Haskell

... und jetzt …?

Dr. Jan Bessai, 2021-07-23

heise Jobs

Stellenanzeige schalten | IT-Jobtage | IT Job Kompakt |

Haskell

1 Job für die Suche nach Haskell

Jobmailer | Merkliste | Filtern und Sortieren

[redacted]

Machine Learning Spezialist (m/w/d)

04.07.2021

[redacted]

Merken

Paderborn

Startseite 1 weitere Jobs

Passende Jobs zu Ihrer Suche ...

... immer aktuell und kostenlos per E-Mail.

Ihre E-Mail-Adresse

Jobs kostenlos per E-Mail

Sie können den Suchauftrag jederzeit abbestellen.

Es gilt unsere Datenschutzerklärung. Sie erhalten passende Angebote per E-Mail. Sie können sich jederzeit wieder kostenlos abmelden.

Paderborn

[redacted] ist der Stellenmarkt für Fach- und Führungskräfte. Seit unserer Gründung 1996 stehen wir für erstklassige Qualität und hohe Kundenzufriedenheit. Heute unterstützen über 200 begeisterte [redacted] Unternehmen aller Größen und Branchen bei der Mitarbeitergewinnung.

Data Scientist. Developer. Analyst.

Sie arbeiten Hand in Hand mit Mathematikern und ML-Experten daran, das [redacted] Machine Learning zu erweitern und zu verbessern. Dabei bekommen Sie den Freiraum, den Sie brauchen, um Cutting-Edge-Technologien zu entwickeln.

Ihre Aufgaben:

- Entwicklung von Machine-Learning Anwendungen
- Weiterentwicklung bestehender Klassifikationssysteme
- Analyse von textuellen Daten
- Erstellung von umfangreichen Statistiken

Ihr Profil:

- Erfahrung in der Entwicklung von ML-Anwendungen
- Sehr gute Programmierkenntnisse, bevorzugt in Java und Python
- Erfahrung mit Pandas, Pytorch und Tensor Flow
- Nice to have: Kenntnisse in MongoDB, MySQL, Docker, Haskell

Es erwartet Sie:

Ein innovatives Unternehmen mit tollen Kollegen*innen. Außerdem:

- Freie Wahl von Entwicklungsumgebung & Betriebssystem
- Wasser- und Kaffeeflat, modernes Equipment inkl. Laptop, persönliche Entwicklung durch Weiterbildungen
- Flexible Arbeitszeiten mit Homeoffice-Anteil, 30 Tage Urlaub, vermögenswirksame Leistungen
- IT-interne Duz-Kultur

Quelle: https://jobs.heise.de/Suchergebnis?jsjn=Haskell&jsjnid=&jsjo=&jsjoid=&jsjr= (Abrufdatum: 23.7.2021)

Gründe?

Jul 2021	Jul 2020	Change	Programming Language	Ratings	Change
1	1		C	11.62%	-4.83%
2	2		Java	11.17%	-3.93%
3	3		Python	10.95%	+1.86%
4	4		C++	8.01%	+1.80%
5	5		C#	4.83%	-0.42%
6	6		Visual Basic	4.50%	-0.73%
7	7		JavaScript	2.71%	+0.23%
8	9	↑	PHP	2.58%	+0.68%
9	13	⇈	Assembly language	2.40%	+1.46%
10	11	↑	SQL	1.53%	+0.13%
11	20	⇈	Classic Visual Basic	1.39%	+0.73%
12	8	⇊	R	1.32%	-1.08%
13	12	↓	Go	1.17%	-0.04%
14	50	⇈	Fortran	1.12%	+0.90%
15	24	⇈	Groovy	1.09%	+0.51%
16	10	⇊	Swift	1.07%	-0.37%
17	16	↓	Ruby	0.95%	+0.14%
18	14	⇊	Perl	0.90%	+0.03%
19	15	⇊	MATLAB	0.88%	+0.05%
20	30	⇈	Delphi/Object Pascal	0.85%	+0.36%

Quelle: https://www.tiobe.com/tiobe-index/ (Abrufdatum: 23.7.2022)

Zurück zu …?

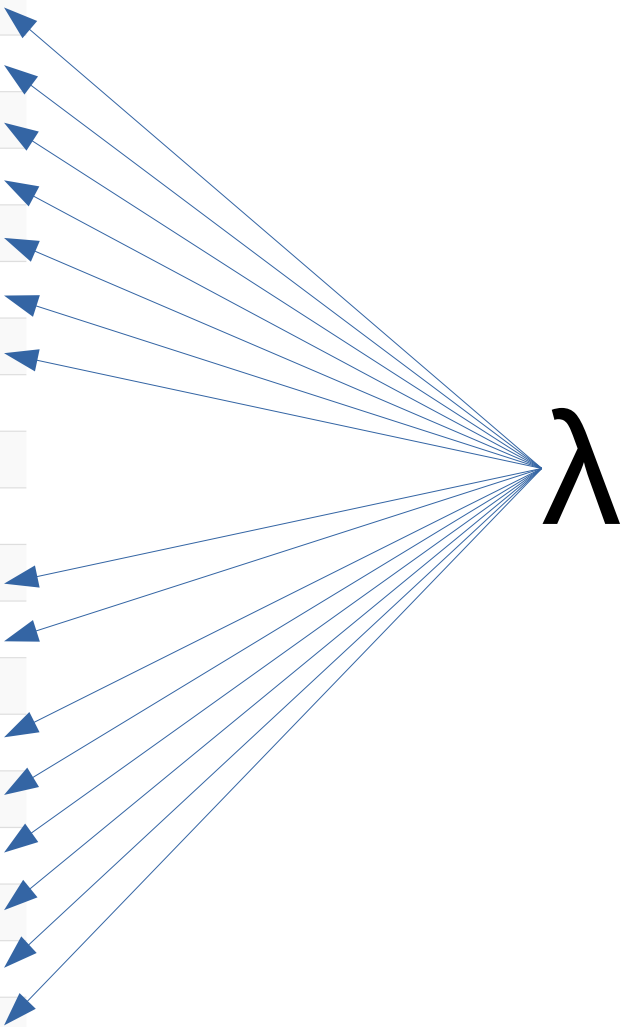
```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
... other checks ...
fail:
  ... buffer frees (cleanups) ...
  return err;
```

Quelle: https://dwheeler.com/essays/apple-goto-fail.html (Abrufdatum: 23.7.2021)

Vorlesung: Ideen & Techniken

λ-Funktionen, Typklassen, Immutability, Listen-Komprehensionen, Funktoren & Monaden, Faltungen und Traversierungsschemata, parametrische Polymorphie, starke Typsysteme, ...

Jul 2021	Jul 2020	Change	Programming Language	Ratings	Change
1	1		C	11.62%	-4.83%
2	2		Java	11.17%	-3.93%
3	3		Python	10.95%	+1.86%
4	4		C++	8.01%	+1.80%
5	5		C#	4.83%	-0.42%
6	6		Visual Basic	4.50%	-0.73%
7	7		JavaScript	2.71%	+0.23%
8	9	⌃	PHP	2.58%	+0.68%
9	13	⌃⌃	Assembly language	2.40%	+1.46%
10	11	⌃	SQL	1.53%	+0.13%
11	20	⌃⌃	Classic Visual Basic	1.39%	+0.73%
12	8	⌄⌄	R	1.32%	-1.08%
13	12	⌄	Go	1.17%	-0.04%
14	50	⌃⌃	Fortran	1.12%	+0.90%
15	24	⌃⌃	Groovy	1.09%	+0.51%
16	10	⌄⌄	Swift	1.07%	-0.37%
17	16	⌄	Ruby	0.95%	+0.14%
18	14	⌄⌄	Perl	0.90%	+0.03%
19	15	⌄⌄	MATLAB	0.88%	+0.05%
20	30	⌃⌃	Delphi/Object Pascal	0.85%	+0.36%



Quelle: https://www.tiobe.com/tiobe-index/ (Abrufdatum: 23.7.2022)

ECOOP 2021: λ-Based Object-Oriented Programming

```
interface Person{
  String name();
  default String greet(){
    return "Hi, I'm "+this.name()+"; nice to meet you!";
  }
}
Person bob = ()->"bob";
bob.greet();

interface GamerPerson extends Person{
  default String greet(){
    return "Hi, I'm "+this.name()+"; and I love computer games!";
  }
}
Person p = (GamerPerson)()->"charles";
p.greet();// dynamic dispatch
```

Quelle: Servetto, Marco, and Elena Zucca. "λ-Based Object-Oriented Programming (Pearl)." 35th European Conference on Object-Oriented Programming (ECOOP 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. DOI: https://doi.org/10.4230/LIPIcs.ECOOP.2021.21

OO + FP


```scala
sealed abstract class List[+A] extends AbstractSeq[A] /* ... */ {
  /* ... */
  final override def flatMap[B](f: A => IterableOnce[B]): List[B] = {
    var rest = this
    var h: ::[B] = null
    var t: ::[B] = null
    while (rest ne Nil) {
      val it = f(rest.head).iterator
      while (it.hasNext) {
        val nx = new ::(it.next(), Nil)
        if (t eq null) {
          h = nx
        } else {
          t.next = nx
        }
        t = nx
      }
      rest = rest.tail
    }
    if (h eq null) Nil else {releaseFence(); h}
  }
  /* ... */
}
```

Logos: wikipedia.org (Abrufdatum: 23.7.2021), Code: https://raw.githubusercontent.com/scala/scala/v2.13.6/src/library/scala/collection/immutable/List.scala (Abrufdatum: 23.7.2021)

Go-Interfaces (Typklassen)

```
type geometry interface {
        area() float64
        perim() float64
}
type rect struct { width, height float64 }
type circle struct { radius float64 }

func (r rect) area() float64 { return r.width * r.height }
func (r rect) perim() float64 { return 2*r.width + 2*r.height }
func (c circle) area() float64 { return math.Pi * c.radius * c.radius }
func (c circle) perim() float64 { return 2 * math.Pi * c.radius }

func measure(g geometry) {
        fmt.Println(g)
        fmt.Println(g.area())
        fmt.Println(g.perim())
}
func main() {
        r := rect{width: 3, height: 4}
        c := circle{radius: 5}
        measure(r)
        measure(c)
}
```

Quelle: https://gobyexample.com/interfaces (Abrufdatum: 23.7.2021)

ImmutableCollectionsExplained

Immutable Collections

Example

```
public static final ImmutableSet<String> COLOR_NAMES = ImmutableSet.of(
  "red",
  "orange",
  "yellow",
  "green",
  "blue",
  "purple");

class Foo {
  final ImmutableSet<Bar> bars;
  Foo(Set<Bar> bars) {
    this.bars = ImmutableSet.copyOf(bars); // defensive copy!
  }
}
```

Why?

Immutable objects have many advantages, including:

- Safe for use by untrusted libraries.
- Thread-safe: can be used by many threads with no risk of race conditions.
- Doesn't need to support mutation, and can make time and space savings with that assumption. All immutable collection implementations are more memory-efficient than their mutable siblings. (analysis)
- Can be used as a constant, with the expectation that it will remain fixed.

Making immutable copies of objects is a good defensive programming technique. Guava provides simple, easy-to-use immutable versions of each standard `Collection` type, including Guava's own `Collection` variations.

Quelle: https://github.com/google/guava/wiki/ImmutableCollectionsExplained (Abrufdatum: 23.7.2021)

5.1.4. Nested List Comprehensions

The initial expression in a list comprehension can be any arbitrary expression, including another list comprehension.

Consider the following example of a 3x4 matrix implemented as a list of 3 lists of length 4:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

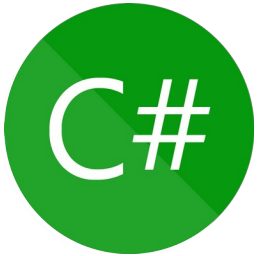
The following list comprehension will transpose rows and columns:

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

As we saw in the previous section, the nested listcomp is evaluated in the context of the `for` that follows it, so this example is equivalent to:

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Logo: wikipedia.org (Abrufdatum 23.7.2021)
Text & Code: https://docs.python.org/3/tutorial/datastructures.html (Abrufdatum: 23.7.2021)



And the (Select, Multiply, Unit) methods can be implemented with (SelectMany, Wrap) methods too:

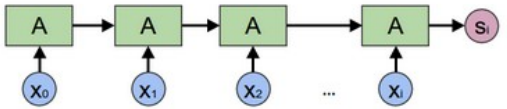
```
public static partial class EnumerableExtensions // (SelectMany, Wrap) implements (Select, Multiply, Unit).
{
    // Select: (TSource -> TResult) -> (IEnumerable<TSource> -> IEnumerable<TResult>).
    public static Func<IEnumerable<TSource>, IEnumerable<TResult>> Select<TSource, TResult>(
        Func<TSource, TResult> selector) => source =>
            from value in source
            from result in value.Enumerable()
            select result;
            // source.SelectMany(Enumerable, (result, value) => value);

    // Multiply: IEnumerable<IEnumerable<TSource>> -> IEnumerable<TSource>
    public static IEnumerable<TSource> Multiply<TSource>(this IEnumerable<IEnumerable<TSource>> sourceWrapper) =>
        from source in sourceWrapper
        from value in source
        select value;
        // sourceWrapper.SelectMany(source => source, (source, value) => value);

    // Unit: TSource -> IEnumerable<TSource>
    public static IEnumerable<TSource> Unit<TSource>(TSource value) => value.Enumerable();
}
```

So monad support is built-in in the C# language. As discussed in the LINQ query expression pattern part, SelectMany enables multiple from clauses, which can chain operations together to build a workflow, for example:

```
internal static void Workflow<T1, T2, T3, T4>(
    Func<IEnumerable<T1>> source1,
    Func<IEnumerable<T2>> source2,
    Func<IEnumerable<T3>> source3,
    Func<T1, T2, T3, IEnumerable<T4>> source4)
{
    IEnumerable<T4> query = from value1 in source1()
                            from value2 in source2()
                            from value3 in source3()
                            from value4 in source4(value1, value2, value3)
                            select value4; // Define query.
    query.WriteLines(); // Execute query.
}
```

Logo: wikipedia.org (Abrufdatum: 23.7.2021)
Text & Code: https://weblogs.asp.net/dixin/category-theory-via-csharp-7-monad-and-linq-to-monads (Abrufdatum: 23.7.2021)

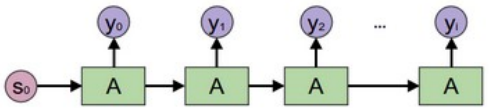
- **Encoding Recurrent Neural Networks** are just folds. They're often used to allow a neural network to take a variable length list as input, for example taking a sentence as input.



fold = Encoding RNN

Haskell: `foldl a s`

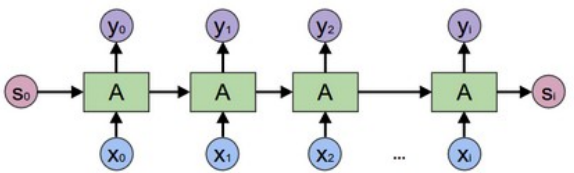
- **Generating Recurrent Neural Networks** are just unfolds. They're often used to allow a neural network to produce a list of outputs, such as words in a sentence.



unfold = Generating RNN

Haskell: `unfoldr a s`

- **General Recurrent Neural Networks** are accumulating maps. They're often used when we're trying to make predictions in a sequence. For example, in voice recognition, we might wish to predict a phenome for every time step in an audio segment, based on past context.



Accumulating Map = RNN

Haskell: `mapAccumR a s`

Quelle: Christopher Olah.
"Neural Networks, Types, and Functional Programming".
URL: http://colah.github.io/posts/2015-09-NN-Types-FP/
(Abrufdatum: 23.7.2021)

THE C PROGRAMMING LANGUAGE

6 A generic selection is a primary expression. Its type and value depend on the selected generic association, as detailed in the following subclause.

[...]

Syntax

1

```
generic-selection:
    _Generic ( assignment-expression , generic-assoc-list )
generic-assoc-list:
    generic-association
    generic-assoc-list , generic-association
generic-association:
    type-name : assignment-expression
    default : assignment-expression
```

[...]

5 EXAMPLE The cbrt type-generic macro could be implemented as follows:

```
#define cbrt(X) _Generic((X),                \
                long double: cbrtl,          \
                default: cbrt,               \
                float: cbrtf                 \
                )(X)
```

Logo: wikipedia.org (Abrufdatum: 23.7.2021)
Text: "Programming languages – C, Committee Draft – April 12, 2011", ISO/IEC 9899:201x
URL: https://port70.net/~nsz/c/c11/n1570.txt (Abrufdatum: 23.7.2021)

Dependent Method Types

Let's consider the following example of a heterogenous database that can store values of different types. The key contains the information about what's the type of the corresponding value:

```
trait Key { type Value }

trait DB {
  def get(k: Key): Option[k.Value] // a dependent method
}
```

Given a key, the method `get` lets us access the map and potentially returns the stored value of type `k.Value`. We can read this *path-dependent type* as: "depending on the concrete type of the argument `k`, we return a matching value".
For example, we could have the following keys:

```
object Name extends Key { type Value = String }
object Age extends Key { type Value = Int }
```

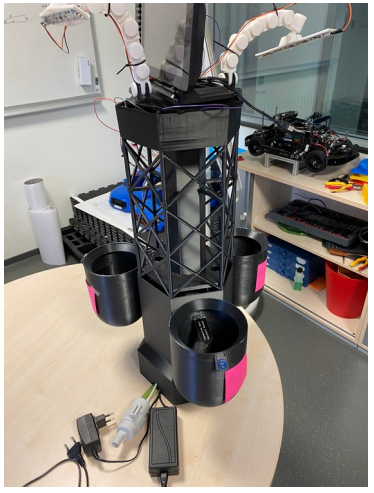
The following calls to method `get` would now type check:

```
val db: DB = ...
val res1: Option[String] = db.get(Name)
val res2: Option[Int] = db.get(Age)
```

Calling the method `db.get(Name)` returns a value of type `Option[String]`, while calling `db.get(Age)` returns a value of type `Option[Int]`. The return type *depends* on the concrete type of the argument passed to `get` —hence the name *dependent type*.

Logo: wikipedia.org (Abrufdatum 23.7.2021)
Text: https://docs.scala-lang.org/scala3/book/types-dependent-function.html (Abrufdatum: 23.7.2021)

Und mehr..?



Bachelor:

- Proseminar: Geschichte der Programmierung (SoSe, Prof. Rehof)
- Abschlussarbeiten (SoSe/WiSe, LS 14 / AG SEAL)

Master:

- Vertiefung: Logische Methoden des Software Engineering I+2 (WiSe, Prof. Rehof)
- Vertiefung: Aktuelle Themen im logikbasierten Software Engineering (SoSe, Dr. Czajka)
- Vertiefung: Type Systems for Correctness and Security (SoSe/WiSe, Prof. Hermann)
- Seminar: Seminar Principles of Programming Languages (WiSe, Prof. Rehof)
- Projektgruppen (WiSe/SoSe, Dr. Hildebrand & Dr. Bessai)
- Abschlussarbeiten (SoSe/WiSe, LS 14 / AG SEAL)