101haskell

Ralf Lämmel, Thomas Schmorleiz, Andrei Varanovich
Software Languages Team, Uni Koblenz-Landau
What's 101haskell?

It is the Haskell-specific sub-chrestomathy of the advanced software chrestomathy 101companies.

http://softlang.uni-koblenz.de/chrestomathy/
What’s 101haskell?

- A collection of Haskell **programs**.
- An underlying **feature** model.
- Wiki-based documentation.
- Integrated **textbooks**.
- Lecture **scripts**.
- ...
What’s 101haskell?

- A collection of Haskell programs.
- An underlying feature model.
- Wiki-based documentation.
- Integrated textbooks.
- Lecture scripts.
- Demos
- ...

© 2013 101companies -- Presentation at IFL 2013
What's 101haskell?

- A collection of Haskell programs.
- An underlying feature model.
- Wiki-based documentation.
- Integrated textbooks.
- Lecture scripts.
- ...

These are actually tiny software systems. They are also called contributions.
Demo of a contribution (screencast)
What’s 101haskell?

- A collection of Haskell programs.
- An underlying feature model.
- Wiki-based documentation.
- Integrated textbooks.
- Lecture scripts.
- ...
Features of 101(haskell)

- **Company**: Companies, department, employees
- **Total**: Total the salaries of employees
- **Median**: Compute the median of the salaries
- **Cut**: Cut the salaries of employees in half
- **Depth**: Compute nesting depth of departments
- **COI**: Conflicts of interests for employees
- **Mentoring**: Associate mentors and mentees
- **Ranking**: Enforce salary to correlate with ranks
- **Singleton**: Constrain for a single company
- **History**: Maintain and analyze company history
- **Serialization**: De-/serialize companies
- **Persistence**: Persist companies
Mentoring: Associate mentors and mentees
Ranking: Enforce salary to correlate with ranks
Singleton: Constrain for a single company
History: Maintain and analyze company history
Serialization: De-/serialize companies
Persistence: Persist companies
Mapping: Map companies across technological space
Distribution: Distribute companies
Parallelism: Total or cut in parallel
Logging: Log company changes
Browsing: Browse companies interactively
Editing: Edit companies interactively
Restructuring: Restructure companies interactively
Web UI: Operate on companies in a web browser
Parsing: Parse companies in concrete syntax
Unparsing: Pretty print companies
What’s 101haskell?

- A collection of Haskell programs.
- An underlying feature model.
- Wiki-based documentation.
- Integrated textbooks.
- Lecture scripts.
- ...

For instance, software concepts used by contributions are documented and linked.
Demo of a concept (screencast)
What’s 101haskell?

- A collection of Haskell programs.
- An underlying feature model.
- Wiki-based documentation.
- Integrated textbooks.
- Lecture scripts.

We have just completed an introductory lecture on FP (Haskell) based on 101haskell.
Outline of the FP course

- Lecture First steps
- Lecture Basic software engineering
- Lecture Searching and sorting
- Lecture Basic data modeling
- Lecture Higher-order functions
- Lecture Type-class polymorphism
- Lecture Functors and friends
- Lecture Monads
- Lecture Parsing and unparsing
- Dry run for final
- Lecture Generic functions
- Lecture Functional data structures
- Final
Demo of a script (screencast)

Course: Lambdas in Koblenz

Headline

Introduction to functional programming at the University of Koblenz-Landau

Events
Why 101haskell?

- Showcase principles of an advanced chrestomathy.
- Leverage an advanced chrestomathy for teaching.
Research questions
(Chrestomathy perspective)

- How to integrate resources (e.g., vocabularies) that should feed into a knowledge base?
- How to manage knowledge in terms of organization, quality, visualization, editing, etc.?
Monads in Haskell can be thought of as *composable* computation descriptions.

In **functional programming**, a **monad** is a structure that represents **computation**.

For a Few **Monads** More

A Fistful of **Monads**

Input and Output

Chapter 14. **Monads**

In Chapter 7, I/O, we talked about the IO monad, but ...

**Concept:** Monad

**Concept:** State monad

**Concept:** Maybe monad

**Concept:** Writer monad

**Contribution:** haskellLogging

**Contribution:** haskellWriter

---

© 2013 101companies -- Presentation at IFL 2013
Research questions

( Teaching perspective )

How can a software chrestomathy (i.e., a systematic program collection) be leveraged in an online-oriented course?
The course concept

- **Integrate** resources; don’t favor one.
- Favor advanced **wiki** over slides.
- Favor **illustration** over explanation.
- Reproduce wiki-available **code** in lecture.
- Use a **running example** to a large extent.
- Homework, lab, exam as usual.
Vocabulary integration
Integrated resources

- Wikipedia
- HaskellWiki
- “Haskell: The Craft of Functional Programming” (CRAFT)
- “Programming in Haskell” (PIH)
- “Real World Haskell” (RWH)
- “Learn You a Haskell” (LYAH)
Vocabulary integration

- Vocabulary mining
- Vocabulary endorsement
- Vocabulary monitoring
Vocabulary integration

- Vocabulary mining
- Vocabulary endorsement
- Vocabulary monitoring
Vocabulary mining: I/O

**Input:** Textbook (preferably online)

**Output:**
- Chapter terms (locally popular terms)
- Book terms (globally popular terms)
- “DOM” ids for term occurrences
Vocabulary mining: procedure

- Index normalization
- Content normalization (likewise)
- Index/content matching
- Chapter term mining
- Book term mining
Vocabulary mining: procedure

- Index normalization
  - Remove subentries
  - Remove special symbols
  - Remove common English

- Content normalization (likewise)

- Index/content matching

- Chapter term mining

- Book term mining
Vocabulary mining: procedure

- Index normalization
- Content normalization
- Index/content matching

Chapter term mining

- Terms are ‘frequent enough’ in the book.
- Terms are in ‘few enough’ chapters.
- The ‘most frequent’ ones of those per chapter.

Book term mining
Chapter terms for CRAFT
<table>
<thead>
<tr>
<th>Term</th>
<th>Getting started with Haskell and GHCi</th>
<th>Basic types and definitions</th>
<th>Designing and writing programs</th>
<th>Data types tuples and lists</th>
<th>Programming with lists</th>
<th>Defining functions over lists</th>
<th>Playing the game IO in Haskell</th>
<th>Reasoning about programs</th>
<th>Generalization patterns of computation</th>
<th>Higher order functions</th>
<th>Developing higher order programs</th>
<th>Overloading type classes and type checking</th>
<th>Algebraic types</th>
<th>Case study Huffman codes</th>
<th>Abstract data types</th>
<th>Lazy programming</th>
<th>Programming with monads</th>
<th>Domain Specific Languages</th>
<th>Time and space behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>algebraic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>algebraic type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base case</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>calculation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>command</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>constructor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Term

Getting started with Haskell and GHCi

Basic types and definitions

Designing and writing programs

Data types tuples and lists

Programming with lists

Defining functions over lists

Playing the game IO in Haskell

Reasoning about programs

Generalization patterns of computation

Higher order functions

Developing higher order programs

Overloading type classes and type checking

Algebraic types

Case study Huффman codes

Abstract data types

Lazy programming

programming with monads

Domain Specific Languages

Table 3: Chapter profiles of CRAFT
Vocabulary mining: procedure

- Index normalization
- Content normalization
- Index/content matching
- Chapter term mining

**Book term mining**

- Order all matched terms by frequency.
- Remove all chapter terms.
- Select the `most frequent` terms.
Vocabulary integration

- Vocabulary mining
- Vocabulary endorsement
- Vocabulary monitoring
Vocabulary endorsement: I/O

Input:
- Textbook terms
- 101wiki
- Wikipedia et al.

Output:
- New 101wiki terms with Wikipedia et al. links
- Mapping of textbook terms to 101wiki terms
- Improved ontology
Vocabulary endorsement: procedure

Is the textbook term already on 101wiki?

- Yes; it's the same term.
  - Done
- Yes; it's a (slightly) different term.
  - Rename or map
- No; it's not on 101wiki.
  - Add the term.
Vocabulary endorsement: procedure

- Add a term to 101wiki:
  - Find the term on Wikipedia et al.
  - Create a 101wiki page.
  - Link it to Wikipedia.
- Maintain ontology.
Metadata for the Monad concept

* `[instanceOf::Type class]`
* `[isA::Programming idiom]`
* `[instanceOf::Namespace:Concept]`
* `[instanceOf::Vocabulary:Functional programming]`
* `[identifies::http://www.haskell.org/haskellwiki/Monad]`
* `[linksTo::http://en.wikipedia.org/wiki/Monad_(category_theory)]`
* `[linksTo::http://en.wikibooks.org/wiki/Haskell/Understanding_monads]`
* `[identifies::http://en.wikipedia.org/wiki/Monad_%28functional_programming%29]`
Vocabulary integration

- Vocabulary mining
- Vocabulary endorsement
- Vocabulary monitoring
Vocabulary monitoring

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language: Haskell (12/0)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Technology: GHC (11/1)</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Algebraic data type (4/8)</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Monad (4/9)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Anonymous function (2/10)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Map function (2/10)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Data constructor (2/6)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Recursion (2/2)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Fold function (2/1)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Higher-order function (2/1)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Monoid (1/11)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Technology: GHCi (1/11)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Technology: Parsec (1/11)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Local scope (1/5)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Module (1/5)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Tuple (1/3)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Writer monad (1/3)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Parsing (1/2)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Introductory Haskell contributions

Textbook terms
Vocabulary monitoring: objective

- Use terms where appropriate.
- Add contributions where needed.
- Understand relevance of contributions.
- Understand characteristics of resources.
Vocabulary monitoring: basic procedure per resource

- Select contributions
  - By enumeration
  - By programming language
- Compute concepts $\times$ contributions matrix
  - Positive (+) and negative (−) references
- Direct and indirect references
## Vocabulary monitoring: comparison of vocabularies

| Terms in CRAFT only: | Local scope, Value, Complexity, Proof, Calculation, Equational reasoning, Head, Equality, Programming, Queue, Argument, Result, Base case, Partial application, Program, Tuple, Set, Program design, Type checking, Higher-order function, Name, Algebraic data type, Infinite list, Float |
| Terms in PIH only: | Haskell script, too generic term, Equation, Function application, Parser combinator, Identity element, Declaration, Function definition, Product function, Lambda abstraction |
| Terms in RWH only: | Foreign function interface, Predicate, Operator precedence, Polymorphism, Thread, Performance, MVar, Profiling, TCP, Directory, Property, Loop, Technology:Parsec, Parsing, Monad transformer, Pointer, Technology:HPC, Type system, User interface, Language:XML, Core, Technology:Glade, Exception, Error, Process, Type signature, Type definition, Program optimization, Data type, Technology:GHC, Pure function, Association list, Query, Output, UDP, Table |
| Terms in LYAH only: | Fmap function, Accumulator, type-class instance, Functor, Data structure, Monadic value, Import, Factorial, Zipper, Condition, Expression, Sum function, Applicative functor |
| Terms in more than one book: | Monoid, Character, Type-class instance, Bit, List comprehension, Testing, Fold function, Operator, Lazy evaluation, Recursion, I/O system, Number, State, Input, Haskell package, Type, String, Type class, Random number, Tree, Command, Parser, Filter function, Code, Data constructor, Pattern, Integer, Database, Catamorphism, Evaluation strategy, Action, Technology:GHCi, Text, Tail, Regular expression, Map function, Language:Haskell, Induction, Function, Pattern matching, Prelude, Stack, Eager evaluation, List, Maybe type, Monad, Module, Guard, Boolean, File |

© 2013 101companies -- Presentation at IFL 2013
| Terms in CRAFT only: | Local scope, Value, Complexity, Proof, Calculation, Equa-
| | tional reasoning, Head, Equality, Programming, Queue, Argument, Result, Base
| | case, Partial application, Program, Tuple, Set, Program design, Type checking, 
| | Higher-order function, Name, Algebraic data type, Infinite list, Float

| Terms in PIH only: | Haskell script, too generic term, Equation, Function ap-
| | plication, Parser combinator, Identity element, Declaration, Function definition, 
| | Product function, Lambda abstraction

| Terms in RWH only: | Foreign function interface, Predicate, Operator precedence, 
| | Polymorphism, Thread, Performance, MVar, Profiling, TCP, Directory, Property, 
| | Loop, Technology:Parsec, Parsing, Monad transformer, Pointer, Technology:HPC, 
| | Type system, User interface, Language:XML, Core, Technology:Glade, Exception, 
| | Error, Process, Type signature, Type definition, Program optimization, Data type, 
| | Technology:GHC, Pure function, Association list, Query, Output, UDP, Table

| Terms in LYAH only: | Fmap function, Accumulator, type-class instance, Functor, 
| | Data structure, Monadic value, Import, Factorial, Zipper, Condition, Expression, 
| | Sum function, Applicative functor

| Terms in more than one book: | Monoid, Character, Type-class instance, Bit, 
| | List comprehension, Testing, Fold function, Operator, Lazy evaluation, Recursion, 
| | I/O system, Number, State, Input, Haskell package, Type, String, Type class, Random 
| | number, Tree, Command, Parser, Filter function, Code, Data constructor, 
| | Pattern, Integer, Database, Catamorphism, Evaluation strategy, Action, Technology:GHCi, 
| | Text, Tail, Regular expression, Map function, Language:Haskell, Induc-
| | tion, Function, Pattern matching, Prelude, Stack, Eager evaluation, List, Maybe 
| | type, Monad, Module, Guard, Boolean, File

© 2013 101companies -- Presentation at IFL 2013
Knowledge management

Time is up!? See past the `last' slide.
That’s it.

Comments?

Questions?

Let’s chat offline about ...

- making 101haskell contributions,
- using / enhancing the course concept,
- ...

© 2013 101companies -- Presentation at IFL 2013
Knowledge management

Not covered in the conference presentation
Knowledge management

- Documentation management
- Feature management
- Similarity management
- Theme management
- Ontology management
- ...

© 2013 101companies -- Presentation at IFL 2013
Knowledge management

- Documentation management
  - Feature management
  - Similarity management
  - Theme management
  - Ontology management
  - ...

© 2013 101companies -- Presentation at IFL 2013
Generic programming in Haskell with SYB

Motivation ...

Illustration

Several of the operations on companies can be implemented in a very concise manner based on the SYB style of generic programming. For instance, the operation for totaling salaries simply extracts all floats from the given term and reduces them by addition:

```
-- Total all salaries in a company
total :: Company -> Float
total = everything (+) (extQ (const 0) id)
```

Architecture ...

Usage ...
Illustration

Several of the operations on companies can be implemented in a very concise manner based on the SYB style of generic programming. For instance, the operation for totaling salaries simply extracts all floats from the given term and reduces them by addition:

```
-- Total all salaries in a company
total :: Company -> Float
total = everything (+) (extQ (const 0) id)
```

== Illustration ==

Several of the operations ...

<fragment url="src/Company/Total.hs/pattern/total"/>
Knowledge management

- Documentation management
- **Feature management**
- Similarity management
- Theme management
- Ontology management
- ...

© 2013 101companies -- Presentation at IFL 2013
Feature management

- **Company**: Companies, department, employees
- **Total**: Total the salaries of employees
- **Median**: Compute the median of the salaries
- **Cut**: Cut the salaries of employees in half
- **Depth**: Compute nesting depth of departments
- **COI**: Conflicts of interests for employees
- **Mentoring**: Associate mentors and mentees
- **Ranking**: Enforce salary to correlate with ranks
- **Singleton**: Constrain for a single company
- **History**: Maintain and analyze company history
- **Serialization**: De-/serialize companies
- **Persistence**: Persist companies
- **Mapping**: Map companies across technological space
- **Distribution**: Distribute companies
- **Parallelism**: Total or cut in parallel
- **Logging**: Log company changes
- **Browsing**: Browse companies interactively
- **Editing**: Edit companies interactively
- **Restructuring**: Restructure companies interactively
- **Web UI**: Operate on companies in a web browser
- **Parsing**: Parse companies in concrete syntax
- **Unparsing**: Pretty print companies

* Rename feature
* Add feature
* Remove feature
* Split feature
* Revise feature
* ...
Knowledge management

- Documentation management
- Feature management
- Similarity management
- Theme management
- Ontology management

...
(Code) similarity management

-- Files shared by contributions

"Company/Data.hs" {haskellApplicative, haskellParsec},
{haskellComposition, monoidal}, {haskellEngineer, haskellLambda},
{haskellSyb, hughesPJ, nonmonadic, writerMonad}, {haskellTermRep, haskellTree, tabaluga}

"Company/Median.hs" {haskellBarchart, haskellProfessional}

"Company/Sample.hs" {haskellApplicative, haskellParsec},
{haskellComposition, monoidal}, {haskellEngineer, haskellLambda, haskellList}, {haskellSyb, hughesPJ, nonmonadic, writerMonad}

"Company/Log.hs" {nonmonadic, writerMonad}

"Main.hs" {haskellApplicative, haskellParsec}, {haskellComposition, haskellData, haskellEngineer, haskellLambda, haskellList, haskell-Record, haskellVariation}, {nonmonadic, writerMonad}, {strafunski, tabaluga}

"Company/SampleCompany.hs" {mvar, wxHaskell}
Knowledge management

- Documentation management
- Feature management
- Similarity management
- Theme management
- Ontology management
- ...

© 2013 101companies -- Presentation at IFL 2013
The introductory Haskell theme

http://101companies.org/wiki/Theme:Haskell_introduction

- **haskellStarter**: Basics of functional programming
- **haskellEngineer**: Basics of software engineering
- **haskellList**: List processing with map and friends
- **haskellProfessional**: Idiomatic code for many features
- **haskellLambda**: Anonymous functions
- **haskellComposition**: Recursive algebraic data types
- **haskellVariation**: Multiple constructors per type
- **haskellMonoid**: Queries in monoidal style
- **haskellLogging**: Logging in non-monadic style
- **haskellWriter**: Logging in monadic style
- **haskellParsec**: Parsing with the Parsec library
- **haskellSyb**: Generic programming à la SYB style
The Haskell genericity theme

http://101companies.org/wiki/Theme:Haskell_genericity

- haskellSyb: "Scrap your boilerplate" style
- strafunski: Strategic programming
- haskellTree: Rose trees for representation
- tabaluga: Folds for systems of data types
The SYB theme

http://101companies.org/wiki/Theme:Scrap_your_boilerplate

- haskellSyb: Illustration of SYB for Haskell
- jsSyb: Illustration of SYB for JavaScript
- pythonSyb: Illustration of SYB for Python
- javaSyb: Illustration of SYB for Java
Knowledge management

- Documentation management
- Feature management
- Similarity management
- Theme management
- Ontology management
- ...

© 2013 101companies -- Presentation at IFL 2013
Mutually dependent dimensions of knowledge in ontology management

- **Programming domains**
  - Parsing, XML processing, GUI programming, ...

- **Programming concepts**
  - Parsing, parser, MVC, monad, ...

- **Programming techniques**
  - Higher-order functions, Monadic style

- **Programming technologies**
  - QuickCheck, Cabal, Parsec, ...

© 2013 101companies -- Presentation at IFL 2013
Beware: just 11 evaluation forms.

Course evaluation results

Not covered in the conference presentation
Semester

1  | 50%
2  | 20%
3  | 20%
4  | 10%

© 2013 101companies -- Presentation at IFL 2013
Number of missed lectures

- 0 mal: 40%
- 1 mal: 20%
- 2 mal: 20%
- 3 mal: 0%
- 4 mal: 10%
- >4 mal: 10%
Hours of study time / week

1 10%
4 50%
5 30%
8 10%
Helpful material?

- No
- Yes

© 2013 101companies -- Presentation at IFL 2013
Good use of didactic tools?

No

Yes
Well-prepared lecturer?

© 2013 101companies -- Presentation at IFL 2013
Comprehensible outline over semester
Substantially improved knowledge?

![Bar Chart]

© 2013 101companies -- Presentation at IFL 2013
Practical examples?

No   | 0%  | 0%  | 0%  | 25% | 37.5% | 37.5% | Yes
---|-----|-----|-----|-----|-------|-------|---
1   | 2   | 3   | 4   | 5   | 6     |       |