : 'K' : 'U' : []
```

`String` considered harmful

```
type String = [Char]
```

```
"DIKU" == 'D' : 'I' : 'K' : 'U' : []
```

But consider the memory usage.

- Each `:` (*cons*) cell contains two slots: *value* and *successor*.
  - Each is a pointer to somewhere in memory.

`String` considered harmful

```
type String = [Char]
```

```
"DIKU" == 'D' : 'I' : 'K' : 'U' : []
```

But consider the memory usage.

- Each `:` (*cons*) cell contains two slots: *value* and *successor*.
  - Each is a pointer to somewhere in memory.
- A four-character string needs four cells, meaning eight pointers in total.
  - That is $8 \cdot 8 = 64$ bytes!
    - For a four-character string!
    - This is $16x$ overhead.
  - And in fact, each Haskell value also needs a *header* for garbage collection purposes, so things are even worse!

`String` considered harmful

```
type String = [Char]

"DIKU" == 'D' : 'I' : 'K' : 'U' : []
```

But consider the memory usage.

- Each `:` (*cons*) cell contains two slots: *value* and *successor*.
  - Each is a pointer to somewhere in memory.
- A four-character string needs four cells, meaning eight pointers in total.
  - That is $8 \cdot 8 = 64$ bytes!
    - For a four-character string!
    - This is 16*x* overhead.
  - And in fact, each Haskell value also needs a *header* for garbage collection purposes, so things are even worse!
- Also the characters are scattered all over memory $\Rightarrow$ terrible locality.

Real Haskell programs use `Data.Text.Text`

`https://hackage.haskell.org/package/text-2.1.1/docs/Data-Text.html`

- Built on top of `Data.ByteString`.

Haskell cheat code

```
unsafePerformIO ::  IO a -> a
```

- **Extremely Dangerous**, much more than you think.

Free Monads are Used In Real Programs!

But...

- The representation you saw in AP is inefficient.
  - Each use of »= has to completely rebuild the computation tree.
  - A better one looks like this:

```
newtype F f a = F (forall r.  (a -> r) -> (f r -> r) -> r)
```

Example of free monads in my own work

- Package resolution: `https://github.com/diku-dk/futhark/blob/master/src/Futhark/Pkg/Solve.hs`
- Interpreter with single stepping, breakpoints, and tracing: `https://github.com/diku-dk/futhark/blob/master/src/Language/Futhark/Interpreter.hs`

Alternatives to Free Monads

Free monads are not the only way used to decouple the specification of effects from their interpretation.

- Another technique (perhaps more common) is the so-called *tagless final*.
- Idea is to describe effects with a *type class*.

Example based on Haskell's `mtl`

```
class Monad m => MonadState s m | m -> s where
  get :: m s
  put :: s -> m ()

class Monad m => MonadReader r m | m -> r where
  ask :: m r
```

Example based on Haskell's `mtl`

```
class Monad m => MonadState s m | m -> s where
  get :: m s
  put :: s -> m ()

class Monad m => MonadReader r m | m -> r where
  ask :: m r
```

The classes can then be implemented by different concrete monads.

```
newtype RS r s a = RS (r -> s -> (a,s))

instance MonadState s (RS r s) where
  get = RS $ \r s -> (s,s)
  put s = RS $ \r _ -> ((),s)

instance MonadReader r (RS r s) where
  ask = RS $ \r s -> (r,s)
```

Real World Haskell

How to Live the Rest of Your Life

Course Evaluation

The Exam

If you liked AP

Unfortunately there is no *Really Advanced Programming* course at DIKU.

But there are courses that have the same spirit.

If you liked AP

Unfortunately there is no *Really Advanced Programming* course at DIKU.

But there are courses that have the same spirit.

- *Programming Massively Parallel Hardware* (PMPH), block 1, taught by Cosmin Oancea. GPU programming. Only course at DIKU where you are specifically graded on writing fast programs.

If you liked AP

Unfortunately there is no *Really Advanced Programming* course at DIKU.

But there are courses that have the same spirit.

- *Programming Massively Parallel Hardware* (PMPH), block 1, taught by Cosmin Oancea. GPU programming. Only course at DIKU where you are specifically graded on writing fast programs.
- *Data Parallel Programming* (DPP), block 2, taught by Cosmin Oancea and myself. High level functional programming and performance. Strongly based on local research.

If you liked AP

Unfortunately there is no *Really Advanced Programming* course at DIKU.

But there are courses that have the same spirit.

- *Programming Massively Parallel Hardware* (PMPH), block 1, taught by Cosmin Oancea. GPU programming. Only course at DIKU where you are specifically graded on writing fast programs.
- *Data Parallel Programming* (DPP), block 2, taught by Cosmin Oancea and myself. High level functional programming and performance. Strongly based on local research.
- *Semantics and Types* (SaT), block 3, taught by Andrzej Filinski. Programming language theory and semantics. **Theoretical.**

Also consider projects

I work in the *Programming Languages and Theory of Computation* section, and there are many researchers willing to supervise projects in functional programming and other PL topics.

- 30 ECTS master thesis.
- 7.5 ECTS thesis preparation project.
- 7.5 ECTS or 15 ECTS “project outside course scope”.

Potential supervisors:

- Troels Henriksen
- Martin Elsman
- Torben Mogensen
- Ken Friis Larsen
- Andrzej Filinski
- Cosmin Oancea
- The others: `https://di.ku.dk/english/research/pltc/`

Real World Haskell

How to Live the Rest of Your Life

Course Evaluation

The Exam

Preliminary context

- **The MSc at DIKU is supposed to be challenging.**
  - *Intended* to be the highest academic level of computer science in Denmark.

Preliminary context

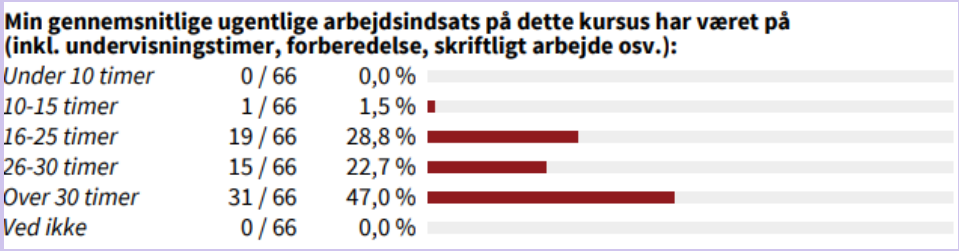
- **The MSc at DIKU is supposed to be challenging.**
  - *Intended* to be the highest academic level of computer science in Denmark.
- **AP is supposed to be hard.**
  - Most ambitious *mandatory* programming course in Denmark (that I could find).

Preliminary context

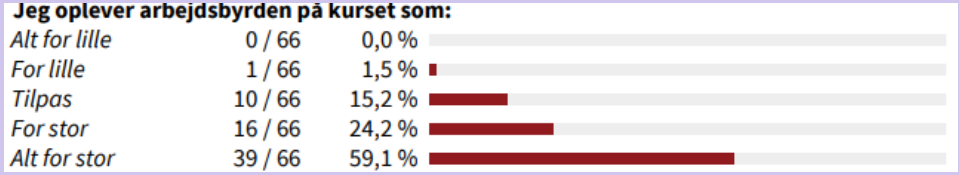
- **The MSc at DIKU is supposed to be challenging.**
  - *Intended* to be the highest academic level of computer science in Denmark.
- **AP is supposed to be hard.**
  - Most ambitious *mandatory* programming course in Denmark (that I could find).
- **Suffering is not the same as learning.**
  - Students don't learn more when they are stressed.
  - Teachers don't benefit from tormenting students.

AP has been notorious for excessive workload

2023 course evaluation — hours spent by students per week



2023 course evaluation — perceived student workload



What we tried to do

What we tried to do

1. *Streamline* the course to minimise how much time is spent on nonproductive activities.
   - E.g. less analysis of requirements.
   - Got rid of Erlang for concurrency.

What we tried to do

1. *Streamline* the course to minimise how much time is spent on nonproductive activities.
   - E.g. less analysis of requirements.
   - Got rid of Erlang for concurrency.
2. Strong *cohesion* between course elements.
   - The notes/lecture $\Rightarrow$ exercise $\Rightarrow$ assignment progression.

What we tried to do

1. *Streamline* the course to minimise how much time is spent on nonproductive activities.
   - E.g. less analysis of requirements.
   - Got rid of Erlang for concurrency.
2. Strong *cohesion* between course elements.
   - The notes/lecture $\Rightarrow$ exercise $\Rightarrow$ assignment progression.
3. Very applied lectures.
   - Inspired by my experiences in other courses.
   - Perhaps we took it too far this year.

What we tried to do

1. *Streamline* the course to minimise how much time is spent on nonproductive activities.
   - E.g. less analysis of requirements.
   - Got rid of Erlang for concurrency.
2. Strong *cohesion* between course elements.
   - The notes/lecture $\Rightarrow$ exercise $\Rightarrow$ assignment progression.
3. Very applied lectures.
   - Inspired by my experiences in other courses.
   - Perhaps we took it too far this year.
4. Made assignments a bit smaller.
   - But not as much as you'd think.

Overall, tried to ensure that weaker students had a good chance to pass by putting in a reasonable amount of effort (20-25 hours per week).

Notes I wrote to guide my work:
`https://sigkill.dk/writings/teaching.html`

Looking at graphs

Selected comments

- Some of these have been rephrased, paraphrased, or translated from Danish.
- I consider them all *constructive* and *useful*, even if I do not always agree.

Please, move one of the TA sessions after the lecture on Tuesday. Having both of them on the same day is a bad idea in my opinion.

Lectures might as well have been a video.

Lectures did not provide anything that was not in the course notes.

The choice to have live coding lectures is a good one overall.

I think the biggest downside of the course is the lectures.

Also the lectures were very inspiring, with good discussions.

The course can be very challenging for students that have no background in functional programming.

The level is too high as a mandatory course for a further development for people who have had DatØk

As a dat-øk, you don't have a chance on earth in relation to the prerequisites that are taken into account.

In my opinion, for international students, functional programming is a new thing as we have only done procedural programming in our bachelors.

I don't understand why elimination of left-recursion in CFGs is part of the syllabus. How is it justified as part of advanced programming? In addition, several have had 1-2 courses where this has already been the syllabus.

- It is emphasized that the principles of the course are emphasized and not the language (Haskell), which is good. However, I miss some perspectives on how to use the principles (or which of them can be used), in languages that do not support e.g. monads. What you learn feels a bit like all or nothing principles - either you choose to write your program in Haskell (or similar) or it can all be the same.

> *Also, I noticed that Haskell is not used much in the job market, maybe we can learn something more practical.*

I will answer with a quote:

> *Also, I noticed that Haskell is not used much in the job market, maybe we can learn something more practical.*

I will answer with a quote:

> *It is not the task of the University to offer what society asks for, but to give what society needs.*
>
> — Edsger Dijkstra, EWD1305 (2000)

Real World Haskell

How to Live the Rest of Your Life

Course Evaluation

The Exam

The exam is about extending APL with new constructs

- Starting point is a slightly simplified solution to the week 4 exercises.
- Several subtasks, some of them simple, some tricky, many of them independent.
  - Incorporate most parts of the curriculum.
  - You'll have to do parsing, evaluation, monadic interpretation of effects in monads as needed.
- Goals:
  - Weaker students should have a chance to pass.
  - Strong students should be able to get a top grade.
  - No stress-related breakdowns.

The exam is about extending APL with new constructs

- Starting point is a slightly simplified solution to the week 4 exercises.
- Several subtasks, some of them simple, some tricky, many of them independent.
  - Incorporate most parts of the curriculum.
  - You'll have to do parsing, evaluation, monadic interpretation of effects in monads as needed.
- Goals:
  - Weaker students should have a chance to pass.
  - Strong students should be able to get a top grade.
  - No stress-related breakdowns.

New language constructs

- Tuples
- Sequential loops.
- Concurrent execution.
- Multi-threaded key/value state.

Tuples

```
(a, b, c)
```

- Pretty much like in Haskell.
- Any number of elements except 1.
- Straightforward evaluation.
- *e*`.`*i* fetches element *i* from a tuple (0-indexed).

Sequential loops

```
loop p = init for i < bound do body
```

Intuitively equivalent to a function f defined as

```
f(i, p) = if i < bound do f (i+1) body else p
```

and invoked as $f(0, init)$.

In words

Repeat *body bound* times, where *p* is initially *init* and then bound to the result of executing the previous loop iteration, returning the final value of *p*.

Sequential loops

```
loop p = init for i < bound do body
```

Intuitively equivalent to a function f defined as

```
f(i, p) = if i < bound do f (i+1) body else p
```

and invoked as $f(0, init)$.

In words

Repeat *body bound* times, where *p* is initially *init* and then bound to the result of executing the previous loop iteration, returning the final value of *p*.

Also `while`-loops:

```
loop p = init while cond do body
```

Concurrent execution

`a && b`:

- Concurrently execute `a` and `b`, returning pair of results.
- `a` and `b` evaluated concurrently.
  - Or not, depending on interpretation function.

`a || b`:

- Concurrently execute `a` and `b`.
- Evaluates to result of `a` *or* `b`.
- “First one to finish”, depending on interpretation.
- May terminate even if `a` or `b` is an infinite loop.
  - But not when both are.
  - Depends on interpretation.

Concurrent key/value state

- In the A4 APL, `get x` failed when `x` was not in the key-value store.
- In the concurrency-enabled APL, `get x` blocks until *someone else* (hopefully) writes `x`.

Example

`(get 0 && put 0 42).0` evaluates to `42`.

Concurrent key/value state

- In the A4 APL, `get x` failed when `x` was not in the key-value store.
- In the concurrency-enabled APL, `get x` blocks until *someone else* (hopefully) writes `x`.

Example

`(get 0 && put 0 42).0` evaluates to `42`.

Nondeterminism

`(get 0 && (put 0 42 && put 0 1337)).0` evaluates to either `42` or `1337`.

Concurrent key/value state

- In the A4 APL, `get x` failed when `x` was not in the key-value store.
- In the concurrency-enabled APL, `get x` blocks until *someone else* (hopefully) writes `x`.

Example

`(get 0 && put 0 42).0` evaluates to `42`.

Nondeterminism

`(get 0 && (put 0 42 && put 0 1337)).0` evaluates to either `42` or `1337`.

Not exactly message-passing, but vaguely inspired by *tuple spaces* (you don't need to know what this is).

Interpretation functions

Design is much like A4 with an evaluator that uses a free monad with some additional effects. You will implement three interpretation functions.

- **Pure:** a simple sequential interpretation with no concurrency, where e.g. `a || b` $\rightsquigarrow$ *a*.

Interpretation functions

Design is much like A4 with an evaluator that uses a free monad with some additional effects. You will implement three interpretation functions.

- **Pure:** a simple sequential interpretation with no concurrency, where e.g. `a || b` $\rightsquigarrow a$.
- **Simulated concurrent:** where we do a kind of "breadth-first" evaluation of the free monad.
- **Concurrent:** using (slightly simplified) SPC to implement physically concurrent execution.

More exam advice

Most important rule

Do not share your solution.

- This includes putting it in public Git repositories.
- This is a disciplinary infraction and you will be suspended from the university.

More exam advice

Most important rule

Do not share your solution.

- This includes putting it in public Git repositories.
- This is a disciplinary infraction and you will be suspended from the university.

- Page limit is long, this does not mean you are expected to fill it. **Short is good.**
- Don’t waste yours and our time with generic ChatGPT-answers to questions — you will receive no credit.
- Take breaks.
- You can pass without solving every task.
- The above information is non-normative.

See you next year

Questions?