☺ Freie Monaden ☺

Ingo Blechschmidt
<iblech@speicherleck.de>

10. September 2015
Augsburger Curry-Club

1. Monoide
   - Definition und Beispiele
   - Nutzen
   - Freie Monoide
2. Funktoren
   - Definition und Beispiele
   - Funktoren als Container
3. Monaden
   - Definition und Beispiele
   - „Monoid in einer Kategorie von Endofunktoren“
4. Freie Monaden
   - Definition
   - Konstruktion
   - Nutzen

Monoide

```haskell
class Monoid m where
    (<>) :: m -> m -> m
    unit :: m
```

Ein **Monoid** besteht aus

- einer Menge M,
- einer Abbildung $(\circ) : M \times M \to M$ und
- einem ausgezeichneten Element $e \in M$,

sodass die **Monoidaxiome** gelten: Für alle $x, y, z \in M$

- $x \circ (y \circ z) = (x \circ y) \circ z$,
- $e \circ x = x$,
- $x \circ e = x$.

Monoide

```haskell
class Monoid m where
    (<>) :: m -> m -> m
    unit :: m
```

Ein **Monoid** besteht aus

- einer Menge M,
- einer Abbildung $(\circ) : M \times M \to M$ und
- einem ausgezeichneten Element $e \in M$,

sodass die **Monoidaxiome** gelten: Für alle $x, y, z \in M$

- $x \circ (y \circ z) = (x \circ y) \circ z$,
- $e \circ x = x$,
- $x \circ e = x$.

Beispiele:

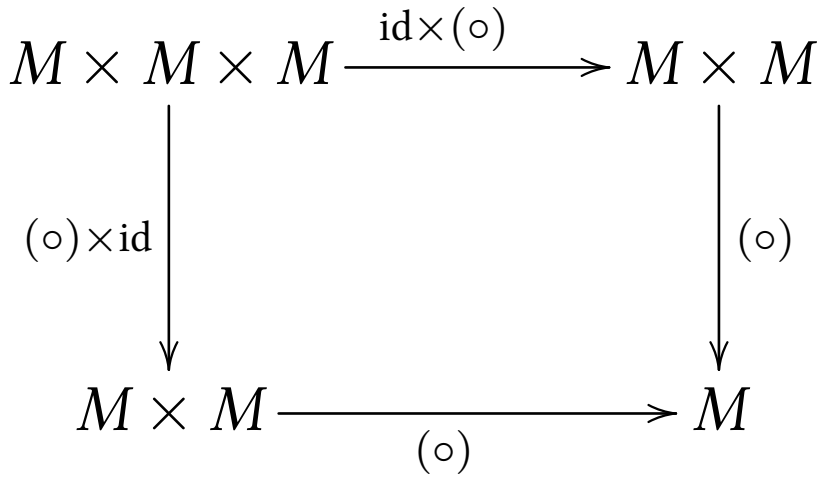
natürliche Zahlen, Listen, Endomorphismen, Matrizen, ...

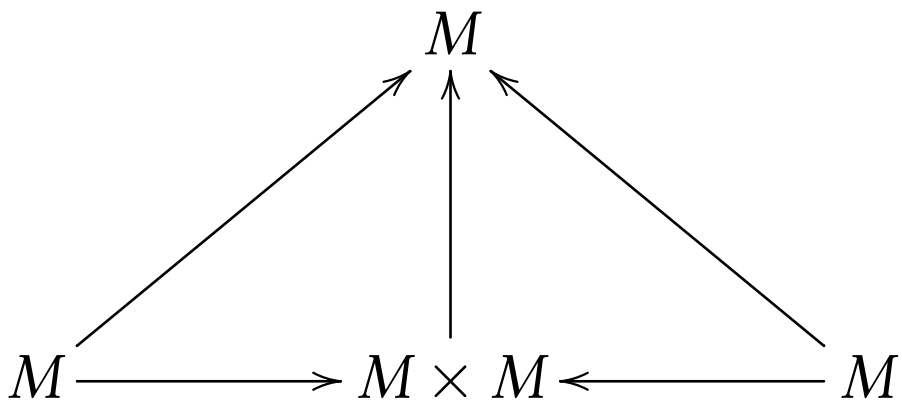
Nichtbeispiele:

natürliche Zahlen mit Subtraktion, nichtleere Listen, ...

- Die natürlichen Zahlen bilden mit der Addition als Verknüpfung und der Null als ausgezeichnetes Element einen Monoid.
- Die natürlichen Zahlen bilden mit der Multiplikation als Verknüpfung und der Eins als ausgezeichnetes Element einen Monoid.
- Die Menge $X^\star$ der endlichen Listen mit Einträgen aus einer Menge X (in Haskell also sowas wie `[X]`, wobei da auch unendliche und partiell definierte Listen dabei sind) bildet mit der Konkatenation von Listen als Verknüpfung und der leeren Liste als ausgezeichnetes Element einen Monoid.
- Ist X irgendeine Menge, so bildet die Menge aller Abbildungen $X \to X$ mit der Abbildungskomposition als Verknüpfung und der Identitätsabbildung als ausgezeichnetes Element einen Monoid.
- Die natürlichen Zahlen bilden mit der Subtraktionen keinen Monoid, da die Differenz zweier natürlicher Zahlen nicht immer wieder eine natürliche Zahl ist. Auch mit den ganzen Zahlen klappt es nicht, da die Subtraktion nicht assoziativ ist: $1 - (2 - 3) \neq (1 - 2) - 3$.

Axiome in Diagrammform





Beide Diagramme sind Diagramme in der Kategorie der Mengen, d. h. die beteiligten Objekte M, $M \times M$ und $M \times M \times M$ sind Mengen und die vorkommenden Pfeile sind Abbildungen. Wer möchte, kann aber auch vorgeben, dass M ein Typ in Haskell ist (statt $M \times M$ muss man dann `(M,M)` denken) und dass die Pfeile Haskell-Funktionen sind.

Das obere Diagramm ist wie folgt zu lesen.

Ein beliebiges Element aus $M \times M \times M$ hat die Form (x, y, z), wobei x, y und z irgendwelche Elemente aus M sind. Bilden wir ein solches Element nach rechts ab, so erhalten wir $(x, y \circ z)$. Bilden wir dieses weiter nach unten ab, so erhalten wir $x \circ (y \circ z)$.

Wir können aber auch den anderen Weg gehen: Bilden wir (x, y, z) erst nach unten ab, so erhalten wir $(x \circ y, z)$. Bilden wir dieses Element weiter nach rechts ab, so erhalten wir $(x \circ y) \circ z$.

Fazit: Genau dann *kommutiert das Diagramm* – das heißt beide Wege liefern gleiche Ergebnisse – wenn die Rechenregel $x \circ (y \circ z) = (x \circ y) \circ z$ für alle Elemente $x, y, z \in M$ erfüllt ist.

Die Pfeile auf dem unteren Diagramm sind nicht beschriftet. Hier die Erklärung, was gemeint ist.

Wir betrachten dazu ein beliebiges Element $x \in M$ (untere linke Ecke im Diagramm). Nach rechts abgebildet erhalten wir (e, x). Dieses Element weiter nach oben abgebildet ergibt $e \circ x$.

Der direkte Weg mit dem Nordost verlaufenden Pfeil ergibt x.

Das linke Teildreieck kommutiert also genau dann, wenn die Rechenregel $e \circ x = x$ für alle $x \in M$ erfüllt ist. Analog kommutiert das rechte Teildreieck genau dann, wenn $x \circ e = x$ für alle $x \in M$.

Wieso der Umstand mit der Diagrammform der Axiome? Das hat verschiedene schwache Gründe (es ist cool), aber auch einen starken inhaltlichen: Ein Diagramm dieser Art kann man nicht nur in der Kategorie interpretieren, in der es ursprünglich gedacht war (also der Kategorie der Mengen). Man kann es auch in anderen Kategorien interpretieren. Wählt man dazu speziell eine Kategorie von Endofunktoren, so erhält man die Definition einer Monade.

Monoidhomomorphismen

Eine Abbildung $\varphi : M \to N$ zwischen Monoiden heißt genau dann **Monoidhomomorphismus**, wenn

- $\varphi(e) = e$ und
- $\varphi(x \circ y) = \varphi(x) \circ \varphi(y)$ für alle $x, y \in M$.

Beispiele:

```
length :: [a] -> Int
sum :: [Int] -> Int
```

Nichtbeispiele:

```
reverse :: [a] -> [a]
head :: [a] -> a
```

- Es gelten die Rechenregeln

$$\texttt{length []} = 0,$$
$$\texttt{length (xs ++ ys)} = \texttt{length xs + length ys}$$

  für alle Listen `xs` und `ys`. Das ist der Grund, wieso die Längenfunktion ein Monoidhomomorphismus ist.

  Achtung: Bevor man eine solche Aussage trifft, muss man eigentlich genauer spezifizieren, welche Monoidstrukturen man in Quelle und Ziel meint. Auf dem Typ `[a]` soll die Monoidverknüpfung durch Konkatenation gegeben sein, auf dem Typ `Int` durch Addition.

- Es gilt die Rechenregel

$$\texttt{sum (xs ++ ys) = sum xs + sum ys}$$

  für alle Listen `xs` und `ys` von Ints. Das ist die halbe Miete zur Begründung, wieso `sum` ein Monoidhomomorphismus ist.

- Die Rechenregel

$$\texttt{reverse (xs ++ ys) = reverse xs ++ reverse ys.}$$

  gilt *nicht* für alle Listen `xs` und `ys`. Daher ist `reverse` kein Monoidhomomorphismus.

Wozu?

- Allgegenwärtigkeit
- Gemeinsamkeite und Unterschiede
- Generische Beweise
- Generische Algorithmen

- Monoide gibt es überall.
- Das Monoidkonzept hilft, Gemeinsamkeiten und Unterschiede zu erkennen und wertschätzen zu können.
- Man kann generische Beweise für beliebige Monoide führen.
- Man kann generische Algorithmen mit beliebigen Monoiden basteln.

Freie Monoide

Gegeben eine Menge X ohne weitere Struktur. Wie können wir auf möglichst unspektakuläre Art und Weise daraus einen Monoid $F(X)$ gewinnen?

Freie Monoide

Gegeben eine Menge X ohne weitere Struktur. Wie können wir auf möglichst unspektakuläre Art und Weise daraus einen Monoid $F(X)$ gewinnen?

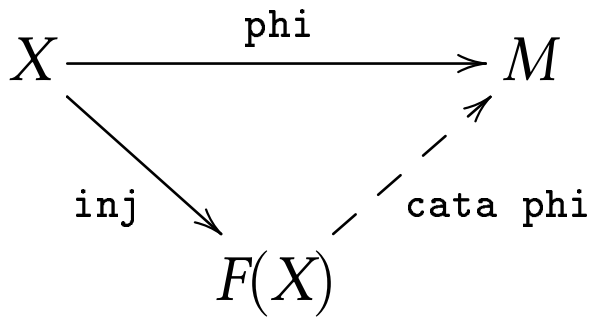
Spoiler: Der gesuchte **freie Monoid** $F(X)$ auf X ist der Monoid der endlichen **Listen** mit Elementen aus X.

Freie Monoide

Gegeben eine Menge X ohne weitere Struktur. Wie können wir auf möglichst unspektakuläre Art und Weise daraus einen Monoid $F(X)$ gewinnen?

Spoiler: Der gesuchte **freie Monoid** $F(X)$ auf X ist der Monoid der endlichen **Listen** mit Elementen aus X.

Essenz der Freiheit: Jede beliebige Abbildung $X \to M$ in einen Monoid M stiftet genau einen Homomorphismus $F(X) \to M$.



In $F(X)$ sollen neben den Elementen aus X gerade nur so viele weitere Elemente enthalten sein, sodass man eine Verknüpfung $(\circ)$ und ein Einselement definieren kann. Es soll sich also jedes Element aus $F(X)$ über die Monoidoperationen aus denen von X bilden lassen.

In $F(X)$ sollen nur die Rechenregeln gelten, die von den Monoidaxiomen gefordert werden.

Die nach „Essenz der Freiheit“ angegebene Forderung heißt auch *universelle Eigenschaft*. Man kann sie schön in Haskell demonstrieren:

```
cata :: (Monoid m) => (a -> m) -> ([a] -> m)
cata phi []     = unit
cata phi (x:xs) = phi x <> cata phi xs
```

Ist `phi` irgendeine Abbildung `a -> m`, so ist `cata phi` eine Abbildung `[a] -> m`. Diese ist nicht nur irgendeine Abbildung, sondern wie gefordert ein Monoidhomomorphismus. Und mehr noch: Das Diagramm kommutiert tatsächlich, das heißt `cata phi` und `phi` haben in folgendem präzisen Sinn etwas miteinander zu tun: `cata phi . inj = phi`. Dabei ist `inj :: a -> [a]` die Abbildung mit `inj x = [x]`.

Unsere Definition von „unspektakulär“ soll sein: Alle Elemente in $F(X)$ setzen sich mit den Monoidoperationen aus denen aus X zusammen, und in $F(X)$ gelten nur die Rechenregeln, die von den Monoidaxiomen erzwungen werden, aber keine weiteren.

Das ist bei der Konstruktion von $F(X)$ als Monoid der endlichen Listen über X auch in der Tat der Fall: Jede endliche Liste mit Einträgen aus X ergibt sich als wiederholte Verknüpfung (Konkatenation) von Listen der Form `inj x` mit `x` aus X. Und willkürliche Rechenregeln wie `[a,b] ++ [b,a] == [c]` gelten, wie es auch sein soll, *nicht*.

Wieso fängt die universelle Eigenschaft genau diese Definition von Unspektakularität ein? Wenn man zu $F(X)$ noch weitere Elemente hinzufügen würde (zum Beispiel unendliche Listen), so wird es mehrere oder keinen Kandidaten für `cata phi` geben, aber nicht mehr nur einen. Genauso wenn man in $F(X)$ durch Identifikation gewisser Elemente weitere Rechenregeln erzwingen würde.

Wenn man tiefer in das Thema einsteigt, erkennt man, dass endliche Listen noch nicht der Weisheit letzter Schluss sind: `http://comonad.com/reader/2015/free-monoids-in-haskell/`

Kurz zusammengefasst: Es stimmt schon, dass der Monoid der endlichen Listen über X der freie Monoid auf X ist, sofern man als Basiskategorie in der Kategorie der Mengen arbeitet. Wenn man dagegen in Hask arbeitet, der Kategorie der Haskell-Typen und -Funktionen, so benötigt man eine leicht andere Konstruktion.

Fun Fact: Die in dem Blog-Artikel angegebene Konstruktion ergibt sich *unmittelbar* aus der universellen Eigenschaft. Es ist keine Überlegung und manuelle Suche nach einem geeigneten Kandidaten für $F(X)$ nötig. (Bemerkung für Kategorientheorie-Fans: Das es so einfach geht, liegt an einer gewissen Vollständigkeitseigenschaft von Hask.)

Freie Monoide

Freie Monoide sind …

… frei wie in Freibier?

Freie Monoide

Freie Monoide sind …

~~… frei wie in Freibier?~~

Freie Monoide

Freie Monoide sind …

~~… frei wie in Freibier?~~

… frei wie in Redefreiheit?

Freie Monoide

Freie Monoide sind …

~~… frei wie in Freibier?~~

~~… frei wie in Redefreiheit?~~

Freie Monoide

Freie Monoide sind …

~~… frei wie in Freibier?~~

~~… frei wie in Redefreiheit?~~

… frei wie in **linksadjungiert**! ✓

Freie Monoide

Freie Monoide sind …

~~… frei wie in Freibier?~~

~~… frei wie in Redefreiheit?~~

… frei wie in **linksadjungiert**! ✓

Der Funktor $F : \mathrm{Set} \to \mathrm{Mon}$ ist **linksadjungiert** zum Vergissfunktor $\mathrm{Mon} \to \mathrm{Set}$.