III. Funktionale Programmierung

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
```

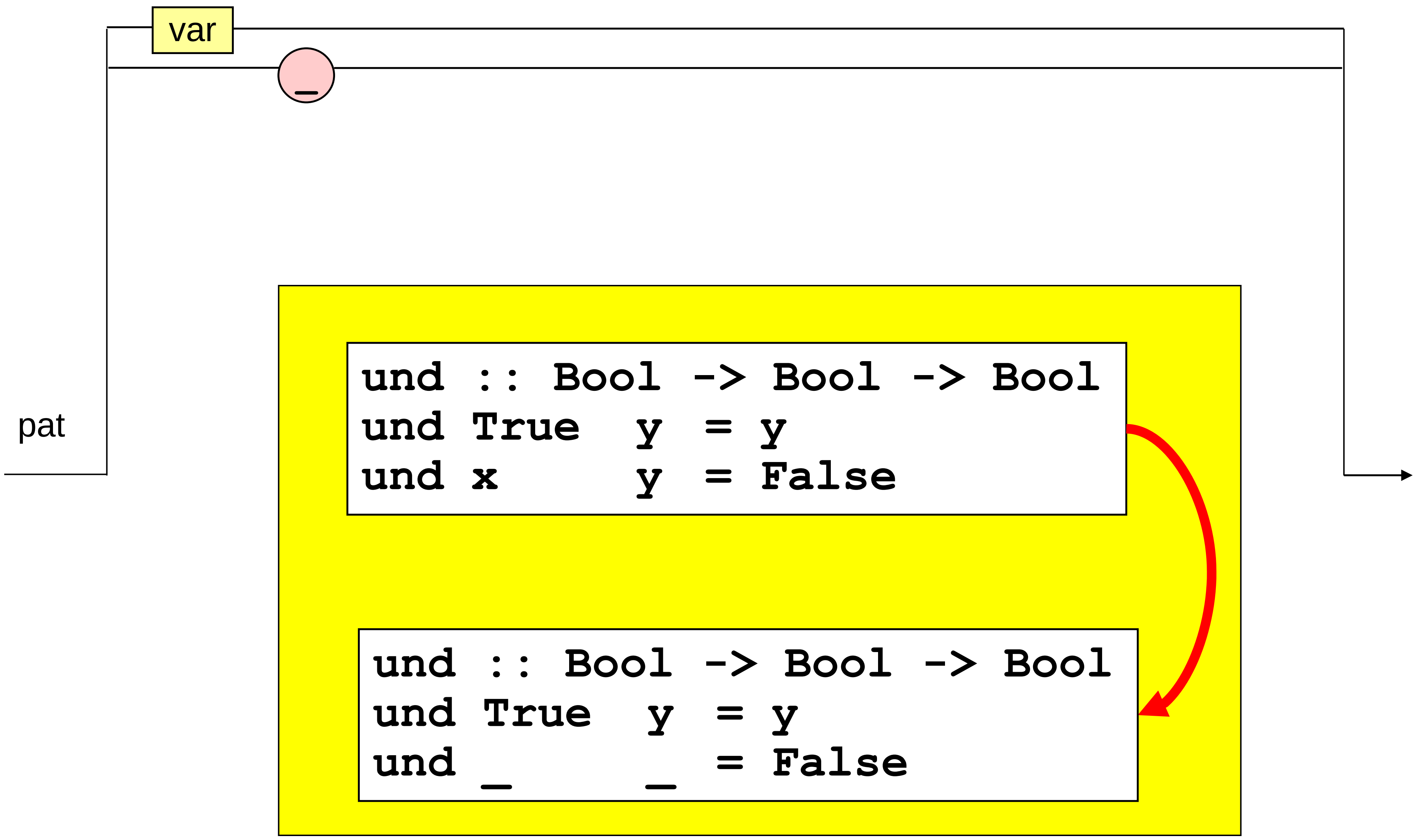
```
Liste = "[]" |
        Element ":" Liste
```

```
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

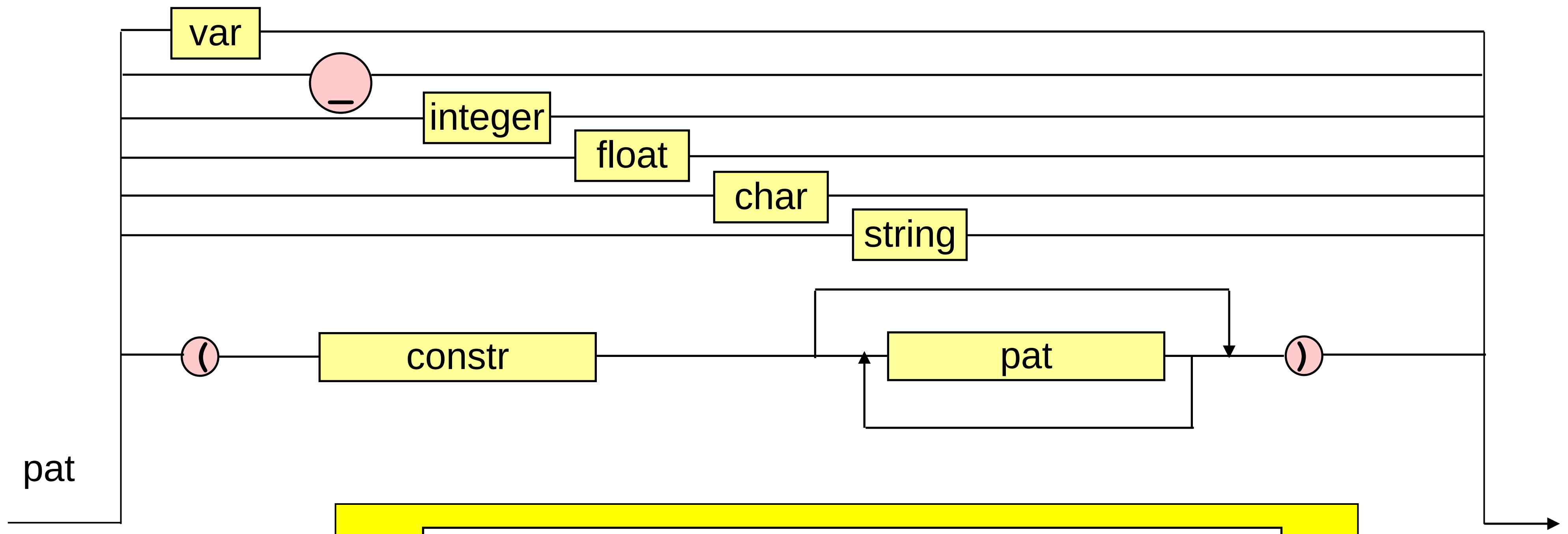
Muster (Patterns)



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

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

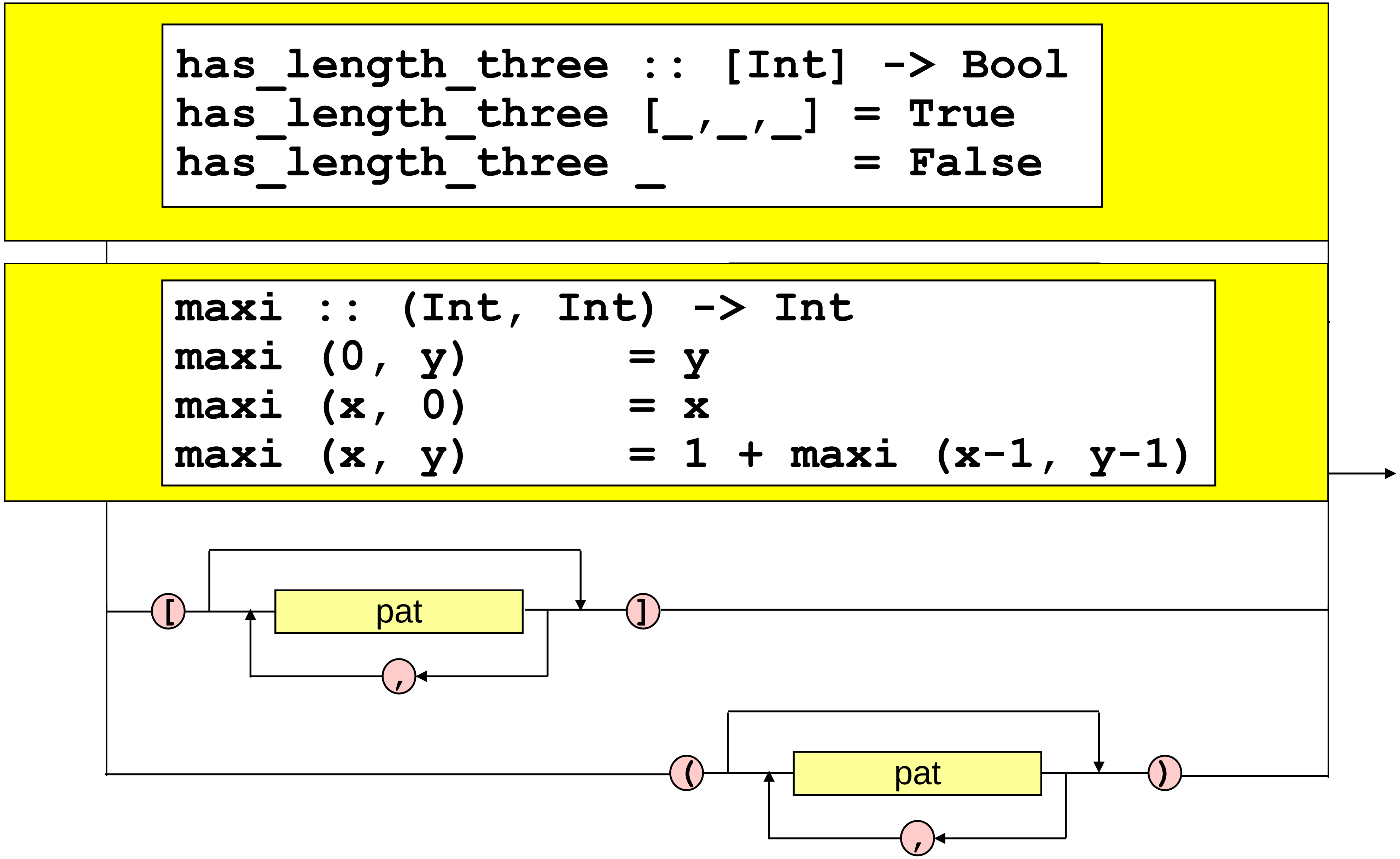
Muster (Patterns)



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

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

Muster (Patterns)



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

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