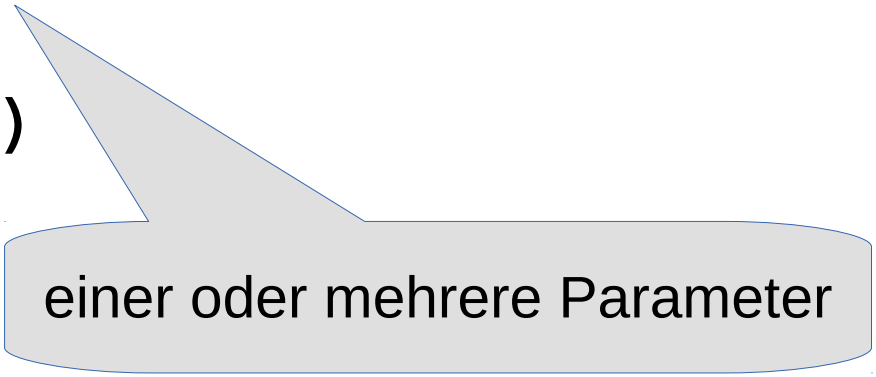
Verallgemeinerung
der Vorgehensweise
bei der Rekursion

# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

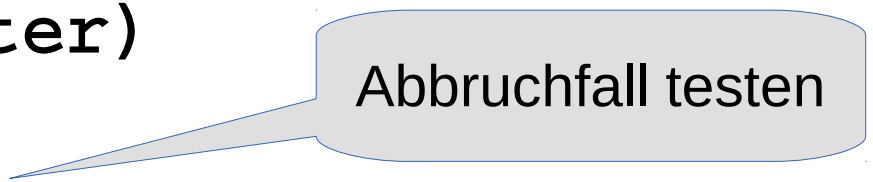
```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```



# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```



# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```

Abbruchwert definieren; sein Erreichen beendet die Rekursionsschritte in die Tiefe

*Anmerkung:*
*Der Abbruchwert kann ebenfalls durch eine Funktion (fkt Parameter) gegeben sein, hängt aber oft nur von dessen Typ ab. Siehe Beispiele.*

# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```

In der Regel erfolgt noch eine Bearbeitung ...

# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```

auf der Basis der aktuellen Parameterwerte ...

# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```

…
des beim Schritt in die Tiefe erzielten Wertes

# Rekursion mit Scheme

- Muster Rekursion allgemein.scm

```
(define
  (rekursive-Fkt Parameter)
  (if
    (Abbruch? Parameter)
    Abbruchwert
    (Bearbeitung
      Parameter
      (rekursive-Fkt (update Parameter))
    )
  )
)
```

Die konkrete Problemlösung hängt von der Bearbeitungsfunktion und der konkreten update-Fkt ab.

# Rekursion mit Scheme

Warum update?

Bearbeitung mit unveränderten Parameterwerten bedeutet bei funktionaler Programmierung

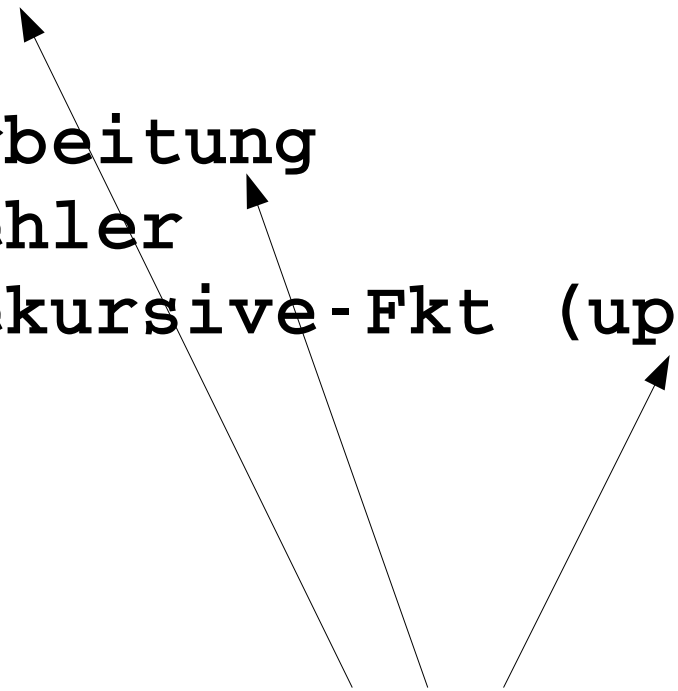
- identische Bearbeitung,
- identische Werte und
- führt zu einer Endlosschleife

Die update-Funktion muss prinzipiell auf die Abbruchbedingung hin führen

# Rekursion mit Scheme

- Beispiel: Rekursion über Zahlen

```
(define
  (rekursive-Fkt zaehler)
  (if
    (abbruch? zaehler)
    1
    (Bearbeitung
      zaehler
      (rekursive-Fkt (update zaehler))
    )
  )
)
```



*mit den konkreten Funktionen ...*

# Rekursion mit Scheme

- Beispiel: Rekursion über Zahlen

```
(define
  (abbruch? zaehler)
  (< zaehler 1))

(define
  (Bearbeitung zaehler rekursionswert)
  (* zaehler rekursionswert))

(define
  (update zaehler)
  (- zaehler 1)
  )
```

*… erhält man hier die Fakultäts-Funktion.*

# Rekursion mit Scheme

- Beispiel: vereinfachte Variante

```
(define
  (fakultaet zaehler)
  (if
    (< zaehler 1)
    1
    (* zaehler
       (fakultaet (- zaehler 1))
     )
  )
)
```

# Rekursion mit Scheme

- Beispiel: Rekursion über Zahlen

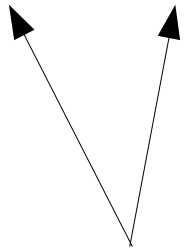
```
(define
  (Abbruch? zaehler endwert)
  (> zaehler endwert))

(define
  (Bearbeitung zaehler Rekursionswert)
  (+ zaehler Rekursionswert))

(define
  (update zaehler endwert)
  (list
   (+ zaehler 2)
   endwert))
```

*Und mit diesen Funktionen?*

# Rekursion mit Scheme

- Beispiel: Rekursion über Zahlen

```
(define
  (rekursive-Fkt zaehler endwert)
  (if
    (Abbruch? zaehler endwert)
    0                          modifizierter Abbruchwert
    (Bearbeitung
      zaehler
      (apply rekursive-Fkt
        (update zaehler endwert))
    )
  )
)
```



*aufwändig wegen der zwei Parameter.*

# Rekursion mit Scheme

Die Anwendung von

apply

ist in der Regel nicht notwendig

- Mehrere Parameter werden üblicherweise durch je eine update-Funktion verändert.
- Dazu ein Beispiel für Listen

# Rekursion mit Scheme

- Beispiel: Rekursion über Listen

```
(define
  (rekursive-Fkt element liste)
  (if
   (null? liste)
   (list element)
   (Bearbeitung
    element
    liste
    (rekursive-Fkt
     (update-1 element)
     (update-2 liste))
    )
   )
  )
```

Abbruchfall Liste leer

*prinzipiell zwei update-Funktionen*

# Rekursion mit Scheme

- Beispiel: Rekursion über Listen

```
(define
  (Bearbeitung element liste rekursionswert)
  (cons (first liste) rekursionswert))
  )
```

*benötigt hier* **element** *nicht*

```
(define
  (update-1 element)
  element)
```

**element** *bleibt hier unverändert*

```
(define
  (update-2 liste)
  (rest liste)
  )
```

# Rekursion mit Scheme

Nicht für jeden Parameter ist eine update-Funktion notwendig, die ihn verändert.

Umgekehrt gilt:

- Bei mehreren Parametern ist für mindestens einen eine update-Funktion notwendig, die seinen Wert verändert.
- Zum Beispiel die vereinfachte Version:

# Rekursion mit Scheme

- Beispiel: Rekursion über Listen

```
(define
  (rekursive-Fkt element liste)
  (if
   (null? liste)
   (list element)
   (Bearbeitung
    liste
    (rekursive-Fkt
     element
     (update liste))
    )
   )
  )
```

# Rekursion mit Scheme

- Beispiel: Rekursion über Listen

```
(define
  (Bearbeitung liste rekursionswert)
  (cons (first liste) rekursionswert))
  )

(define
  (update liste)
  (rest liste)
  )
```

# Rekursion mit Scheme

- Endversion

```
(define
  (anhaengen element liste)
  (if
   (null? liste)
   (list element)
   (cons
    (first liste)
    (anhaengen element (rest liste))
    )
   )
  )
```