EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät

Programmiersprachen und Softwaretechnik

Prof. Klaus Ostermann

Leitung des Seminars
David Binder
Ingo Skupin

Kategorientheorie für Programmierer

Hausaufgabenblatt 8 – SS18

Tübingen, 2. Juli 2018

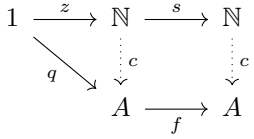
Aufgabe 1: Lektüre

Für die kommende Woche lesen Sie bitte Kapitel 24 ohne Abschnitt 24.6 (Coalgebras) und schicken Ihre Fragen bis Dienstag Abend an uns.

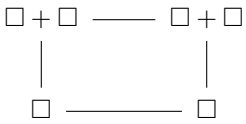
Aufgabe 2: Natural Number Object (NNO)

In einer Kategorie $\mathcal{C}$ mit terminalem Objekt 1 lässt sich die kategorielle Variante von natürlichen Zahlen definieren: ein natural number object. (Wie immer gilt, dass nicht jede Kategorie natural number objects besitzt.)

Ein natural number object ist gegeben durch ein Objekt $\mathbb{N}$ und Morphismen $z : 1 \to \mathbb{N}$ sowie $s : \mathbb{N} \to \mathbb{N}$ mit folgender universeller Eigenschaft: Für jedes Objekt A und Morphismen $q : 1 \to A$ und $f : A \to A$ existiert exakt ein Morphismus $c : \mathbb{N} \to A$ sodass das folgende Diagramm kommutiert:



1. Programmieren Sie in Haskell die Funktion c, welche q und f als Argumente erhält.
2. Wie müssen A, q und f in **Hask** aussehen, damit c die Funktion ist, welche eine Zahl n auf ihre unär kodierte Darstellung abbildet?
3. Die im obigen Diagramm enthaltene Information lässt sich in einer Kategorie mit Koprodukten auch anders anordnen. Übersetzen Sie das obige Diagramm in folgendes Schema und tragen die fehlenden Informationen nach.



Aufgabe 3: Programmieren mit Catamorphismen

Gegeben sei der folgende Datentyp für einfache arithmetische Ausdrücke.

```haskell
data ExprF e = Lit Int
             | Var String
             | Add e e
             | Mult e e
type Expression = Fix ExprF
```

zusammen mit

```haskell
type Env = [(String, Int)]
lookupEnv :: Env -> String -> Maybe Int
lookupEnv [] _ = Nothing
lookupEnv ((x,v):xs) s = if s == x
                         then Just v
                         else lookupEnv xs s
```

Geben Sie eine Funktor-Instanz für `ExprF` an und definieren Sie eine `eval`-, eine `pretty`- sowie eine `subst`-Funktion für `Expression` mithilfe von `cata :: (ExprF a -> a) -> Expression -> a`. Dabei wertet `eval :: Env -> Expression -> Maybe Int` einen Ausdruck aus, `pretty :: Expression -> Int` pretty-printet einen Ausdruck und `subst :: String -> Int -> Expression -> Expression` ersetzt innerhalb eines Ausdrucks alle Vorkommen einer Variable durch einen Wert.

Aufgabe 4: Bäume als F-Algebren

Der induktive Typ von Listen mit Elementen in `a` ist in Haskell durch folgende Definition gegeben:

```haskell
data ListF a r = Empty | Cons a r
type List a = Fix (ListF a)
```

Der Funktor `ListF` lässt sich formal auch schreiben als $F_a(r) = 1+(a\times r)$. Anstelle von `Fix (ListF a)` benutzt man die alternative Schreibweise $\mu r.F_a(r)$.

Geben Sie zwei Funktoren $B_a(r)$ und $K_a(r)$ an, sodass die initialen $B_a(r)$- und $K_a(r)$-Algebren gerade jeweils Binärbäumen mit blatt- beziehungsweise knotenorienterter Speicherung entsprechen.