III. Funktionale Programmierung

- 1. Prinzipien der funktionalen Programmierung
- 2. Deklarationen
- 3. Ausdrücke
- **4. Muster (Patterns)**
- 5. Typen und Datenstrukturen
- 6. Funktionale Programmiertechniken

Pattern Matching

```
und :: Bool -> Bool -> Bool
und True  y  = y
und   x   y  = False
```

```
Bool = "True" | "False"
```

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

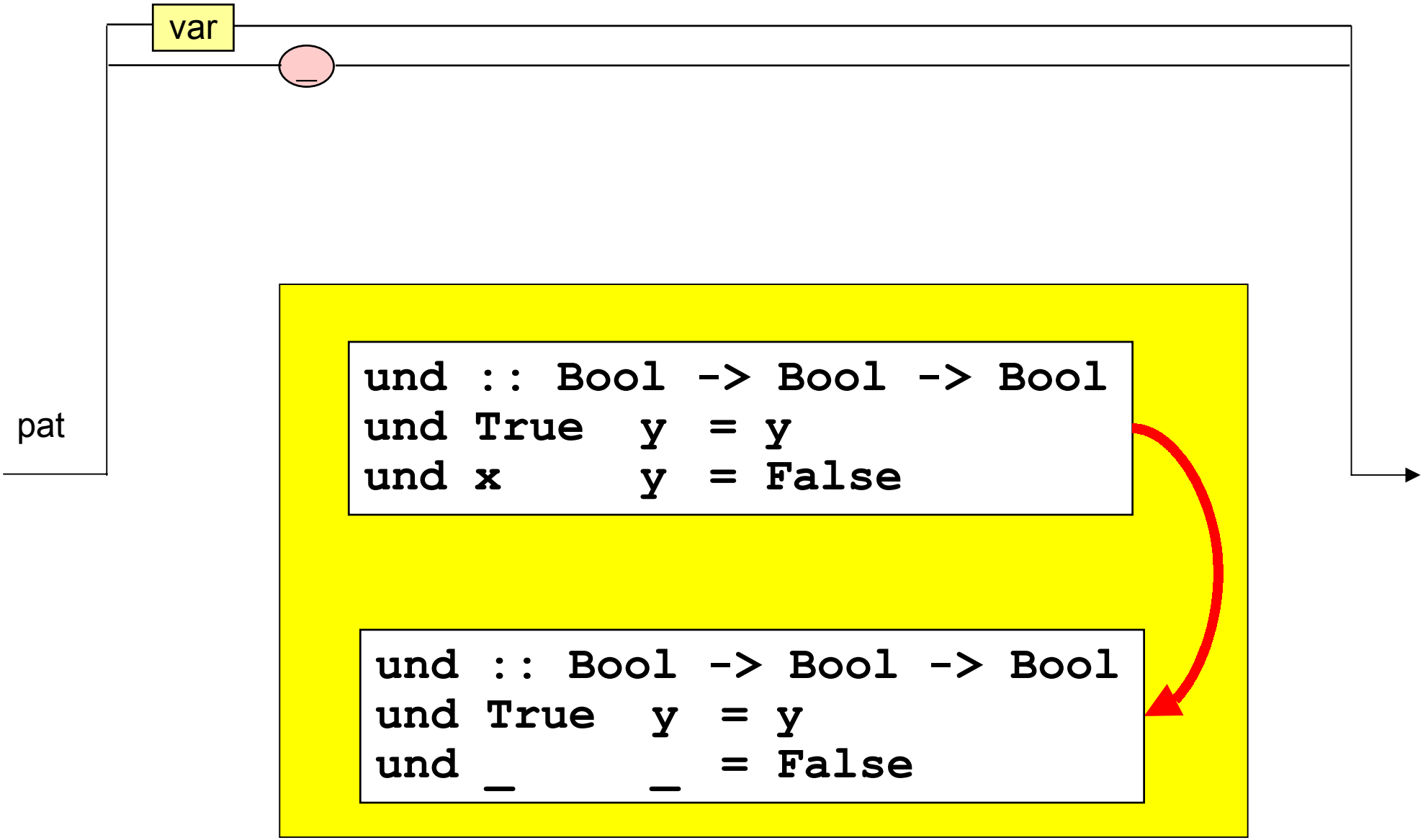
```
Liste = "[]" |
        Element ":" Liste
```

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

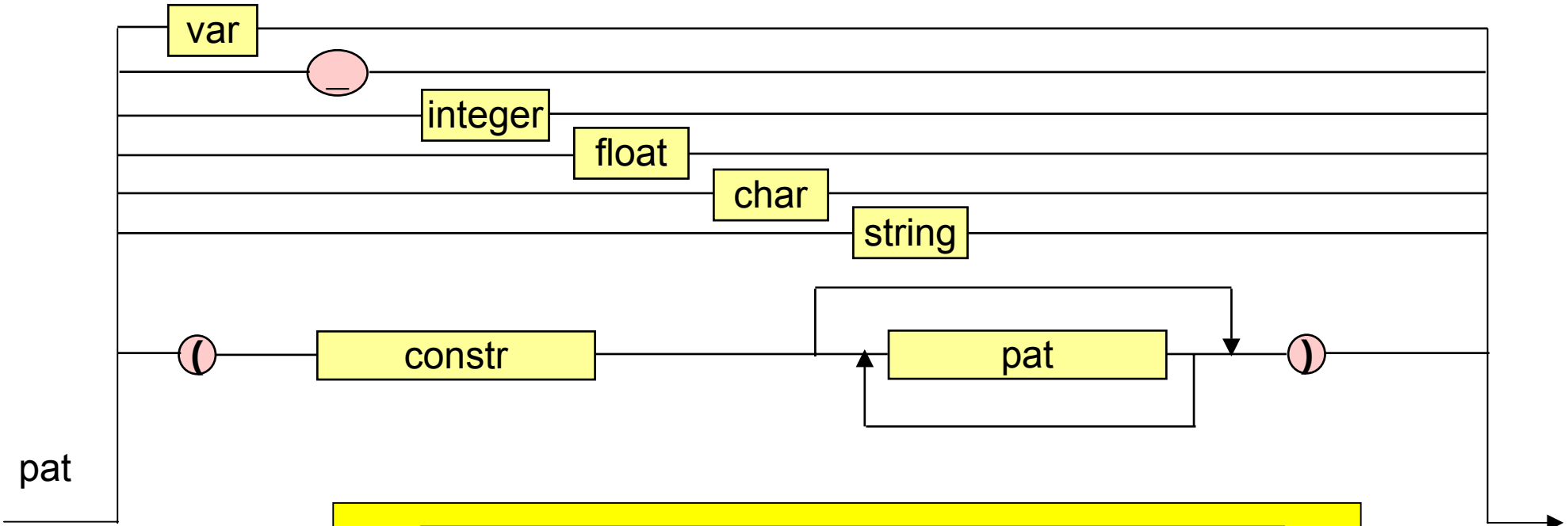
```
equal :: [Int] -> [Int] -> Bool
equal xs xs          = True
equal xs (x : xs)    = False
```

Nicht erlaubt!
Linke Seiten müssen linear sein

Muster (Patterns)



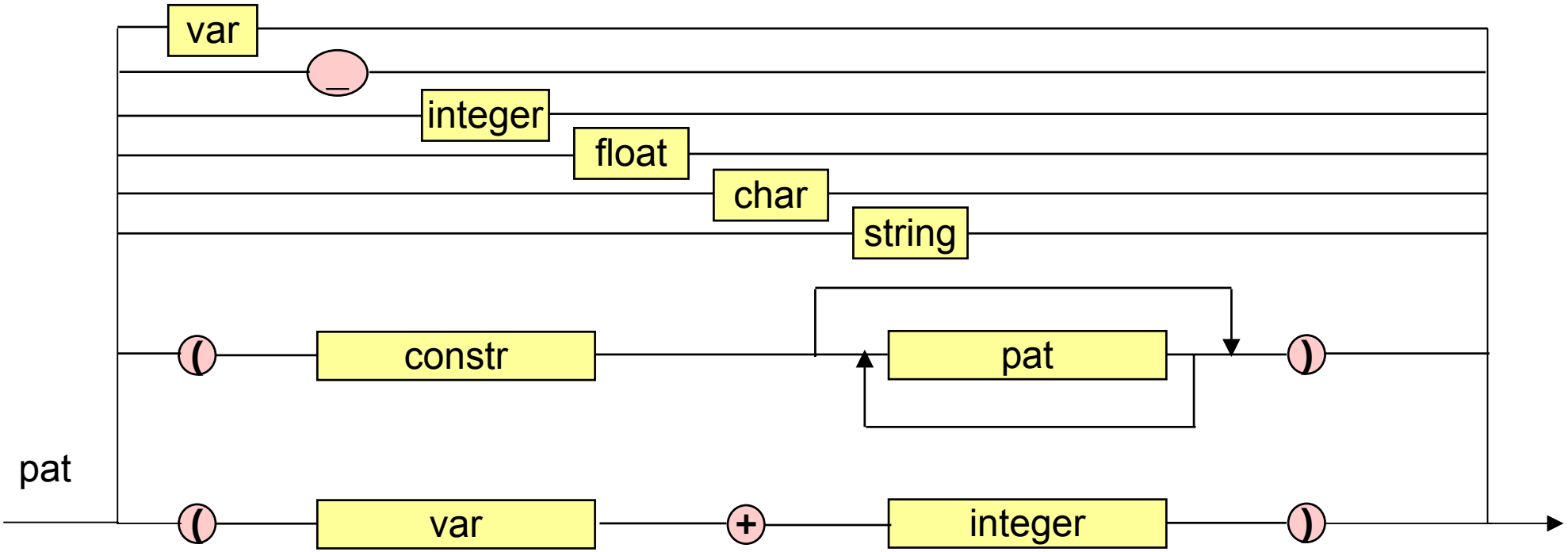
Muster (Patterns)




```
und :: Bool -> Bool -> Bool
und True  y  = y
und _     _  = False
```

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

Muster (Patterns)




```
fac :: Int -> Int
fac 0       = 1
fac (x + 1) = (x+1) * fac x
```

```
sub7 :: Int -> Int
sub7 (x + 7) = x
```

Muster (Patterns)

```
has_length_three :: [Int] -> Bool
has_length_three [x,y,z] = True
has_length_three _       = False
```

```
maxi :: (Int, Int) -> Int
maxi (0, y)       = y
maxi (x, 0)       = x
maxi (x+1, y+1)   = 1 + maxi (x, y)
```