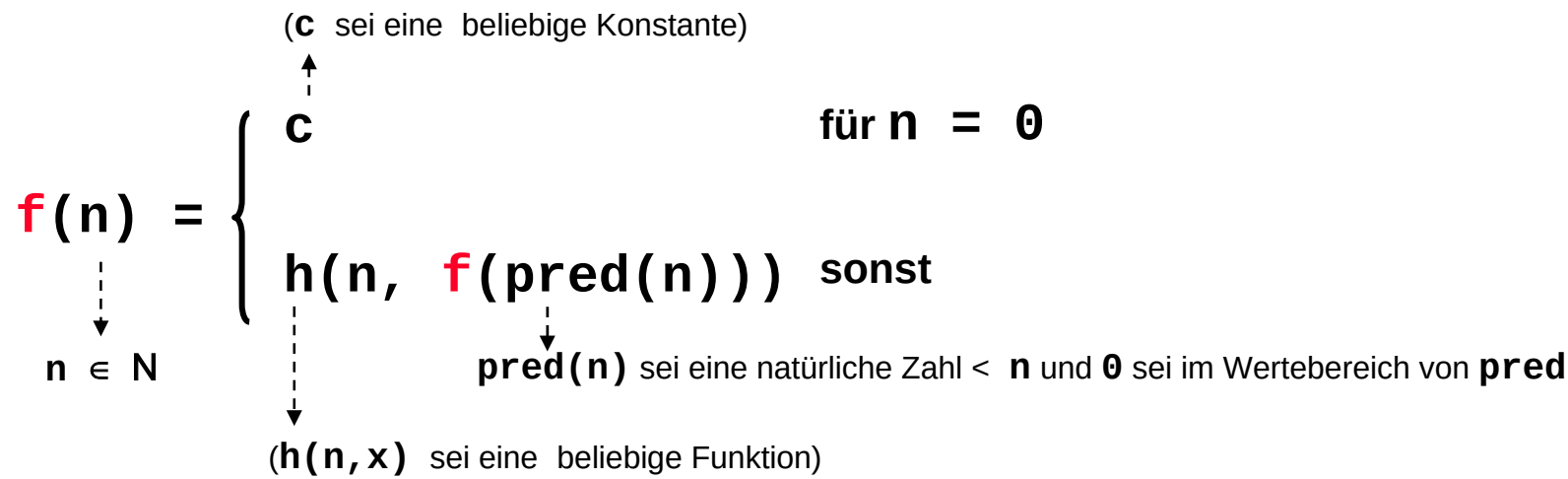
Grundlagen der Programmierung

Vorlesung 10

Sebastian Iwanowski
FH Wedel

Primitiv rekursive Funktionen




```
procedure f(n: Integer): Integer
if (n=0)
  then
    return c
  else
    return h(n, f((pred(n)))
end {f}
```

Vorteil: **Terminierung ist immer gewährleistet**

Endrekursive Funktionen

(**g(x)** sei eine beliebige Funktion)

$$f(x) = \begin{cases} g(x) & \text{für ein logisches Prädikat } P(x) \\ f(r(x)) & \text{sonst} \end{cases}$$

r(x) sei eine beliebige Funktion, solange der Wertebereich im Definitionsbereich für f ist

x beliebig

(**x** kann auch ein mehrdimensionaler Vektor sein !)

```
procedure f(x): ResultType
if P(x)
  then
    return g(x)
  else
    return f(r(x))
end {f}
```

Vorteil: **Es gibt Algorithmus zur automatischen Implementierung auf dem Computer**

Linear rekursive Funktionen

$$f(x) = \begin{cases} g(x) & \text{für ein logisches Prädikat } P(x) \\ h(x, f(r(x))) & \text{sonst} \end{cases}$$

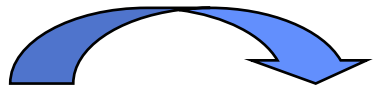
```
procedure f(x): ResultType
if P(x)
  then
    return g(x)
  else
    return h(x, f(r(x)))
end {f}
```

Primitiv rekursive Funktionen ⇘

Linear rekursive Funktionen

Endrekursive Funktionen ⇗

Transformationen



Linear rekursive Funktion

Endrekursive Funktion

geht z.B. unter den Voraussetzungen:

i) $\forall x,y,z: h(x, h(y,z)) = h(h(x,y), z)$

ii) $\exists e: \forall x: h(e, x) = x$

```
procedure f(x): ResultType
if P(x)
  then
    return g(x)
  else
    return h(x,f(r(x)))
end {f}
```

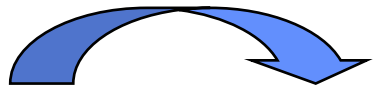
zweidimensionaler Vektor (x,a)

```
procedure fAux(x,a): ResultType
if P(x)
  then
    return h(a,g(x))
  else
    return fAux(r(x),h(a,x)))
end {fAux}
```

„rAux“: Funktion von (x,a)

```
procedure f(x): ResultType
  return fAux(x,e)
end {f}
```

Transformationen



Endrekursive Funktion **Schleife**

geht immer !

```
procedure f(x): ResultType
if P(x)
  then
    return g(x)
  else
    return f(r(x))
end {f}
```

```
procedure f(x): ResultType
while ¬P(x)do
  x := r(x);
return g(x)
end {f}
```

Allgemeine rekursive Funktionen

Fibonacci-Funktion:

$$f(n) = \begin{cases} 1 & \text{für } n \leq 1 \\ f(n-2) + f(n-1) & \text{sonst} \end{cases}$$

McCarthy-Funktion:

$$f(n) = \begin{cases} n-10 & \text{für } n > 100 \\ f(f(n+11)) & \text{sonst} \end{cases}$$

Ulam-Collatz-Funktion:

$$f(n) = \begin{cases} 1 & \text{für } n = 1 \\ f(n/2) & \text{für gerade } n \\ f(3n+1) & \text{für ungerade } n \end{cases}$$

Allgemeine rekursive Funktionen

- **Nicht jede rekursive Funktion ist linear rekursiv oder endrekursiv**
- **Alle rekursiven Funktionen können auch nichtrekursiv formuliert werden.**

Was ist besser: Rekursive oder iterative Formulierung ?

Zusammenfassung: Programmverifikation

Zuweisungen

- Schwächste Vorbedingung und stärkste Nachbedingung kann immer ausgerechnet werden.
- Beispiele: Vorlesungen 5 und 6
- Zusammenfassung: Vorlesung 6 (Anfang)
- Übungsaufgaben: Blatt 5

Verzweigungen

- Schwächste Vorbedingung und stärkste Nachbedingung kann immer ausgerechnet werden.
- Beispiele und Zusammenfassung:Vorlesung 6
- Übungsaufgaben: Blatt 6

Zusammenfassung: Programmverifikation

Schleifen

Aufgaben:

- Zu gegebener Vorbedingung die Funktionsweise berechnen **und beweisen** (d.h. eine Nachbedingung angeben).
- Zu gegebener Vorbedingung und Nachbedingung beweisen, dass die Schleife die Vorbedingung in die Nachbedingung überführt.
- Zu gegebener Nachbedingung eine Vorbedingung (möglichst die schwächste) suchen und beweisen, dass die Schleife die Vorbedingung in die Nachbedingung überführt.

Alternative Beweistechniken:

- Vollständige Induktion
- Beweis mit Invariantenbedingung und Variantenzahl

Zusammenfassung: Programmverifikation

Schleifen

Beweistechnik Vollständige Induktion

- häufig Induktion über die Anzahl der durchgeführten Schleifendurchläufe
- Korrektheit und Terminierung werden meistens zugleich bewiesen
  und hängen voneinander ab.
- Häufig muss ein schärferer Sachverhalt bewiesen werden als verlangt, um den Induktionsschritt korrekt durchführen zu können.
  (bewiesen werden muss meistens nur ein Analogon zur Invariantenbedingung, gebraucht wird aber auch ein Analogon zur Variantenzahl)

Beweistechnik Invariantenbedingung und Variantenzahl

- Zeige, dass aus Schleifenbedingung und Invariantenbedingung die Invariantenbedingung folgt.
- Zeige, dass die Variantenzahl sich so verändert, dass irgendwann einmal die Schleifenbedingung verletzt wird.

Zusammenfassung: Programmverifikation

Rekursion

Aufgaben:

- Zu gegebener Vorbedingung die Funktionsweise berechnen **und beweisen** (d.h. eine Nachbedingung angeben).
- Zu gegebener Vorbedingung und Nachbedingung beweisen, dass die Rekursion die Vorbedingung in die Nachbedingung überführt.
- Zu gegebener Nachbedingung eine Vorbedingung (möglichst die schwächste) suchen und beweisen, dass die Rekursion die Vorbedingung in die Nachbedingung überführt.

Beweistechnik: Vollständige Induktion

- Die Induktion kann häufig direkt über einen Eingabeparameter geführt werden.
- Induktionsverankerung entspricht dem nichtrekursiven Zweig der Fallunterscheidung
- Induktionsschluss entspricht dem rekursiven Zweig der Fallunterscheidung

Beim nächsten Mal:

Entwurf und Analyse von Algorithmen