

Small step Semantik

Beispiel einer höheren Funktion ohne Rekursion: *twice*

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

steht für

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

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

mit Regel (LET-EXEC)

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

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

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

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

```
→ (λx:int. * (x, x)) 25
mit Regel (OP)
```

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

```
→ 625
mit Regel (OP)
```