
III. Funktionale Programmierung

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

Deklarationen

```
len :: [Int] -> Int
len []           = 0
len (kopf : rest) = 1 + len rest

square :: Int -> Int
square x = x * x
```

Typdeklarationen

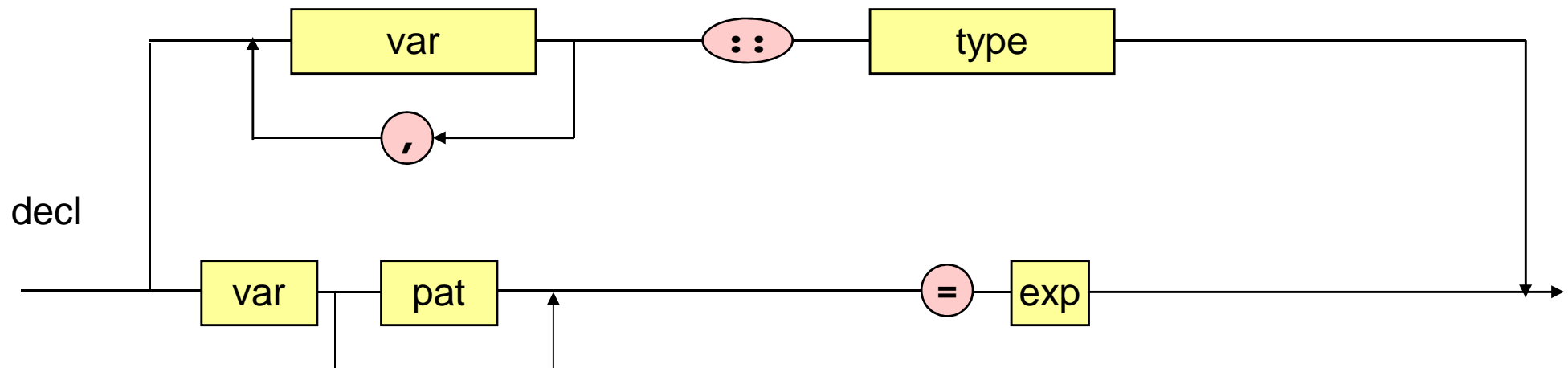
Funktionsdeklarationen

Programm in *Haskell*: Folge von linksbündig untereinander stehenden Deklarationen

Deklarationen

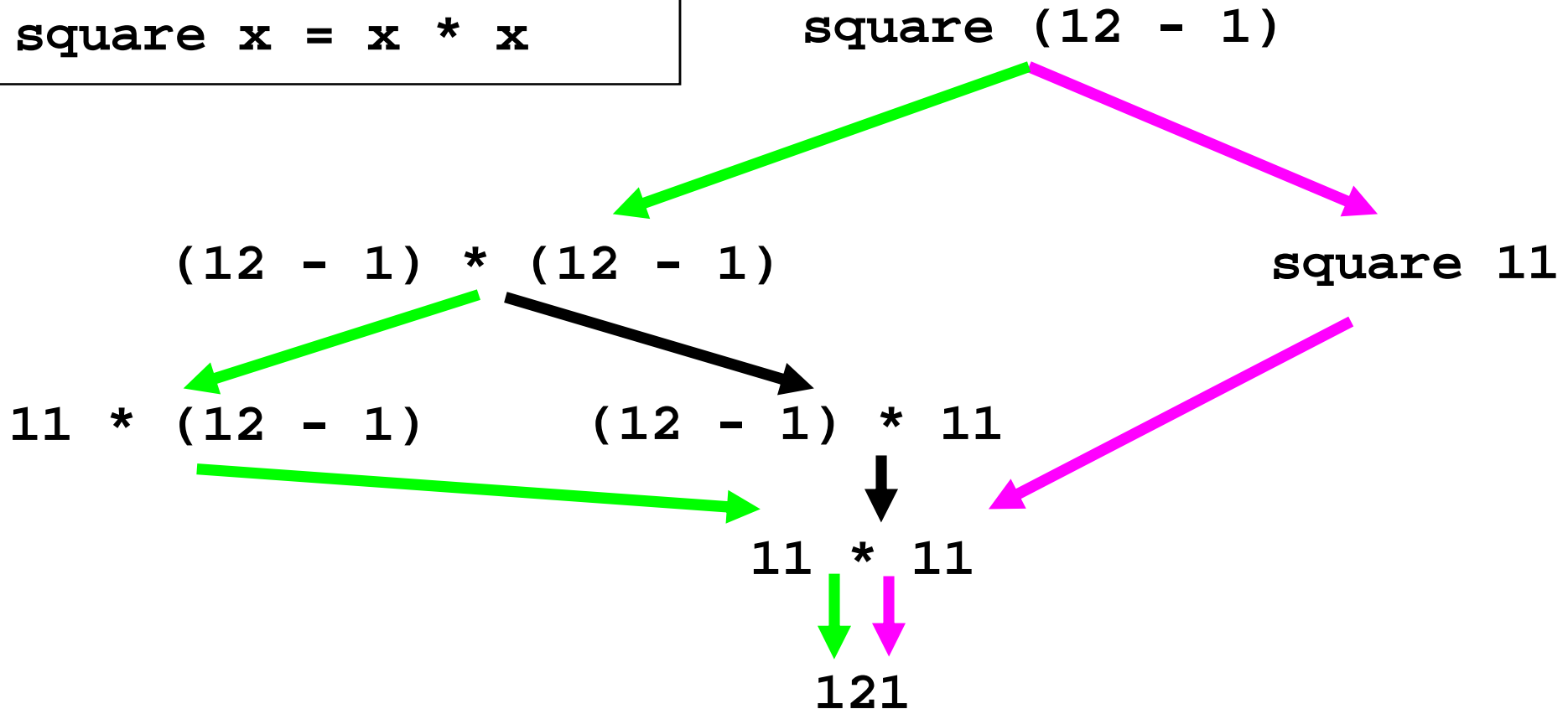
```
len :: [Int] -> Int
len []           = 0
len (kopf : rest) = 1 + len rest

square :: Int -> Int
square x = x * x
```



Auswertungsstrategie

```
square :: Int -> Int
square x = x * x
```

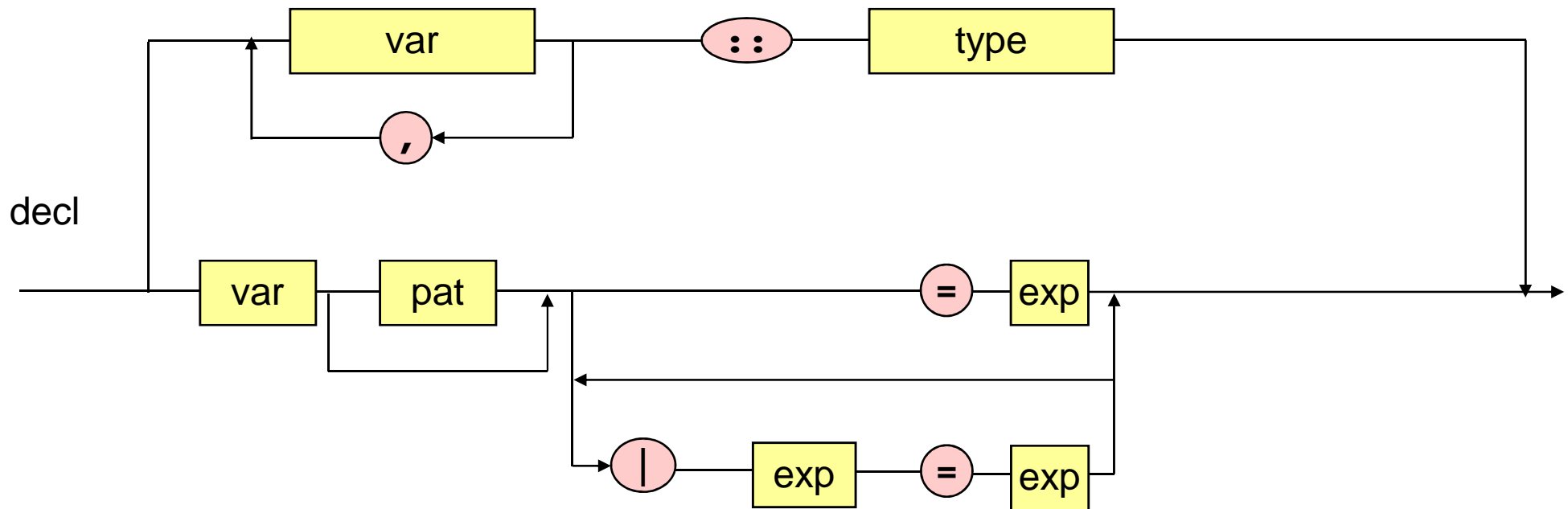


- strikte Auswertung (call-by-value), innen links
- nicht-strikte Auswertg. (call-by-name), außen links

Bedingte definierende Gleichungen

```
maxi :: (Int, Int) -> Int
```

```
maxi (x, y) | x >= y    = x  
            | otherwise = y
```

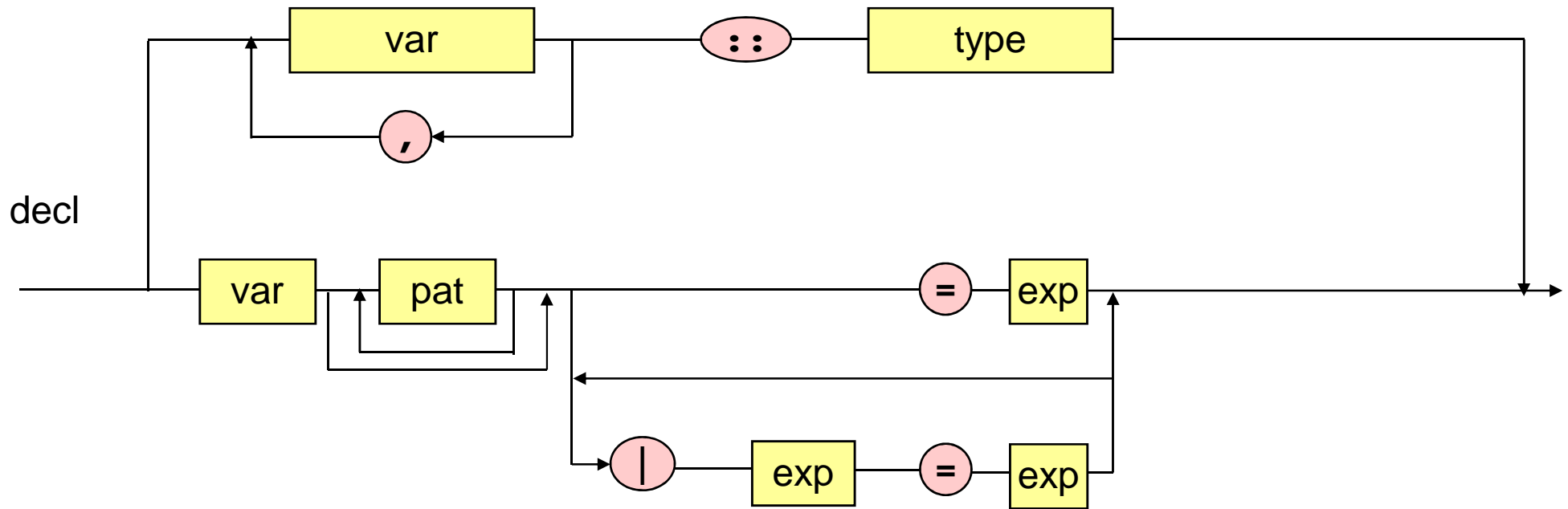


Currying

```
plus :: (Int, Int) -> Int  
plus (x, y) = x + y
```

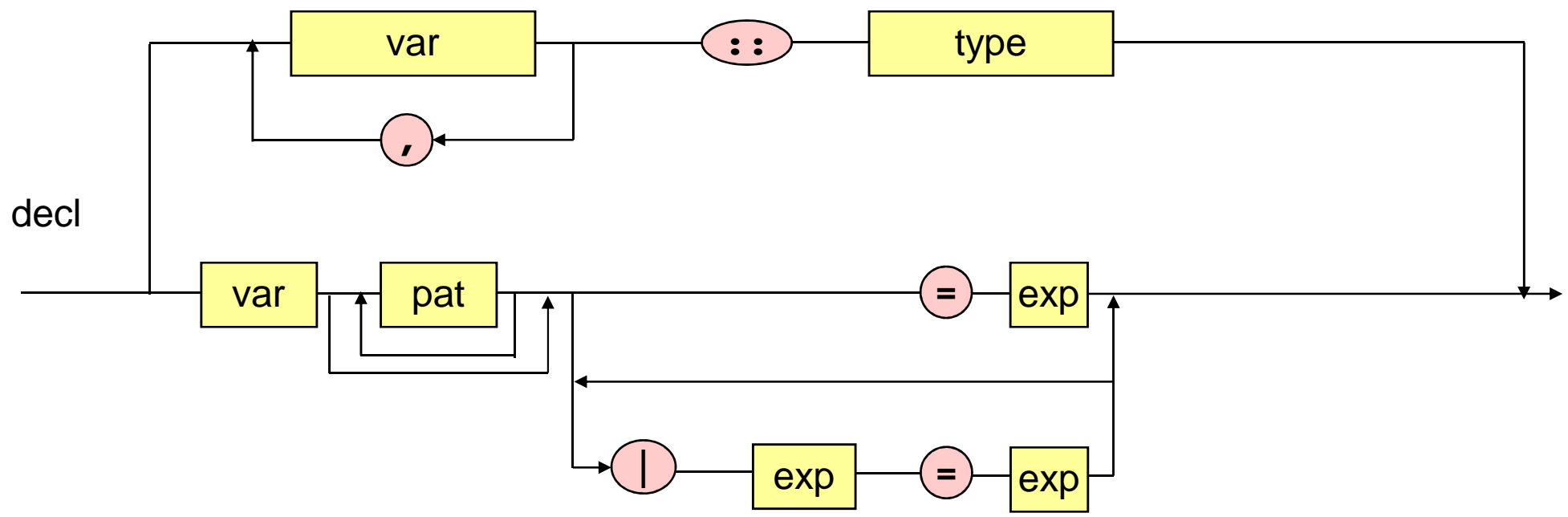
```
plus :: Int -> Int -> Int  
plus x y = x + y
```

Currying



Pattern Matching

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



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

Lokale Deklarationen

```
roots :: Float -> Float -> Float -> (Float, Float)
```

```
roots a b c = ((-b - d)/e, (-b + d)/e)
```

```
    where d = sqrt b*b - 4*a*c
```

```
          e = 2*a
```

