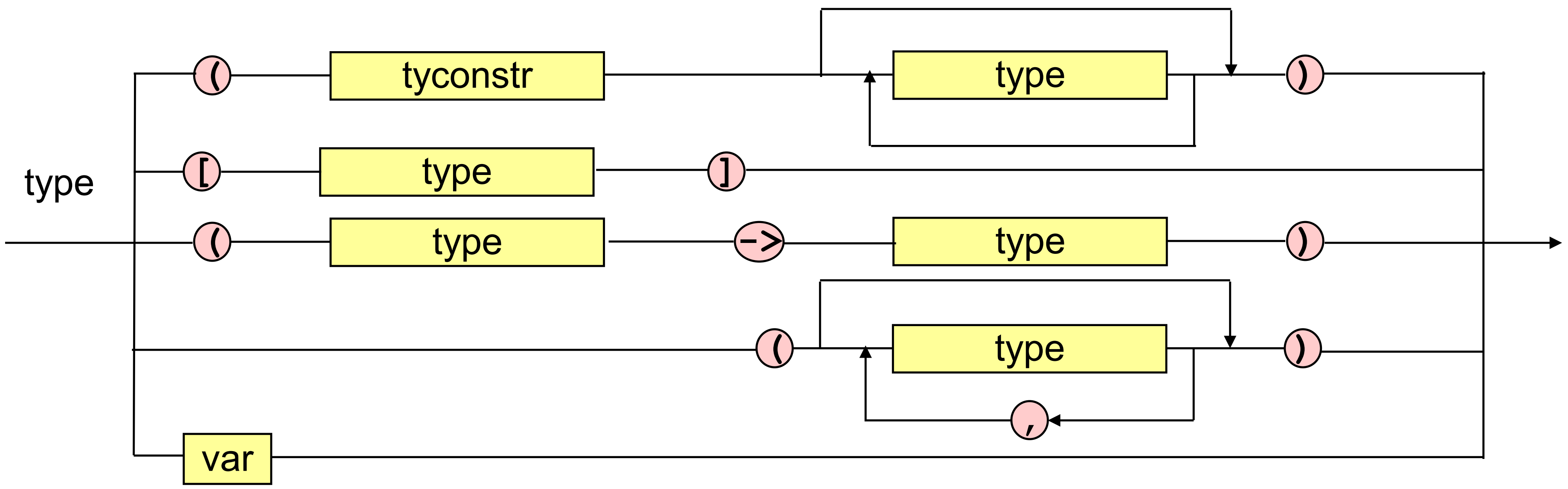
III. Funktionale Programmierung

Typen



Parametrische Polymorphie

```
len :: [Bool] -> Int
len []         = 0
len (x : xs) = 1 + len xs
```

```
len :: [Int] -> Int
len []         = 0
len (x : xs) = 1 + len xs
```

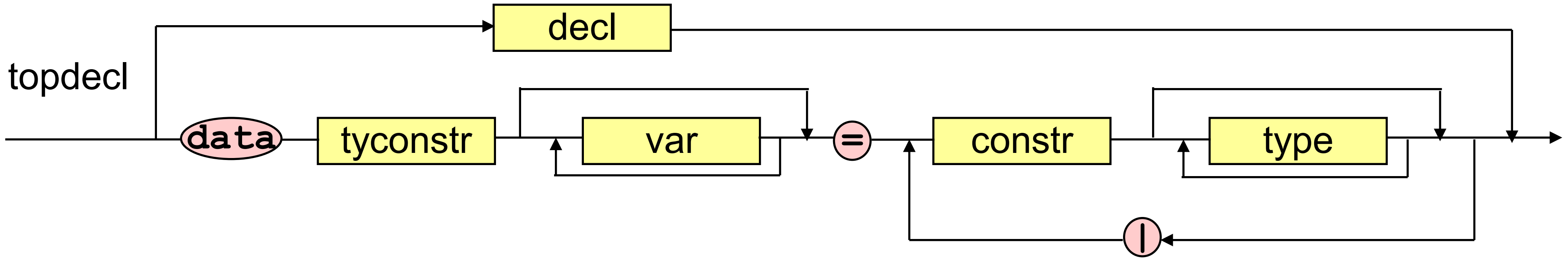
stattdessen

```
len :: [a] -> Int
len []         = 0
len (x : xs) = 1 + len xs
```

```
ident :: a -> a
ident x = x
```

```
app :: [a] -> [a] -> [a]
app []       ys  = ys
app (x : xs) ys  = x : app xs ys
```

Deklaration neuer Datentypen

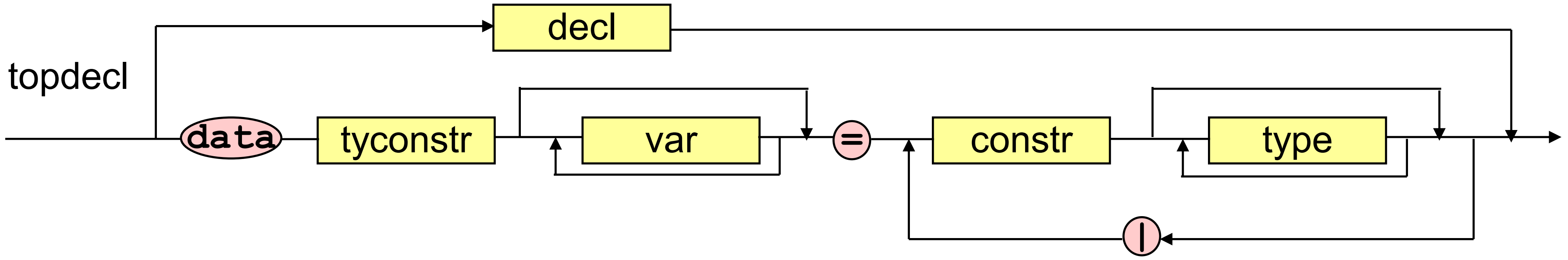


```
data Color = Red | Yellow | Green  deriving Show
data MyBool = Wahr | Falsch        deriving Show
```

```
ampel :: Color -> Color
ampel Red    = Green
ampel Green  = Yellow
ampel _      = Red
```

```
und :: MyBool -> MyBool -> MyBool
und Wahr  y  = y
und _     _  = Falsch
```

Deklaration neuer Datentypen

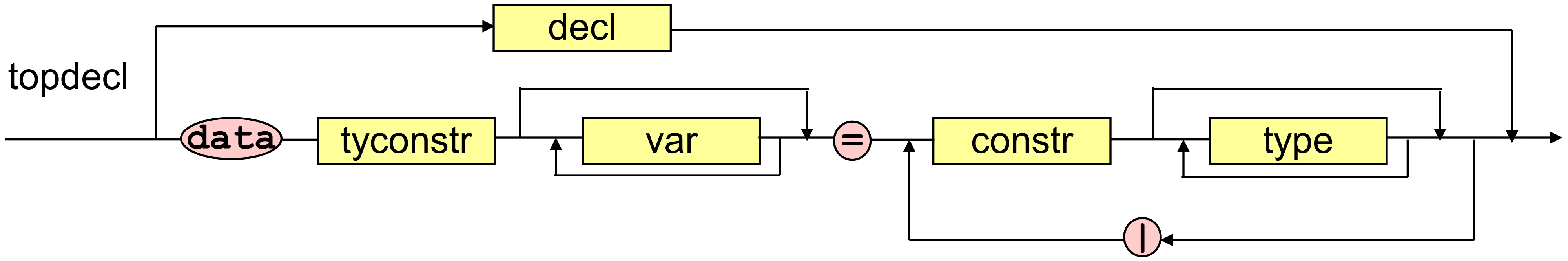


```
data Nats = Zero | Succ Nats  deriving Show
```

```
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)
```

Deklaration neuer Datentypen



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

```
len :: List a -> Int
len Nil           = 0
len (Cons x xs)   = 1 + len xs
```

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