

Zentralübung zur Informatik I

Dr. Markus Schneider
Technische Universität München

Wintersemester 2000/2001

4. Dezember 2000

Überblick

- Allgemeine Bemerkungen zum Übungsbetrieb
- Bemerkungen zur Programmieraufgabe von Übungsblatt 5
- Beispiele zur rekursiven Funktionsauswertung
- Semantische Äquivalenz und Termersetzung
 - Semantische Äquivalenz
 - Vergleich von Text- und Termersetzung
 - partielle Korrektheit
 - Instanziierung und Termersetzung

Allgemeine Bemerkungen

- Kein Übungsbetrieb am 7. November:
 - Am 7. November fallen die Übungen anlässlich des Dies Academicus aus. Die Teilnehmer der Donnerstagsübungen werden gebeten sich auf die Freitagsgruppen zu verteilen!
- Hausaufgaben auf den Übungsblättern:
 - Die Übungsblätter stellen Ihnen häufig zu jedem Themengebiet mehrere Aufgaben zur Verfügung. Mit den Tutoren werden die einzelnen **Aufgabentypen** besprochen. Die verbleibenden Aufgaben können von Ihnen als Hausaufgabe bearbeitet werden. Die Musterlösungen enthalten die Lösungen **aller** Aufgaben!
 - Hausaufgaben werden zukünftig als solche gekennzeichnet!

Programmieraufgabe von Übungsblatt 5

- Informatiker ist eine Unterklasse von Absolvent: Der Informatiker erbt somit sämtliche Attribute des Absolventen. Dies sind:

int alter

int semesterzahl

.....

- Zur Verfügung gestellte Operationen:
 - ∧ boolean and (boolean a, boolean b);
 - ∨ boolean or (boolean a, boolean b);
 - ¬ boolean not (boolean a);

.....

- Beispiel: Es soll überprüft werden, ob der Informatiker entweder mehr als 15 Semester studiert hat oder ob er älter als 29 Jahre ist.
 - Teilaussagen:
 - A : hat mehr als 15 Semester studiert
 - B : ist älter als 29
 - Formalisierter Term: $(\neg A \wedge B) \vee (A \wedge \neg B)$

Lösung des Beispiels:

- **Vorgehensweise:**

- Schreibe zunächst die beiden Funktionen, die den Wahrheitswert der booleschen Aussagen A bzw. B ermitteln!
- Schreibe anschließend eine Funktion, die obige boolesche Formel auswertet!

- **Implementierung in Java:**

```
boolean A () {  
    return ((semesterzahl > 15) ? true : false);  
}
```

```
boolean B () {  
    return ((alter > 29) ? true : false);  
}
```

```
boolean pruefe() {  
    return or(and(not(A()), B() ), and(A(), not(B()) ) );  
}
```

Rekursive Funktionen

- Beispiel: Berechnung der Fakultät der ersten n natürlichen Zahlen.

```
int fakultaet (int n) {  
    return n==0 ? 1 : fakultaet(n-1) * n;  
}
```
- **Definition:** Eine Funktion bzw. Funktionsdeklaration heißt **rekursiv**, wenn der Ausdruck im Rumpf der Funktion einen Aufruf derselben Funktion enthält.
- **Praktischer Hinweis:** Eine rekursive Funktionsdefinition enthält stets einen einfach zu behandelnden Spezialfall (In obigem Beispiel $n==0$) und einen oder mehrere Funktionsaufrufe derselben Funktion mit 'einfacheren' Argumenten.
- **Auswertung:** In Java erfolgt der Funktionsaufruf durch **Call-by-value**
- Die Auswertung ist strikt
- Die formalen Parameter im Funktionsrumpf werden ersetzt durch die **Werte** der aktuellen Parameter (die Argumente)

1. Beispiel: Fakultätsfunktion

```
int fakultaet (int n) {  
    return n==0 ? 1 : fakultaet(n-1) * n;  
}
```

- Gesucht:

fakultaet(4)
(fakultaet(3) * 4)
((fakultaet(2) * 3) * 4)
(((fakultaet(1) * 2) * 3) * 4)
((((fakultaet(0) * 1) * 2) * 3) * 4)
((((1 * 1) * 2) * 3) * 4)
(((1 * 2) * 3) * 4)
((2 * 3) * 4)
(6 * 4)
24

2. *Beispiel: Listenoperationen*

- Gegeben sei ein String `s` beliebiger Länge; die Elemente des Strings sind vom Typ `char`. Beispiel: `abba`
- Für den Datentyp `String` seien folgende Funktionen definiert:
 - `rest: String → String`; Gibt einen String zurück, dem das erste Element fehlt.
 - `first: String → char`; Gibt das erste Element der Liste zurück
 - `isEmpty: String → boolean`; Ermittelt, ob die Liste leer ist oder nicht
- Problemstellung: Es soll eine rekursive Funktion definiert werden, die ermittelt, wieviele der Zeichen ``a`` der String enthält!
- Lösung (in Java-Notation):

```
int zaehle_a (String s) {  
    return isEmpty(s) ? 0  
        : first(s) == `a` ? zaehle_a(rest(s)) + 1  
        : zaehle_a(rest(s)) ;  
}
```

Anwendung: `zaehle_a(`abba`)`

```
int zaehle_a (String s) {  
    return isEmpty(s) ? 0  
        : first(s) == `a` ? zaehle_a(rest(s)) + 1  
        : zaehle_a(rest(s)) ; }
```

`zaehle_a(`abba`)`

`zaehle_a(rest(`abba`)) + 1`

`zaehle_a(`bba`) + 1`

`zaehle_a(rest(`bba`)) + 1`

`zaehle_a(`ba`) + 1`

`zaehle_a(rest(`ba`)) + 1`

`zaehle_a(`a`) + 1`

`zaehle_a(rest(`a`)) + 1 + 1`

`zaehle_a(``) + 2`

`0 + 2`

`2`

Semantische Äquivalenz, Term- und Textersetzung

- Semantische Äquivalenz boolescher Terme: Zwei boolesche Terme t_1 und t_2 heißen semantisch äquivalent, wenn sie für alle möglichen Belegungen β bezüglich der Interpretation I_β gleich sind:

- $I_\beta [t_1] = I_\beta [t_2]$

- Beispiel: $x \wedge (x \vee y) \Leftrightarrow x$. Bei semantischer Äquivalenz muß gelten:

$$I_\beta [x \wedge (x \vee y)] = I_\beta [x].$$

$$I_\beta [x] \wedge (I_\beta [x] \vee I_\beta [y]) = I_\beta [x]$$

Mit der Wahrheitstabelle folgt nun die semantische Äquivalenz

$I_\beta [x] = \beta(x)$	$I_\beta [y] = \beta(y)$	$I_\beta [x] \wedge (I_\beta [x] \vee I_\beta [y])$
O	O	O
O	L	O
L	O	L
L	L	L

- Termersetzungssystem:
 - Gewisse Teilterme werden gemäß den Booleschen Gesetzen durch andere Terme ersetzt.
 - Der Term hat nach dieser Umformung dieselbe Bedeutung, die Terme sind semantisch äquivalent.
- Textersetzungssystem:
 - Beispiel: Semi-Thue-Systemen des letzten Übungsblattes;
Die Regel $II \rightarrow \varepsilon$ führt die Strichzahl III in die Strichzahl II über.
Die beiden Ausdrücke sind **nicht äquivalent** bezüglich der verwendeten Interpretation

Rechenstruktur der natürlichen Zahlen

- Signatur: $\Sigma = \{\text{zero}, \text{succ}, \text{pred}, \text{add}, \text{mult}, \text{div}\}$
- Konkrete Funktionen: $\{+, -, *, /\}$
- Trägermenge: \mathbb{N}
- Regeln (Auszug): $R = \{\text{add}(\text{succ}(x), y) \rightarrow \text{succ}(\text{add}(x, y)),$
 $\text{mult}(x, \text{succ}(y)) \rightarrow \text{add}(x, \text{mult}(x, y)) \dots\}$
- Interpretation:
 - $I[\text{succ}(\text{zero})] = 1$
 - $I[\text{pred}(\text{zero})] = \perp$
 - $I[\text{succ}(x)] = (I[x] + 1)$
 - $I[\text{pred}(x)] = (I[x] - 1)$, falls $I[x] > 0$
 - $I[\text{pred}(x)] = \perp$, falls $I[x] = 0$
 - $I[\text{add}(x, y)] = (I[x] + I[y])$
 - $I[\text{mult}(x, y)] = I[x] * I[y]$
 - $I[\text{div}(x, y)] = (I[x]/I[y])$, falls $I[y] > 0$
 - $I[\text{div}(x, y)] = \perp$, falls $I[y] = 0$

Partielle Korrektheit

- Definition: Eine Termersetzungsregel $t \rightarrow r$ heißt partiell korrekt in einer Rechenstruktur A , falls für jede Belegung β in A gilt:

$$I_\beta [t] = I_\beta [r]$$

d.h. die Terme t und r sind semantisch äquivalent.

- Beispiel: Die Regel $\text{add}(\text{succ}(x),y) \rightarrow \text{succ}(\text{add}(x,y))$ ist partiell korrekt.

$$\begin{aligned} I[\text{add}(\text{succ}(x),y)] & \\ &= (I[\text{succ}(x)] + I[y]) \\ &= ((I[x] + 1) + I[y]) \\ &= ((I[x] + I[y]) + 1) \\ &= (I[\text{add}(x, y)] + 1) \\ &= I[\text{succ}(\text{add}(x, y))] \end{aligned}$$

Instanziierung und Termersetzung

- Problem: Es soll der Term

$\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero}))$

auf Normalform gebracht werden;

d.h. er soll nur noch **succ** und **zero** enthalten.

- 1. Regelanwendung: $\text{add}(\text{succ}(x), y) \rightarrow \text{succ}(\text{add}(x, y))$

Instanziierung:

$\text{add}(\text{succ}(x), y)[\text{zero}/x, \text{succ}(\text{zero})/y] \rightarrow \text{succ}(\text{add}(x, y))[\text{zero}/x, \text{succ}(\text{zero})/y]$

ergibt:

$\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})) \rightarrow \text{succ}(\text{add}(\text{zero}, \text{succ}(\text{zero}))) \quad (*)$

- 2. Regelanwendung: $\text{add}(\text{zero}, y) \rightarrow y$

Instanziierung:

$\text{add}(\text{zero}, y)[\text{succ}(\text{zero})/y] \rightarrow y[\text{succ}(\text{zero})/y]$

ergibt:

$\text{add}(\text{zero}, \text{succ}(\text{zero})) \rightarrow \text{succ}(\text{zero})$

nach Einsetzen dieser Instanz in (*) folgt der Term in Normalform:

$\text{succ}(\text{add}(\text{zero}, \text{succ}(\text{zero}))) \rightarrow \text{succ}(\text{succ}(\text{zero}))$