7. Übung Programmierung

Dominic Deckert

27. Mai 2017

Previously on ...

- Lambda-Kalkül
  - Fixpunkt-Kombinator
  - “Arithmetische” Kombinatoren

1 a)

$$(\lambda zx.z\,x(\lambda y.y\,x))(\lambda y.z\,x)(\lambda z.z)$$

1 a)

$(\lambda zx.z\,x(\lambda y.y\,x))(\lambda y.z\,x)(\lambda z.z)$

	$(\lambda z$	$x.z\,x$	$(\lambda y.y\,x))$	$(\lambda y.z\,x)$	$(\lambda z.z)$
		$GV=\{x,y\}$		$FV=\{z,x\}$	
$\Rightarrow_\alpha$	$(\lambda z$	$a.z\,a$	$(\lambda y.y\,a))$	$(\lambda y.z\,x)$	$(\lambda z.z)$
$\Rightarrow_\beta$		$(\lambda a.(\lambda y.z\,x)\,a$	$(\lambda y.y\,a))$		$(\lambda z.z)$
$\Rightarrow_\beta$		$((\lambda y.z\,x)$	$(\lambda z.z)$	$(\lambda y.y\,(\lambda z.z)))$	
$\Rightarrow_\beta$		$((z\,x)$		$(\lambda y.y\,(\lambda z.z)))$	

1 b)

Hinweis: $< f >=< Y >< F >$

1 b)

Hinweis: $< f >=< Y >< F >$

$$< F >= (\lambda fxy. \quad < ite > \quad (< iszero > (< pred > x)(< mult >< 2 > y)$$
$$(< add > y \quad (< mult > (< succ > x)(f(< pred > x)(< add > xy))))))$$

1 c)

Wie immer gilt hier

$$< Y >< G >= (\lambda h.(\lambda x.h(xx))(\lambda x.h(xx)))$$
$$\Rightarrow^* (\lambda x. < G > (xx))(\lambda x. < G > (xx)) =< Y_G >$$
$$\Rightarrow_\beta < G >< Y_G >$$

$$< Y >< G >< 3 >< 0 > \Rightarrow^* < G >< Y_G >< 3 >< 0 >$$
$$= (\lambda gxy.(< ite > (< iszero > y)(< succ > x)$$
$$(gx(< pred > y))) < Y_G >< 3 >< 0 >$$
$$\Rightarrow^* (< ite > (< iszero >< 0 >)(< succ >< 3 >)$$
$$(< Y_G >< 3 > (< pred >< 0 >)))$$
$$\Rightarrow^* < succ >< 3 >$$
$$\Rightarrow^* < 4 >$$

AM_0

AM_0: Rechnermodell mit Speicherregistern & Datenkeller
Befehlstypen:

- arithmetische/logische Befehle: ADD, DIV, GT (nur mit dem Keller)
- Transportbefehle: LOAD, LIT (vom Keller/ auf den Keller)
- Sprungbefehle JMP, JMC (nur auf Befehlszähler)

Protokoll

1. Befehlszähler (Nummer des auszuführenden Befehls)
2. Datenkeller (mit ":" getrennt, Kopf links)
3. Hauptspeicher (Liste von "Registernummer/ Wert"-Einträgen)
4. Input-Queue
5. Output-Queue

Protokoll - 1

BZ	DK	HS	In	Out
1	ε	[]	2	ε
2	ε	[1/2]	ε	ε
3	2			
4	1:2			
5	1			

Protokoll - 2

BZ	DK	HS	In	Out
6	ε			
7	2			
8	2:2			
9	1			
10	ε	[1/1]		
11				1

Protokoll - 3

BZ	DK	HS	In	Out
2				
3	1			
4	1:1			
5	0			
12	ε	[1/1]	ε	1

C_0

C_0: eingeschränkte C-Syntax, die nach AM_0 umgeformt werden kann

- Nur main-Funktion
- Variablen nur am Anfang deklariert
- keine Pointer

Umformung

baumstrukturierte Adressen: als Ziele von Sprüngen relevant
Adresse = Adresse der Oberstruktur . fortlaufende Nummer in Struktur
Oberstruktur: Zeile/ Schleifen-Kopf, -Rumpf/ etc.

Variablenadressen werden in Symboltabelle gespeichert
Adressen Schritt für Schritt erstellt (wenn nötig)

trans

$$
\begin{aligned}
trans(Max) =& trans(\#\text{include ¡stdio.h¿; int main()}\{block\}\\
=& blocktrans(block)\\
=& stseqtrans(\text{scanf...printf(...)}, tab_{\emptyset}[a/(var,1), b/(var,2), max/(var,3)], 1)\\
=& sttrans(\text{scanf(...)}, tab_1, 1.1)...sttrans(\text{printf(...)}, tab_1, 1.4)\\
=& READ1; READ2;\\
& boolexptrans(\text{a>b}, tab_1)JMC1.3.1;\\
& sttrans(\text{max = a;}, tab_1, 1.2)JMP1.3.3;\\
1.3.1& sttrans(\text{max = b;}, tab_1, 1.4)\\
1.3.3& WRITE3;
\end{aligned}
$$

transformiertes Programm

$READ1; READ2;$
$LOAD1; LOAD2;$
$GT; JMC1.3.1;$
$LOAD1; STORE3; JMP1.3.3;$
$1.3.1 LOAD2; STORE3;$
$1.3.3 WRITE3;$

Linearisierung mit Durchnummerieren der Befehle erreichbar

transformiertes Programm

1 : $READ1$; **2** : $READ2$;
3 : $LOAD1$; **4** : $LOAD2$;
5 : GT; **6** : JMC**10**;
7 : $LOAD1$; **8** : $STORE3$; **9** : JMP**12**;
10 : $LOAD2$; **11** : $STORE3$;
12 : $WRITE3$;

Protokoll - 1

BZ	DK	HS	In	Out
1	ε	[]	5:7	ε
2		[1/5]	7	
3		[1/5, 2/7]	ε	
4	5			
5	7:5			

Protokoll - 2

BZ	DK	HS	In	Out
6	0			
10	ε			
11	7			
12	ε	[1/5, 2/7, 3/7]		
13				7

Organisatorisches

Tutoren für das Wintersemester (AuD) gesucht!
Qualifikationen:

- Interesse am Stoff
- gutes Verständnis der Lehrinhalte
- kommunikative Fähigkeiten