

Small step Semantik

Beispiel einer höheren Funktion ohne Rekursion: *twice*

Durch die unterschiedlichen Parameternamen wird die Semantik besser lesbar.

```
let val twice = λ f:int → int. λ x:int. f (f x)
    val square = λ y:int. y * y
in twice square 5
end
```

steht für

```
let val twice = λ f:int → int. λ x:int. f (f x)
in let val square = λ y:int. * (y, y)
    in twice square 5
    end
end
```

```
→ let val square = λ y:int. * (y, y)
    in (λ f:int → int. λ x:int. f (f x)) square 5
    end
```

mit Regel (LET-EXEC)

```
→ (λ f:int → int. λ x:int. f (f x)) (λ y:int. * (y, y)) 5
mit Regel (LET-EXEC)
```

```
→ (λ x:int. (λ y:int. * (y, y)) ((λ y:int. * (y, y)) x)) 5
mit Regel (BETA-V)
```

```
→ (λ y:int. * (y, y)) ((λ y:int. * (y, y)) 5)
mit Regel (BETA-V)
```

```
→ (λ y:int. * (y, y)) (* (5, 5))
mit Regel (BETA-V)
```

```
→ (λ y:int. * (y, y)) 25
mit Regel (OP)
```

```
→ * (25, 25)
mit Regel (BETA-V)
```

```
→ 625
mit Regel (OP)
```