Technische Universität München
Fakultät für Informatik
M. Bauer, D. Golias, S. Riesner,
Dr. M. Schneider

WS 2002/2003
Lösungsvorschläge zu Blatt 8
28. November 2002

Übungen zu Einführung in die Informatik I

Aufgabe 24 Substitution mittels Markow-Algorithmen (Lösungsvorschlag)

a) Falls x_1 in t_1 nicht vorkommt, besteht die einfachste Lösung aus nur einer Regel

$$x_1 \to t_1 \tag{1}$$

(Dies ist streng genommen ein Regelschema für beliebige vorgegebene x_1 und t_1.) Unter Umständen muss in dieser Regel der Term t_1 noch geklammert werden, damit nach Substitution wieder ein korrekter Term entsteht. Das hängt von t_1 ab und sei vorher erledigt:

$$x_1 \to (t_1) \tag{2}$$

Falls x_1 in t_1 vorkommt, ist dieses Textersetzungssystem ungeeignet. Sobald nämlich x_1 in t vorkommt, würde eine Berechnung das mit dem Term t_1 neu eingesetzte x_1 auch wieder substituieren, d.h. die Berechnung terminiert nicht.

Wir formulieren einen Markow-Algorithmus, der den Term t von links nach rechts durchläuft und dabei die Substitution aller x_1 durch t_1 vornimmt. Die aktuelle Arbeitsposition wird durch ein "Schiffchen" # angezeigt.

#(	$\to$	(#	Schiffchen weiter nach rechts schieben
#)	$\to$	)#	- " -
#$\neg$	$\to$	$\neg$ #	- " -
#$\wedge$	$\to$	$\wedge$#	- " -
#$\vee$	$\to$	$\vee$#	- " -
#$\Rightarrow$	$\to$	$\Rightarrow$#	- " -
#$\Leftrightarrow$	$\to$	$\Leftrightarrow$#	- " -
#true	$\to$	true#	- " -
#false	$\to$	false#	- " -
#x_1	$\to$	t_1#	Substitution ausführen, ggf. Regel #$x_1 \to (t_1)$#
#x_2	$\to$	x_2#	Schiffchen weiter nach rechts schieben
$\vdots$		$\vdots$	
#x_n	$\to$	x_n#	Schiffchen weiter nach rechts schieben
#	$\to .$	ε	Schiffchen löschen und terminieren
ε	$\to$	#	Schiffchen ganz links einführen

Die letzten beiden Regeln müssen unbedingt am Schluss stehen, und zwar in der angegebenen Reihenfolge. Davor ist die Reihenfolge unerheblich.

b) Das in (a) angegebene Textersetzungssystem würde nicht terminieren ohne die vorletzte Regel, da immer wieder ein neues Schiffchen erzeugt würde. Es kann kein Textersetzungssystem ohne Punktregel geben, das die Substitution korrekt durchführt und terminiert. Nehmen wir an, R wäre so eines. Dann würde beim Beispiel

$$\begin{aligned} t &=_{def} x_1 \\ t_1 &=_{def} (\neg x_1) \end{aligned}$$

für die durch R induzierte Abbildung f_R gelten:

$$\begin{aligned} f_R(x_1) &= (\neg x_1) \\ f_R((\neg x_1)) &= (\neg(\neg x_1)) \end{aligned}$$

das heißt, R liefert in der Berechnung zu x_1 das Ergebnis $(\neg x_1)$. Darauf ist aber R wieder anwendbar und soll das Ergebnis $(\neg(\neg x_1))$ liefern. Ohne eine terminierende Punktregel ist also R immer wieder anwendbar und die Berechnung zu t terminiert nicht. Dies ist ein Widerspruch zur Annahme.

c) Damit die vorgegebene simultane Substitution definiert ist, müssen $x_1, ..., x_n$ paarweise verschieden sein. Das Textersetzungssystem ist analog zu dem in (a)

#(	$\rightarrow$	(#	Schiffchen weiter nach rechts schieben
#)	$\rightarrow$	)#	- " -
#$\neg$	$\rightarrow$	$\neg$ #	- " -
#$\wedge$	$\rightarrow$	$\wedge$#	- " -
#$\vee$	$\rightarrow$	$\vee$#	- " -
#$\Rightarrow$	$\rightarrow$	$\Rightarrow$#	- " -
#$\Leftrightarrow$	$\rightarrow$	$\Leftrightarrow$#	- " -
#true	$\rightarrow$	true#	- " -
#false	$\rightarrow$	false#	- " -
#x_1	$\rightarrow$	t_1#	(*) Subst. ausführen, ggf. Regel #$x_1 \rightarrow (t_1)$#
#x_2	$\rightarrow$	t_2#	(*) Subst. ausführen, ggf. Regel #$x_2 \rightarrow (t_2)$#
$\vdots$		$\vdots$	
#x_n	$\rightarrow$	t_n#	(*) Subst. ausführen, ggf. Regel #$x_n \rightarrow (t_n)$#
#	$\rightarrow.$	ε	Schiffchen löschen und terminieren
ε	$\rightarrow$	#	Schiffchen ganz links einführen

d) Für die mit (*) markierten Substitutionsregeln in (c) haben wir in diesem speziellen Beispiel:

#x	$\rightarrow$	y#	(*) Substitution ausführen
#y	$\rightarrow$	x#	(*) Substitution ausführen

Die Berechnung für den Term $(x \wedge y)$ liefert:
$(x \wedge y) \rightarrow \#(x \wedge y) \rightarrow (\#x \wedge y) \rightarrow (y\# \wedge y) \rightarrow (y \wedge \#y) \rightarrow (y \wedge x\#) \rightarrow (y \wedge x)\# \rightarrow (y \wedge x)$.

Aufgabe 25 Programmierung eines Interpreters für Markowalgorithmen

Im Rahmen dieser Aufgaben soll ein einfacher Interpreter für Markow-Algorithmen mit Hilfe von OCaml implementiert werden.

a) Implementieren Sie eine Funktion *istAnfang* die ermittelt, ob eine Liste Anfang einer anderen Liste ist.

```
let rec istanfang first second =
  match (first, second) with
   | ([], _) -> true
   | (_, []) -> false
   | (a::xs, b::ys) when (a=b) -> istanfang xs ys
   | (a::xs, b::ys)            -> false;;
```

b) Implementieren Sie eine Funktion *istTeil* die ermittelt, ob eine Liste Teil einer anderen Liste ist.

```
let rec istteil first second =
  match (first, second) with
   | (_,[]) -> false
   | (x,y) when istanfang x y -> true
   | (xs, b::ys) -> istteil xs ys;;
```

c) Implementieren Sie eine Funktion *vorwort* die, falls die erste Liste in der zweiten vorkommt, den Teil der ersten Liste zurückgibt, der **vor** der zweiten Liste steht.

```
let rec vorwort first second =
  match (first, second) with
   | (_,[]) -> failwith "undefined"
   | (xs,ys) when not (istteil xs ys) -> failwith "undefined"
   | (xs,ys) when (istanfang xs ys)   -> []
   | (xs,b::ys) -> b::(vorwort xs ys);;
```

d) **(H)** Verwenden Sie die Funktion *markow* die Sie in der Datei `markow.ml` finden, um den Markow-Algorithmus aus der Lösung von Aufgabe 24 d), zu implementieren.

```
let subst = [
   (['#'; '('], ['('; '#'], false);
   (['#'; ')'], [')'; '#'], false);
   (['#'; '-'], ['-'; '#'], false);
   (['#'; '^'], ['^'; '#'], false);
   (['#'; 'v'], ['v'; '#'], false);
   (['#'; '>'], ['>'; '#'], false);
   (['#'; '='], ['='; '#'], false);
   (['#'; 't'], ['t'; '#'], false);
   (['#'; 'f'], ['f'; '#'], false);
   (['#'; 'x'], ['y'; '#'], false);
   (['#'; 'y'], ['x'; '#'], false);
   (['#'], [], true);
    ([], ['#'], false)
];;

markow ['('; 'x'; '^'; 'y'; ')'] subst ;;
```

Aufgabe 26 Die Präfixrelation (Lösungsvorschlag)

a) Es wird gezeigt, dass die Präfixrelation reflexiv, antisymmetrisch und transitiv ist:

Reflexivität: Sei $x \in C^*$. Da $x = x \circ \varepsilon$ gilt für alle $x \in C^*$ folgt $x \sqsubseteq x$ und die Präfixrelation ist reflexiv.

Antisymmetrie: Sei $x, y \in C^*$; außerdem gelte $x \sqsubseteq y$ und $y \sqsubseteq x$. Dann gibt es ein $u, u' \in C^*$ derart, dass $y = x \circ u$ und $x = y \circ u'$. Daraus folgt $y = y \circ u \circ u'$ und $u = u' = \varepsilon$. Damit gilt $y = x \circ \varepsilon$ und schließlich $y = x$. Die Präfixrelation ist somit antisymmetrisch.

Transitivität: Sei $x, y, z \in C^*$; außerdem gelte $x \sqsubseteq y$ und $y \sqsubseteq z$. Dann gibt es ein $v, v' \in C^*$ derart, dass $y = x \circ v$ und $z = y \circ v'$. Wir erhalten also $z = x \circ v \circ v'$. Mit $v'' = v \circ v'$ folgt $z = x \circ v''$ oder $x \sqsubseteq z$. Die Präfixrelation ist somit transitiv.

Insgesamt ist also gezeigt, dass die Präfixrelation eine Halbordnung ist.

b) Totalität, Wohlordnung und Fundiertheit der Präfixrelation:

Wir zeigen durch Angabe eines Gegenbeispiels: Die Präfixrelation ist keine totale Ordnung. Sei $C = \{a, b\}$ und $x, y \in C^*$ mit $x = a$ und $y = b$. Dann gilt weder $x \sqsubseteq y$ noch $y \sqsubseteq x$ und die Präfixrelation ist nicht total.

Eine partielle Ordnung heißt bekanntlich fundiert, wenn jede (echt) absteigende Kette endlich ist: Durch sukzessives Löschen eines Zeichens am „rechten Ende“ eines Wortes $x \in C^+$, erhalten wir stets eine echt absteigende Kette, die nach endlich vielen Schritten mit dem leeren Wort ε abbricht.

Eine partielle Ordnung heißt wohlgeordnet, wenn jede Teilmenge ein kleinstes Element hat. Wichtig ist hierbei, dass ein kleinstes Element mit allen anderen Elementen vergleichbar, d.h. in Relation steht. Nun gibt es aber in der Praefixrelation Teilmengen $N \subset C^*$ die zwar minimale, nicht jedoch ein kleinstes Element enthalten; Beispiel $N = \{aabb, aa, b, bb\}$; hierbei sind aa und b minimal, d.h. minimale Elemente einer absteigenden Kette, jedoch unvergleichbar hinsichtlich der Praefixrelation. Die Praefixrelation ist also nicht wohlgeordnet wohl aber fundiert.

c) Wir definieren zunächst die lexikographische Ordnung $\preceq$ auf C^* wie folgt:

$$
\begin{aligned}
\varepsilon \preceq x, &\qquad \forall\, x \in C^* \\
x \preceq y, &\qquad x, y \in C^* \;\wedge\; x \sqsubseteq y \\
xay \preceq xby' &\qquad x, y, y' \in C^* \;\wedge\; a, b \in C \;\wedge\; a <_C b
\end{aligned}
$$

Hierbei ist $\sqsubseteq$ die Praefixrelation und $<_C$ die auf dem Alphabet geltende totale Ordnung. Diese lexikographische Ordnung ist nicht fundiert, denn für $a \leq_C b$ gibt es unendlich viele Wörter ax, $x \in C^*$ zwischen a und b und somit unendlich absteigende Ketten.

d) Eine fundierte totale Ordnung über C^* ist die Ordnung, die jedem $x \in C^*$ seine Länge zuordnet.