Formale Sprachen und Komplexität, SS 18,
Prof. Dr. Volker Heun

Übungsblatt 8

Abgabe: bis Mo. 25.06.2018 8 Uhr

Institut für Informatik **Ludwig-Maximilians-Universität München**
Prof. Dr. Volker Heun

Formale Sprachen und Komplexität, SS 18
Übungsblatt 8

Abgabe: bis Mo. 25.06.2018 8 Uhr

Nach Bearbeitung dieses Übungsblattes sollten Sie:

	Check
Einfache Abfragen (z.B. *if*) mit Hilfe von LOOP-, WHILE- und GOTO-Programmen simulieren können.	

Diese Ziele sind wichtige Hinweise für die Klausur!

Erlaubte Konstrukte für		
LOOP-Programme	WHILE-Programme	GOTO-Programme
$x_i := x_j + c$	$x_i := x_j + c$	M_k: $x_i := x_j + c$
$x_i := x_j - c$	$x_i := x_j - c$	M_k: $x_i := x_j - c$
P_1 ; P_2	P_1 ; P_2	P_1 ; P_2
`LOOP` x_i `DO` P `END`	`WHILE` $x_i \neq 0$ `DO` P `END`	M_k: `GOTO` M_j
		M_k: `IF` $x_i = c$ `THEN GOTO` M_j
		M_k: `HALT`
		jede Marke nur ein Mal

jeweils für $i, j, k \in \mathbb{N}$ und $c \in \mathbb{N}$

In allen Aufgaben sind diese Konstrukte genau in der angegebenen Form zu verwenden, keine Abkürzungen oder vereinfachenden Schreibweisen. Allerdings sind verständlichere Bezeichner für etwaige Hilfsvariablen erlaubt.

Aufgabe 8-1 schriftlich bearbeiten
LOOP-, WHILE-, GOTO-Programme

Geben Sie jeweils ein LOOP-Programm, ein WHILE-Programm und ein GOTO-Programm an, das die modifizierte Subtraktion für natürliche Zahlen berechnet. Jedes dieser Programme soll also die gleiche Wirkung haben wie die (in keinem der drei Berechnungsmodelle erlaubte) Zuweisung $x_0 := x_1 - x_2$, mit Ergebnis 0 falls der Wert von x_2 größer ist als der Wert von x_1.

Lösungsvorschlag:

Implementierung der modifizierten Subtraktion $f: \begin{array}{lcl} \mathbb{N} \times \mathbb{N} & \to & \mathbb{N} \\ (n_1, n_2) & \mapsto & n_1 \dot{-} n_2 \end{array}$

Konvention in allen drei Berechnungsmodellen:
Endwert von x_0 ist Ergebnis, Startwert von x_1 ist n_1, Startwert von x_2 ist n_2.

LOOP-Programm	WHILE-Programm	GOTO-Programm
$x_0 := x_1 + 0;$	$x_0 := x_1 + 0;$	$M_1:$ $x_0 := x_1 + 0;$
	$x_{99} := x_2 + 0;$	$M_2:$ $x_{99} := x_2 + 0;$
`LOOP` x_2 `DO`	`WHILE` $x_{99} \neq 0$ `DO`	$M_3:$ `IF` $x_{99} = 0$ `THEN GOTO` $M_7;$
	$x_{99} := x_{99} - 1;$	$M_4:$ $x_{99} := x_{99} - 1;$
$x_0 := x_0 - 1$	$x_0 := x_0 - 1$	$M_5:$ $x_0 := x_0 - 1;$
`END`	`END`	$M_6:$ `GOTO` $M_3;$
		$M_7:$ `HALT`

Aufgabe 8-2 schriftlich bearbeiten
WHILE-Programme

Simulieren Sie das Konstrukt

$$\texttt{IF } x_i = 0 \texttt{ THEN } P_0 \texttt{ ELSE } P_1 \texttt{ END}$$

durch ein WHILE-Programm. Die Simulation soll auch dann funktionieren, wenn die in der Bedingung verwendete Variable in P_0 oder in P_1 vorkommt.

Lösungsvorschlag:

Ziele:
P_0 und P_1 dürfen nur im entsprechenden Fall ausgeführt werden, und dann nur ein Mal.
Die Variablen in P_0 und P_1 dürfen nur von diesen selbst verändert werden.
Das gilt auch, falls die Testvariable x_i darin vorkommt.

Angenommene Hilfsvariablen, die nicht in P_0, P_1 vorkommen:

null	nie überschrieben, also stets Wert 0
test	Speicher für Wert von x_i, damit x_i unverändert bleibt
ja	1 falls $x_i = 0$ und 0 sonst
nein	0 falls $x_i = 0$ und 1 sonst

Grundidee, wenn wir einseitiges IF zur Verfügung hätten:

$ja := null + 1;$
$nein := null + 0;$ } Vorbesetzung für Fall $x_i = 0$

`IF` $x_i \neq 0$ `THEN`
  $ja := null + 0;$
  $nein := null + 1$ } Umkehrung der Besetzung falls $x_i \neq 0$
`END;`

`IF` $ja \neq 0$ `THEN` P_0 `END;`

`IF` $nein \neq 0$ `THEN` P_1 `END`

Endgültiges Programm

$test := x_i + 0;$ überschreibbare Kopie des Werts von x_i
$ja := null + 1;$
$nein := null + 0;$ } Vorbesetzung für Fall $x_i = 0$

`WHILE` $test \neq 0$ `DO`
  $ja := null + 0;$
  $nein := null + 1;$ } Umkehrung der Besetzung falls $x_i \neq 0$
  $test := null + 0$ Schleifenabbruch
`END;`

`WHILE` $ja \neq 0$ `DO`
  $P_0;$
  $ja := null + 0$ Schleifenabbruch
`END;`

`WHILE` $nein \neq 0$ `DO`
  $P_1;$
  $nein := null + 0$ Schleifenabbruch
`END`

Aufgabe 8-3 LOOP-Programme

Geben Sie ein LOOP-Programm an, das die modifizierte Division für natürliche Zahlen berechnet. Das Programm soll also die gleiche Wirkung haben wie die (nicht erlaubte) Zuweisung $x_0 := x_1$ `DIV` x_2. Zum Beispiel gilt:

$$\begin{array}{lll} 0\,\texttt{DIV}\,3 = 0, & 3\,\texttt{DIV}\,3 = 1, & 6\,\texttt{DIV}\,3 = 2, \\ 1\,\texttt{DIV}\,3 = 0, & 4\,\texttt{DIV}\,3 = 1, & 7\,\texttt{DIV}\,3 = 2 \\ 2\,\texttt{DIV}\,3 = 0, & 5\,\texttt{DIV}\,3 = 1, & \end{array}$$

außerdem sei n `DIV` $0 = 0$.

Sie können die zusätzlichen Konstrukte $x_i := x_j - x_k$ und `IF` $x_i \neq 0$ `THEN` P `END` verwenden (die durch LOOP-Programme mit den ursprünglichen Konstrukten simulierbar sind).

Lösungsvorschlag:

Wirkung soll sein wie $x_0 := x_1$ `DIV` x_2 falls $x_2 \neq 0$, andernfalls $x_0 := 0$.
erlaubte Zusatzkonstrukte $x_i := x_j - x_k$ und `IF` $x_i \neq 0$ `THEN` P `END`

Vorüberlegungen:

Kern des Programms ist eine Schleife:
im Rumpf subtrahiert man von x_1 immer wieder x_2 und zählt mit, wie oft das geht:

$$x_1 := x_1 - x_2; \qquad x_0 := x_0 + 1;$$

Beispiele für Abbruchkriterium (Spalte z kommt erst später dazu)

	x_0	x_1	x_2	z
Startwerte:	0	6	3	4
	1	3		1
	2	0		0
	3	0		

	x_0	x_1	x_2	z
Startwerte:	0	8	3	6
	1	5		3
	2	2		0
	3	0		

Den Fall $x_2 = 0$ muss man abfangen, dafür terminiert die Schleife nicht.
Für $x_2 > 0$ hat x_0 den richtigen Wert, sobald $x_1 < x_2$ ist.

Grundidee, wenn wir WHILE mit beliebigen Bedingungen zur Verfügung hätten

```
IF x2 ≠ 0 THEN
    WHILE NOT(x1 < x2) DO
        x1 := x1 − x2;
        x0 := x0 + 1
    END
END
```

Abgesehen von der zu komplexen Bedingung stellt sich die Frage,
wie man diese WHILE-Schleife als LOOP-Programm realisieren kann.
Für eine LOOP-Schleife müssen wir die Anzahl der Durchläufe angeben.
Die soll aber gerade durch das Hochzählen von x_0 bestimmt werden.
Wenn wir diese Anzahl bei Schleifeneintritt kennen würden,
bräuchten wir die LOOP-Schleife ja gar nicht mehr.

Beobachtung

Wir kennen nicht die genaue Anzahl der Durchläufe, aber wir kennen eine Obergrenze:
Für $x_2 > 0$ muss die Schleife höchstens x_1 Mal durchlaufen werden.

In solchen Fällen kann man die WHILE-Schleife immer simulieren
durch eine LOOP-Schleife mit einseitigem IF im Rumpf:

```
IF x2 ≠ 0 THEN
    LOOP x1 DO
        IF NOT(x1 < x2) THEN
            x1 := x1 − x2;
            x0 := x0 + 1
        END
    END
END
```

Der Übergang zu diesem Programm war der zentrale Punkt für die Lösung.
Jetzt muss nur noch die zu komplexe Bedingung auf einen Test $\neq 0$ zurückgeführt werden.

Nebenrechnung

$x_1 < x_2$ gdw. $x_1 - x_2 < 0$ gdw. $x_1 + 1 - x_2 < 1$ gdw. $x_1 + 1 - x_2 = 0$
Der letzte Schritt wäre falsch in $\mathbb{Z}$ oder $\mathbb{R}$, aber er gilt in $\mathbb{N}$.

Also Hilfsvariable z mitführen, so dass stets $z = x_1 + 1 - x_2$ ist. (Spalte z in Tabelle)

Endgültiges Programm

```
IF x2 ≠ 0 THEN
    z := x1 + 1;   z := z − x2;
    LOOP x1 DO
        IF z ≠ 0 THEN
            z := z − x2;
            x1 := x1 − x2;
            x0 := x0 + 1
        END
    END
END
```

Bemerkung:

Der Endwert von x_1 ist der Rest der Division.

Ein Programm mit der Wirkung $x_0 := x_1$ `MOD` x_2 erhält man
mit der zusätzlichen Zeile $x_0 := x_1 + 0$

Alternative Lösung:

```
IF x_2 ≠ 0 THEN
    LOOP x_1 DO
        diff := x_1 − x_2;
        IF diff ≠ 0 THEN
            x_0 := x_0 + 1;
            x_1 := x_1 − x_2
        END
    END
    x_0 := x_0 + 1;          //Falls x_1 MOD x_2 == 0 fehlt noch 1 zum richtigen Ergebnis
    diff := x_2 − x_1;       //x_1 ≤ x_2 an dieser Stelle, da x_1 der Rest der Division
    IF diff ≠ 0 THEN         //Falls x_1 MOD x_2 ≠ 0 rückgängig machen
        x_0 := x_0 − 1
    END
END
```

Aufgabe 8-4 schriftlich bearbeiten

Umkehrfunktion einer WHILE-berechenbaren Funktion

Sei $f : \mathbb{N} \to \mathbb{N}$ eine WHILE-berechenbare Bijektion auf den natürlichen Zahlen. Zeigen Sie, dass die Umkehrfunktion f^{-1} ebenfalls WHILE-berechenbar ist, indem sie ein entsprechendes Programm angeben.

Lösungsvorschlag:

Sei `P` das Programm, das f berechnet. Wie Üblich nimmt es die Variable `x_1` als Eingabewert und schreibt den berechneten Wert in die Variable `x_0`.
Wir schreiben ein WHILE Programm, das eine Variable i so lange erhöht bis $y = f(i)$ gleich dem Eingabewert des Programms ist.
Dazu benötigen wir eine Testvariable z, in die die absolute Differenz von y und dem Eingabewert gespeichert wird.

$x := x_1 + 0$
$y := 1;$
$i := 0;$
$z := 1;$
`WHILE` $z \neq 0$ `DO`
    $x_1 := i + 0;$
    $P;$
    $y := x_0 + 0;$
    $i := i + 1;$
    $z := |y - x|;$
`END`
$x_0 := i - 1;$