Grundlagen der Programmierung

Vorlesung 9

Sebastian Iwanowski
FH Wedel

Modularisierung

Prozeduren mit Rückgabewert (Funktionen)

```
in := LiesEingabedatum ();
n := LiesTageszahl ();
out := BerechneAusgabedatum (in, n);
GibAusgabedatumAus (out);
```

kein Rückgabewert ----→

in der Prozedur (Funktion):

```
BerechneAusgabedatum (inDate, count);
begin
      .
      .
      .
      return outDate;
end
```

im aufrufenden Programm:

```
out := BerechneAusgabedatum (in, n);
```

- Die Prozedur darf in der return-Anweisung einen beliebigen Wertausdruck haben.
- Dieser wird bei der Ausführung der return-Anweisung ausgewertet
- Nach der return-Anweisung wird die Prozedur beendet und der Rückgabewert im Hauptprogramm an der entsprechenden Stelle eingesetzt.

Modularisierung

Variablen in Prozeduren

- Da Prozeduren alle Funktionalitäten von Programmen haben, darf man in ihnen auch Variablen definieren.
  (falls die Programmiersprache das erlaubt)
- Variablen sollten grundsätzlich nur lokal verwendet werden, auch wenn die Programmiersprache mehr erlaubt (wie Pascal)
- Die Kommunikation zwischen zwei verschiedenen Programmteilen (Prozeduren) hat in der Regel über Parameter zu erfolgen (und nicht durch Benutzung gemeinsamer Variablen)
- Daher behandeln wir hier keine Sichtbarkeitsregeln.

Unterscheide Parametervariablen von anderen lokalen Variablen !

Modularisierung

Verifikation bei Prozeduren

- Die Verifikation innerhalb von Prozeduren unterscheidet sich nicht von der Verifikation allgemeiner Programme.

Einziger neuer Punkt der Beachtung: **Parameterübergabe**

- Die aktuellen Parameter müssen die Vorbedingungen der entsprechenden formalen Parameter erfüllen.

Die meisten Compiler unterstützen das durch Typprüfungen.

- Der Rückgabewert muss die Vorbedingungen für die zugewiesene Variable des Hauptprogramms erfüllen.

Analoges gilt bei der Benutzung von Variablenparametern.

Grundlagen der Programmierung

1. Einführung
   - Grundlegende Eigenschaften von Algorithmen und Programmen
2. Logik
   - Aussagenlogik
   - Prädikatenlogik
3. Programmentwicklung und –verifikation
   - Grundlagen der Programmverifikation
   - Zuweisungen und Verbundanweisungen
   - Verzweigungen
   - Schleifen
   - Modularisierung
   - ➔ Rekursion
4. Entwurf und Analyse von Algorithmen
   - Klassifikation von Algorithmen
   - Programmierung von Algorithmen
   - Bewertung von Algorithmen

Rekursion

oder: Wie programmiert man wirklich elegant ?

```
procedure f (n: Integer): Integer
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 ?

Rekursion

Verifikation von rekursiven Prozeduren

Rekursive Prozeduren haben viel mit Schleifen gemeinsam

- Daher bietet sich eine ähnliche Verifikationsstechnik an:

1) **Beweise, dass die Berechnungen in jedem Durchlauf der Prozedur so fortschreiten, dass nach Abbruch der Rekursion das Richtige berechnet ist.**

   (falls die Rekursion irgendwann einmal abbricht)

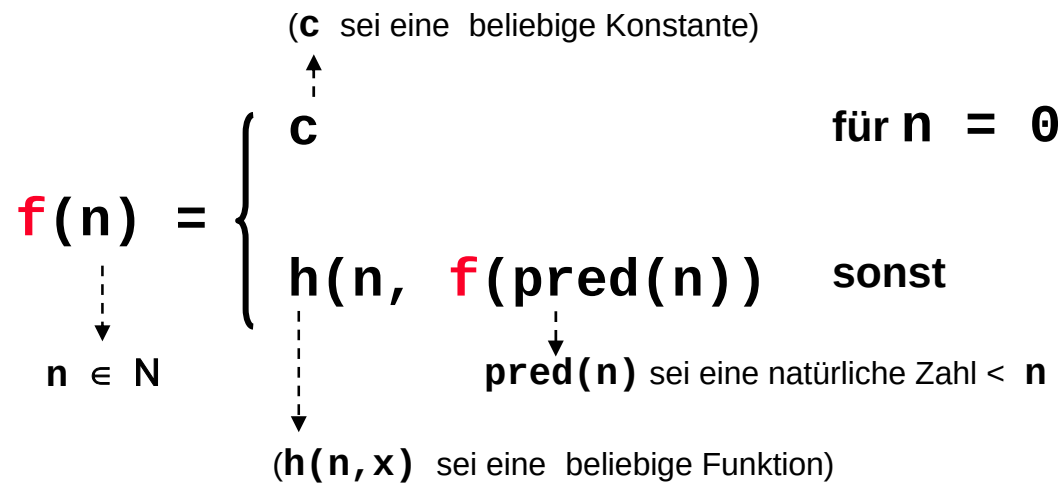
   Mögliches Beweiselement: ***Invariantenbedingung***

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

   Essentielle Beweiselemente: ***Variante und Terminierungsbedingung***

Wichtigste Beweistechnik: Vollständige Induktion

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 ⇗

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 ?

Beim nächsten Mal:

Transformationen zwischen den verschiedenen Rekursionsarten

Zusammenfassung und Übungen: Programmentwicklung und -verifikation