HASKELL-patterns

$$
\begin{array}{lcl}
\underline{\mathsf{pat}} & \rightarrow & \underline{\mathsf{var}} \\
 & | & \_ \\
 & | & \underline{\mathsf{integer}} \\
 & | & \underline{\mathsf{float}} \\
 & | & \underline{\mathsf{char}} \\
 & | & \underline{\mathsf{string}} \\
 & | & (\underline{\mathsf{constr}}\ \underline{\mathsf{pat}}_1\ \ldots\ \underline{\mathsf{pat}}_n) \\
 & | & \underline{\mathsf{var}}@\underline{\mathsf{pat}} \\
 & | & (\underline{\mathsf{var}} + \underline{\mathsf{integer}}) \\
 & | & [\underline{\mathsf{pat}}_1, \ldots, \underline{\mathsf{pat}}_n], \quad n \geq 0 \\
 & | & (\underline{\mathsf{pat}}_1, \ldots, \underline{\mathsf{pat}}_n), \quad n \geq 0
\end{array}
$$

HASKELL-types

$$
\begin{array}{lcll}
\underline{\mathsf{type}} & \rightarrow & (\underline{\mathsf{tyconstr}}\ \underline{\mathsf{type}}_1 \ldots \underline{\mathsf{type}}_n), & n \geq 0 \\
 & | & [\underline{\mathsf{type}}] & \\
 & | & (\underline{\mathsf{type}}_1 \rightarrow \underline{\mathsf{type}}_2) & \\
 & | & (\underline{\mathsf{type}}_1, \ldots, \underline{\mathsf{type}}_n), & n \geq 0 \\
 & | & \underline{\mathsf{var}} &
\end{array}
$$

$$
\underline{\mathsf{tyconstr}} \rightarrow \text{string starting with upper case symbol}
$$

Top declarations and type introduction

$$\underline{\textsf{topdecl}} \rightarrow \quad \underline{\textsf{decl}}$$

$$\mid\ \texttt{type}\ \underline{\textsf{tyconstr}}\ \underline{\textsf{var}}_1\ \ldots\ \underline{\textsf{var}}_n\ =\ \underline{\textsf{type}},\qquad n \geq 0$$

$$\mid\ \texttt{data}\ \underline{\textsf{tyconstr}}\ \underline{\textsf{var}}_1\ \ldots\ \underline{\textsf{var}}_n\ =$$

$$\underline{\textsf{constr}}_1\ \underline{\textsf{type}}_{1,1} \ldots \underline{\textsf{type}}_{1,n_1}\ \mid$$

$$\vdots$$

$$\underline{\textsf{constr}}_k\ \underline{\textsf{type}}_{k,1} \ldots \underline{\textsf{type}}_{k,n_k},$$

$$n \geq 0, k \geq 1, n_i \geq 0$$

Definition of simple data structures

```
data Color = Red | Yellow | Green

data MyBool = MyTrue | MyFalse


traffic_light :: Color -> Color
traffic_light Red    = Green
traffic_light Green  = Yellow
traffic_light Yellow = Red


und :: MyBool -> MyBool -> MyBool
und MyTrue y = y
und _      _ = MyFalse
```

Definition of natural numbers

```
data Nats = Zero | Succ Nats
```

```
plus :: Nats -> Nats -> Nats
plus Zero     y = y
plus (Succ x) y = Succ (plus x y)

half :: Nats -> Nats
half Zero            = Zero
half (Succ Zero)     = Zero
half (Succ (Succ x)) = Succ (half x)
```

Definition of lists

```
data List a = Nil | Cons a (List a)

len :: List a -> Nats
len Nil          = Zero
len (Cons x xs) = Succ (len xs)

append :: List a -> List a -> List a
append Nil          ys = ys
append (Cons x xs) ys = Cons x (append xs ys)
```