



Übungsblatt 10 – Lösungsvorschläge

21.12.2001

Aufgabe 1 While-Sprache

(Bierwald)

*Eine Maschine ist nicht dann perfekt wenn man nichts mehr hinzufügen kann,
sondern erst wenn es nichts mehr gibt, das man weglassen könnte.*

ANTOINE DE SAINT-EXUPERY

Eine Maschine ist die While-Sprache (ohne Erweiterungen) nicht. Aber ist sie perfekt?

Hinweis. Zeigen Sie, dass man auf die Vorgaenger-Funktion verzichten kann und nur Zuweisungen der Form "X becomes 0" oder "X becomes Nachf(X)" benoetigt.

(i) Zuweisungen $X := Y$ können wir durch

```
begin
  X := 0;
  while X ≠ Y do
    X := Nachf(X)
  od
end
```

ersetzen.

(ii) Zuweisungen $X := \text{Nachf}(Y)$ ersetzen wir zunächst durch

```
begin
  X := Y;
  X := Nachf(X)
end
```

und anschließend beseitigen wir die Zuweisung $X := Y$ wie in (i).

(iii) Das folgende erweiterte While-Programm berechnet die Vorgaenger-Funktion.

```
begin
  X1Alt := X1;
  if X1Alt ≠ 0 then
    X1 := 0;
    X1Nachf := 1;
    while X1Nachf ≠ X1Alt do
      X1Nachf := Nachf(X1Nachf);
      X1 := Nachf(X1)
    od
  fi
end
```

Wir führen das Programm in ein nicht erweitertes While-Programm über:

```

begin
  X1Alt := 0;
  while X1Alt ≠ X1 do X1Alt := nachf(X1Alt) od;
  X1 := 0;
  X1Nachf := 0; X1Nachf := nachf(X1Nachf);
  while X1Nachf ≠ X1Alt do
    X1Nachf := nachf(X1Nachf);
    X1 := nachf(X1)
  od;
  X1Alt := 0;
end

```

Streng genommen, haben wir hier noch immer kein While-Programm, da die Variablen – aus Gründen der Lesbarkeit – nicht von der Form $X\langle\text{Zahl}\rangle$ sind. Ersetzt man jedes Vorkommnis von $X1Alt$ durch $X2$, $Xnull$ durch $X3$ und $X1Nachf$ durch $X4$, so hat das Programm schließlich die gewünschte Form.

Aufgabe 2 Berechenbare Funktionen (Bierwald)

Sei m Ihre Matrikelnummer. Ist die Funktion $f(x) = \begin{cases} 1, & \text{falls } \varphi_m(m) \downarrow \\ 0, & \text{sonst} \end{cases}$ berechenbar?

Sie ist es.

Entweder ist sie konstant 1 oder konstant 0, und mithin in beiden Fällen – egal, welcher es nun tatsächlich ist – berechenbar.

Aufgabe 3 While-Programme (Bierwald)

Zeigen Sie, daß kein While-Programm (ohne Erweiterungen) mit genau einer Variablen die Funktion $f(x) = 2x$ berechnen kann.

Wir nehmen an, daß es ein While-Programm \mathcal{P} gibt, daß f berechnet. Die einzige Variable sei o.B.d.A $X1$.

Wenn eine Whileschleife vorkommt, hat sie die Gestalt `while $X1 \neq X1$ do ... od`. Offensichtlich wird diese nie betreten. Wir können also alle Whileschleifen aus dem Programm entfernen, ohne daß sich an der berechneten Funktion etwas ändert.

Streichen wir nun sämtliche inneren `begin`'s und `end`'s ändert sich ebenfalls nichts an der berechneten Funktion, und es bleibt ein Programm übrig, daß nur aus einer Folge von Zuweisungen der Form `$X1:=0$` , `$X1:=nachf(X1)$` oder `$X1:=vorg(X1)$` besteht.

Würde eine Anweisung $X1:=0$ im Programm vorkommen, so könnte der berechnete Funktionswert nicht mehr von der Eingabe abhängen und insbesondere die Funktion f nicht berechnet werden.

Ohne Änderung der berechneten Funktion können wir also \mathcal{P} durch ein Programm ersetzen, das nur noch Zuweisungen $X1:=nachf(X1)$ oder $X1:=vorg(X1)$ enthält. In diesem Programm streichen wir alle Anweisungsfolgen $X1:=nachf(X1); X1:=vorg(X1)$ bis sich am Programm nichts mehr ändert. Auch dieses Programm berechnet noch die ursprüngliche Funktion, nämlich f , und hat die Form $\text{begin } [X1:=vorg(X1);]^m [X1:=nachf(X1);]^n \text{ end}$. Dabei sind n und m natürliche Zahlen.

Jetzt betrachten wir zwei geschickt gewählte Eingaben:

$X1 = 0$: Das Programm terminiert mit $X1 = n$. Da das Programm die Funktion f berechnet, muss $X1 = 2 \cdot 0 = 0$ gelten und somit $n = 0$ sein.

$X1 = 1$: Das Programm terminiert mit $X1 = 1 \div m \in \{0, 1\}$. Da das Programm die Funktion f berechnet, muss $X1 = 2 \cdot 1 = 2$ gelten. Widerspruch, da $X1 \in \{0, 1\}$.

Die Annahme, daß es ein Programm mit nur einer Variablen gibt, daß f berechnet muss also falsch sein.

Aufgabe 4 Gödelisierungen

(Bierwald, Käuffl)

a) Ein fauler Mensch sagt, man könne die Gödelisierung dadurch vereinfachen, daß man jedes Zeichen der While-Sprache als Ziffer eines geeigneten Zahlensystems auffaßt, wodurch jedes While-Programm zu seiner eigenen Gödelzahl wird. Ein anderer meint, man könne die Zahl der Zeichen, aus denen das Programm besteht, als Gödelzahl ansehen. Was meinen Sie dazu?

While-Programme (ohne Ergänzungen) sind unter Benutzung von k Symbolen aufgebaut. Jedem Symbol ordnen wir mit der Funktion κ eineindeutig eine der Zahlen zwischen 1 und k zu. Da Zahlen üblicherweise ohne führende Nullen geschrieben werden, dürfen wir die 0 keinem Symbol zuordnen. Also $g = k + 1$. Dem Programm $w = w_n \dots w_1$ ordnen wir die g -adische Zahl $\kappa(w_n) \cdot g^{n-1} + \dots + \kappa(w_2) \cdot g + \kappa(w_1)$ zu. Diese Zuordnung bezeichnen wir mit G .

1. Injektivität von G : Begründung wie bei der in der Vorlesung behandelten Gödelisierung.

2. Algorithmische Berechenbarkeit von G :

Die zur Berechnung der g -adischen Zahl notwendige Multiplikationen und Exponentiationen sind alle mit Hilfe von While-Programmen definierbar. Wir benötigen noch While-Programme, die s Zahlen als Argumente haben und deren Summe berechnen. (Für jedes s ist ein solches Programm anzugeben. Wir haben nicht die Möglichkeit Programme mit variabler Anzahl von Argumenten zu schreiben.)

$$X_1 := X_1 + X_2;$$

$$X_1 := X_1 + X_3;$$

...

$$X_1 := X_1 + X_s$$

ist ein Schema für diese Programme.

3. Es gibt einen Algorithmus, der zu jedem $n \in N$ ein w mit $G(w) = n$ findet.

Λ sei wieder das nirgendwo terminierende Programm.

Zum Nachweis der letzten Aussage zeigen wir, daß für beliebiges n stets einer der folgenden beiden Fälle eintritt.

1. n enthält die Ziffer 0.

Dann ist n nicht Bild eines syntaktisch korrekten While-Programms.

Ergebnis: Λ

2. Jede g -adische Ziffer z von n wird durch $\kappa^{-1}(z)$ ersetzt.

2.1. Das erhaltene Wort w ist syntaktisch korrekt.

Ergebnis: w

2.2. sonst

Ergebnis: Λ

Die Anzahl der Zeichen ist keine eineindeutige Abbildung, kann daher keine Gödelisierung sein.

b) Jede natürliche Zahl ist eindeutig als Produkt von Primzahlpotenzen darstellbar. Konstruieren Sie unter Benutzung dieses Sachverhalts eine Gödelisierung der While-Programme (ohne Ergänzungen).

Wir ordnen wie in Teil a mit Hilfe einer eineindeutigen Funktion κ jedem Symbol der While-Programme (ohne Ergänzungen) eine Zahl zu.

Sei p eine Funktion für die $p(i)$ die i -te Primzahl, also $p(0) = 2, p(1) = 3$. Einem While-Programm $w_1 \dots w_n$ ordne G die Zahl

$$\prod_{i=0}^{n-1} p(i)^{\kappa(w_{i+1})}$$

zu.

1. G ist injektiv, da die Zerlegung einer natürlichen Zahl in ein Produkt von Primzahlpotenzen eindeutig ist.

2. G ist algorithmisch berechenbar, da es für die Multiplikation, die Potenz und p While-Programme gibt. Für die Multiplikation von s Zahlen können wir wie in Teilaufgabe a ein Programmschema angeben.

3. Dekodierung einer Zahl n

Es gibt ein While-Programm exp , für das $exp(n, i)$ der größte Exponent ist, für den $p(i)$ Teiler von n ist.

1. $i := 0$;

$w_i := exp(n, i)$;

$i := i + 1$;

Wiederhole 1 solange, bis $w_i = 0$.

2.1. $\kappa^{-1}(w_1) \dots \kappa^{-1}(w_n)$ ist nicht syntaktisch korrektes While-Programm

Ergebnis: Λ

2.2. sonst

Ergebnis: $\kappa^{-1}(w_1) \dots \kappa^{-1}(w_n)$

Aufgabe 5 Übersetzungstechniken für Programme der ergänzten While-Sprache (Käufel)

a) Geben Sie eine allgemeine Methode an, die erlaubt, Zuweisungen aus der While-Sprache mit Ergänzungen durch Folgen von Zuweisungen der Form $x := T$ äquivalent zu ersetzen. T ist dabei Term der While-Sprache, von der Form $y \text{ op } z$, mit op als Operationssymbol der ergänzten While-Sprache oder Variable.

1. Bereinigung der Argumente von Nachf und Vorg:

$$x := T$$

T enthält Term der Form $\text{Vorg}(S)$ bzw. $\text{Nachf}(S)$ und S Term der ergänzten While-Sprache. Die Zuweisung wird ersetzt durch

$$h := S; x := T',$$

wobei T' aus T durch Ersetzung von S durch h entsteht. h neue Variable.

Auf diese Weise können alle Argumente für Vorg und Nachf in Argumente umgewandelt werden, die zur While-Sprache gehören. In dem so bereinigten Programm ist in Zuweisungen $x := T$ der Term T schon in der While-Sprache oder enthält nur noch Operatoren $+$, $*$, \div usw.

2. Bereinigung der Argumente der Operatoren. Wir setzen voraus, daß op in $T_1 \text{ op } T_2$ das äußerste Operationssymbol ist.

$$x := T_1 \text{ op } T_2$$

Ist T_1 keine Variable, wird die Zuweisung durch

$$h := T_1; x := h \text{ op } T_2$$

ersetzt. (h neue Variable.) Ist T_2 keine Variable, wird T_2 analog durch eine neue Variable ersetzt.

Damit können schrittweise alle Zuweisungen der Form $x := T_1 \text{ op } T_2$ durch solche der Form $x := y \text{ op } z$ ersetzt werden.

Angewandt auf ein ergänztes While-Programm liefert dieses Verfahren ein While-Programm, in dem alle Zuweisungen die Gestalt

$$x := y \text{ op } z, x := y, x := \text{Vorg}(y) \text{ oder } x := \text{Nachf}(y)$$

haben.

b) Geben Sie ein allgemeines Verfahren an, das erlaubt, Zuweisungen der Form $x := y \text{ op } z$ durch Einsetzen des Programms für op in eine äquivalente Folge von Anweisungen der While-Sprache umzuwandeln. (Hinweis: Verwenden Sie als Definition: Ein While-Programm ist ein Paar $(\{X_1, \dots, X_n\}, P)$, wobei P der Programmtext in der While-Sprache ist und $\{X_1, \dots, X_n\}$ die in P vorkommenden Variablen.)

Ersetzung von $x := y$ op z . Das Programm, das diese Zuweisung enthält, besitze die Variablen $\{X_1, \dots, X_l\}$. Zu op gehöre das Programm $(\{X_1, \dots, X_k\}, F)$. Die Zuweisung wird durch

$$X_{i+1} := y; X_{i+2} := z; F'; x := X_{l+1}$$

ersetzt, wobei F' aus F durch Ersetzung der X_1, \dots, X_k durch X_{l+1}, \dots, X_{l+k} hervorgeht. (Die X_{l+1}, \dots, X_{l+k} müssen neue Variable sein.)

Angewandt auf ein Programm, wie es als Ergebnis der Teilaufgabe a entsteht, erhält man eines, in dem die Zuweisungen bis auf diejenigen der Gestalt $x := y$ zur nicht ergänzten While-Sprache gehören.

c) Geben Sie die Schritte an, die notwendig sind, um ein Programm in der While-Sprache mit Ergänzungen in ein äquivalentes überzuführen, das die Bedingungen der Teilaufgabe a erfüllt.

Wir müssen voraussetzen, daß alle Variablen des Programms von der Form X_i sind. Zumindest muß bei einem While-Programm für eine k -stellige Funktion festgelegt sein, welche der Variablen die Rolle der X_1, \dots, X_k übernehmen. (Vgl. Definition 10.1.)

1. Ersetzung der bedingten Anweisungen wie in Abschnitt 9.2 beschrieben.

Dies ergibt ein Programm, in dem nur noch While-Anweisungen vorkommen.

2. Ersetzung von While-Anweisungen mit Prädikaten wie in Abschnitt 9.2 beschrieben.

Dies führt Zuweisungen der Form $u := t_p$ ein.

3. Das so erhaltene Programm kann mit Hilfe der Methoden unter a und b in eine Form gebracht werden, in der nur noch Zuweisungen der Form $x := y$ nicht zur (nicht ergänzten) While-Sprache gehören.

Aufgabe 6 Die fast überall verschwindende Funktion (Käufel)

Zeigen Sie, daß jede Funktion mit N als Definitionsbereich, die nur für endlich viele Argumente einen von Null verschiedenen Wert hat, berechenbar ist.

Für f sei $f(x_i) = y_i \neq 0$ für x_1, x_2, \dots, x_n . Das While-Programm zur Berechnung von f ist

```

if  $X_1 = x_1$  then  $X_1 := y_1$ 
else
if  $X_1 = x_2$  then  $X_1 := y_2$ 
...
else
if  $X_1 = x_n$  then  $X_1 := y_n$  else  $X_1 := 0$  fi
... fi fi

```

Das Programm erfüllt noch nicht die Bedingungen von Definition 9.1, kann aber mit den in Kapitel 9 der Vorlesung genannten Techniken dahingehend umgewandelt werden.

Es hat nur die Variable X_1 , die x_i und y_i sind Zahlen. (Anm. Zahlen ungleich 0 sind in der While-Sprache nicht als Terme zugelassen – durch entsprechende Nachfolgerterme ersetzen.)

Aufgabe 7 (Schuster)

Der Kode des Parsergenerator befindet sich auf den Internetseiten zur Vorlesung.