Grundlagen der Theoretischen Informatik

Sebastian Iwanowski
FH Wedel

Kap. 2: Verifikationstechniken
Teil 4: Rekursive Prozeduren und Funktionen

Rekursion

oder: Wie programmiert man wirklich elegant ?

```
procedure f (n: Integer): Integer
begin {f}
  if (n=0)
    then
      return 1
    else
      return n • f(n-1)
end {f}
```

Was berechnet diese Prozedur ?

Gibt es irgendwelche Vorbedingungen ?

Wie beweist man das alles ?

Verifikation von Rekursiven Prozeduren

Wesentliche Schritte bei der Verifikation von Rekursiven Prozeduren:

1) **Beweise, dass die Berechnungen der Prozedur kontinuierlich auf dem richtigen Weg sind,**
   sodass <u>nach Abbruch</u> der Rekursion das Richtige berechnet ist.

   (falls die Rekursion jemals abbricht)

   Irgendetwas muss unverändert richtig sein → Invariante (Bedingung)

2) **Beweise, dass die Rekursion abbricht.**

   Irgendetwas muss sich im Laufe der Rekursion ändern → Variante (Zahl)

Wichtigste Beweistechnik: Vollständige Induktion

Verifikation von Rekursiven Prozeduren

Verifikation von Rekursiven Prozeduren mit vollständiger Induktion:

Es muss in Induktionsverankerung und im Induktionsschluss von i auf i+1 bewiesen werden, dass die Rekursion immer abbricht <u>und</u> das Richtige berechnet.

Die Induktionsbehauptung muss also sowohl eine Invariante als auch eine Variante im Sinne der vorigen Definition enthalten.

Typische Beispiele für die Wahl der Induktionsvariablen i:

1) i = Parameter im rekursiven Aufruf (oder Modifikationen davon)
2) i = Anzahl der *noch auszuführenden* Rekursionsschritte

Typische Beweistechnik im Induktionsschluss:

1) Die Abhängigkeit des Funktionswertes von i+1 zum Funktionswert von i ergibt sich *direkt aus der Rekursionsgleichung*.
2) Finde variante Zahl, die für Abbruch der Rekursion sorgt (Tipp: Diese kann in der Regel als Induktionsvariable gewählt werden).

Primitiv rekursive Funktionen

(**c** sei eine beliebige Konstante)

$$\mathbf{f(n)} = \begin{cases} \mathbf{c} & \text{für } \mathbf{n} = 0 \\ \mathbf{h(n,\ f(pred(n)))} & \text{sonst} \end{cases}$$

$\mathbf{n} \in N$

pred(n) sei eine natürliche Zahl < **n**

(**h(n,x)** sei eine beliebige Funktion, die nicht f benutzt)

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

weil Integers auch < 0 sein können

Vorteil: **Terminierung ist immer gewährleistet**

Endrekursive Funktionen

(g(x) sei eine beliebige Funktion, die nicht f benutzt)

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

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

x beliebig

(x kann auch ein mehrdimensionaler Vektor sein !)

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

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

Transformation

Endrekursive Funktion

Schleife

geht immer !

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

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

Linear rekursive Funktionen

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

x beliebig

h(x,y) hat zwei Parameter:
x hat den Datentyp der Eingabe von f
y hat den Datentyp der Ausgabe von f

(x kann auch ein mehrdimensionaler Vektor sein !)

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

Linear rekursive Funktionen können in Spezialfällen in endrekursive umgewandelt werden.

Primitiv rekursive Funktionen $\Rightarrow$

Linear rekursive Funktionen

Endrekursive Funktionen $\Rightarrow$

Allgemeine rekursive Funktionen

In den folgenden Beispielen sei n $\in \mathbb{N}$

Fibonacci-Funktion:

$$f(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ f(n-2) + f(n-1) & \text{sonst} \end{cases}$$

McCarthy-Funktion:

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