5. Übungsblatt

Ausgabe: 21.01.10
Abgabe: 04.02.10

8 *Gutes Benehmen* *15 Punkte*

In diesem Übungsblatt soll eine einfache kleine DSL zur Raumschiffsteuerung implementiert werden. Diese Sprache basiert auf sogenannten Verhalten (*behaviours*), welche im wesentlichen über der Zeit Veränderliche sind (wobei Time die Zeit repräsentiert):

```
type Beh α = Time → (Maybe α)  -- (*)
```

Ein Verhalten b *terminiert*, wenn b t ≡Nothing; danach soll für alle s mit $s > t$ gelten, dass b s ≡Nothing. Für diese Verhalten gibt es verschiedene Funktionen:

```
lift1 :: (α→ β)→ Beh α→ Beh β
lift2 :: (α→ β→ γ)→ Beh α→ Beh β→ Beh γ
cond  :: Beh Bool→ Beh α→ Beh α→ Beh α
instance (Num α)⇒ Num (Beh α) where
  ...

seq   :: Beh α → Beh α→ Beh α        -- Sequentialisierung
par   :: Beh α → Beh β→ Beh (α, β)   -- Parallelkomposition
loop  :: Beh α → Beh α
while :: Beh (α, Bool)→ Beh α
```

Der Trick ist, dass wir die Typdefininition (∗) oben vergessen und Beh α als abstrakten Datentyp behandeln. Intern repräsentieren wir dann Verhalten als (potentiell unendliche) Listen, so dass insbesondere terminierende Verhalten auch irgendwann wieder abgeräumt werden können:

```
mk     :: (Time→ Maybe α)→ Beh α
sample :: Beh α→ [α]
```

Die Funktion mk, überführt eine "echte" Funktion punktweise in ein Verhalten, und sample gibt uns das Verhalten punktweise zu jedem Zeitpunkt ("tick"), welches im wesentlichen die interne Repräsentation ist.

Implementieren Sie das Verhalten in einem Modul Behaviour, welches die Typdefinition von Beh nicht exportiert.

Zum Testen unser Implementation von Beh implementieren wir

```
type Animation = Beh Graphic

animate :: Animation→ IO ()
```

welches Animationen (über der Zeit veränderliche Grafiken) darstellt. Beispiele für Animationen finden sich in Paul Hudak's Buch *The Haskell School of Expression*.

9 *Behaviour in Space* *5 Punkte*

Jetzt können wir die Steuerung des Raumschiffes aus dem 2. Übungsblatt durch Verhalten implementieren. Dazu werden die Datentypen Input, Ship und Control zu Verhalten geliftet:

```
type InputBeh   = Beh Input
type ShipBeh    = Beh Ship
type ControlBeh = Beh Control
```

und damit ein Kontrollverhalten implementiert:

```
control :: InputBeh → ShipBeh → ControlBeh
```

Implementieren Sie das regelbasierte Ausweichverfahren aus dem 2. Übungsblatt als Verhalten: wenn eine Kollision mit einem Asteroiden droht, dann soll das Verhalten eine Drehung in eine Richung einleiten, in der keine Kollision droht.