1. Übungsbesprechung Programmkonstruktion

Karl Gmeiner

karl@complang.tuwien.ac.at

October 20, 2011

Termine

Übungsbesprechung

- Do, 20. Oktober 2011, 14:00 - 15:00
- Do, 27. Oktober 2011, 14:00 - 15:00
- Do, 17. November 2011, 14:00 - 15:00
- Mo, 12. Dezember 2011, 14:00 - 15:00
- Do, 12. Jänner 2011, 14:00 - 15:00

- Übungsabgaben bis spätestens 12:00 am Tag der Übungsbesprechung
- Bei Abgaben max. 5 MB.

Termine

Tests

- Test am Do, 10. November 2011, 13:00 - 14:00 und 14:00 - 15:00
- Test am Do, 1. Dezember 2011, 13:00 - 14:00 und 14:00 - 15:00
- Test am Do, 22. Dezember 2011, 13:00 - 14:00 und 14:00 - 15:00
- Test am Do, 26. Jänner 2011, 13:00 - 14:00 und 14:00 - 15:00

Aufgabe 1

Angabe

1. Ermitteln Sie Ihre Matrikelnummer und die von allen ihren Gruppenmitgliedern in Hexadezimal- und Binärdarstellung.
2. Trennen Sie die 16 *Least Significant Bits* ab und berechnen Sie Bit 15...8 |XOR| Bit 7...0. (z.B. wenn die 16 LSB 00101100.00011001 sind, berechnen Sie 00101100 |XOR| 00011001).
3. Wandeln Sie das Ergebnis in Dezimaldarstellung und Hexadezimaldarstellung um.

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567		

- Ergebnis als Binärzahl:

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	

- Ergebnis als Binärzahl:

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1

- Ergebnis als Binärzahl: 1

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1

- Ergebnis als Binärzahl: 11

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1

- Ergebnis als Binärzahl: 111

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1
154320	77160	0
77160	38580	0

- Ergebnis als Binärzahl: 00111

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1
154320	77160	0
77160	38580	0
⋮	⋮	⋮

- Ergebnis als Binärzahl: 10110101101000 00111

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1
154320	77160	0
77160	38580	0
⋮	⋮	⋮
4	2	0
2	1	0
1	0	1

- Ergebnis als Binärzahl: 100101101010110010000111

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1
154320	77160	0
77160	38580	0
⋮	⋮	⋮
4	2	0
2	1	0
1	0	1

- Ergebnis als Binärzahl: 100101101011010000111
- Als Hexadezimalzahl: $1.0010.1101.0110.1000.0111 = 12D687$

Aufgabe 1 - Teil 1

Matrikelnummer 1234567 in Binärzahl umwandeln

Zahl	Dividiert durch 2	Rest
1234567	617283	1
617283	308641	1
308641	154320	1
154320	77160	0
77160	38580	0
⋮	⋮	⋮
4	2	0
2	1	0
1	0	1

- Ergebnis als Binärzahl: 10010110101101000111
- Als Hexadezimalzahl: $1.0010.\underbrace{1101.0110}_{\text{Bit } 15 \ldots 8}.\underbrace{1000.0111}_{\text{Bit } 7 \ldots 0} = 12D687$

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

$$\begin{array}{rr} & 1101.0110 \\ \text{XOR} & 1000.0111 \\ \hline & \end{array}$$

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

	1101.0110
XOR	1000.0111
	0101.0001

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

$$\begin{array}{rr} & 1101.0110 \\ \text{XOR} & 1000.0111 \\ \hline & 0101.0001 \end{array}$$

0101.0001b = 51h in Dezimalzahl umwandeln

Horner-Schema: $(\cdots((a_k \times 2 + a_{k-1}) \times 2 + a_{k-2}) \times 2 \cdots) + a_0$

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

$$\begin{array}{rr} & 1101.0110 \\ \text{XOR} & 1000.0111 \\ \hline & 0101.0001 \end{array}$$

0101.0001b = 51h in Dezimalzahl umwandeln

Horner-Schema: $(\cdots((a_k \times 2 + a_{k-1}) \times 2 + a_{k-2}) \times 2 \cdots) + a_0$

Ziffer	Ergebnis Dezimal	
1	1	a_6

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

	1101.0110
XOR	1000.0111
	0101.0001

0101.0001b = 51h in Dezimalzahl umwandeln

Horner-Schema: $(\cdots((a_k \times 2 + a_{k-1}) \times 2 + a_{k-2}) \times 2 \cdots) + a_0$

Ziffer	Ergebnis Dezimal	
1	1	a_6
0	2	$a_6 \times 2 + a_5$

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

$$\begin{array}{rr} & 1101.0110 \\ \text{XOR} & 1000.0111 \\ \hline & 0101.0001 \end{array}$$

0101.0001b = 51h in Dezimalzahl umwandeln

Horner-Schema: $(\cdots((a_k \times 2 + a_{k-1}) \times 2 + a_{k-2}) \times 2 \cdots) + a_0$

Ziffer	Ergebnis Dezimal	
1	1	a_6
0	2	$a_6 \times 2 + a_5$
1	5	$(a_6 \times 2 + a_5) \times 2 + a_4$

Aufgabe 1 - Teil 2, 3

1101.0110 XOR 1000.0111

$$\begin{array}{rr} & 1101.0110 \\ \text{XOR} & 1000.0111 \\ \hline & 0101.0001 \end{array}$$

0101.0001b = 51h in Dezimalzahl umwandeln

Horner-Schema: $(\cdots((a_k \times 2 + a_{k-1}) \times 2 + a_{k-2}) \times 2 \cdots) + a_0$

Ziffer	Ergebnis Dezimal	
1	1	a_6
0	2	$a_6 \times 2 + a_5$
1	5	$(a_6 \times 2 + a_5) \times 2 + a_4$
0	10	...
0	20	...
0	40	...
1	81	...

Aufgabe 2: Assembler

Register	Variablen vom Typ Integer
Befehlszeiger	Zeiger auf den aktuellen Befehl (PC). Wird nach jedem Befehl außer Sprüngen um 1 erhöht.
Sprung	Verändert den PC.
Flag	Variable vom Typ Boolean

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Assembler - Befehle

`add` *Reg1 Reg2*	*Reg1* `+=` *Reg2*
`dec` *Reg*	*Reg*`-`
`cmp` *Reg Zahl*	`zero = (` *Reg* `==` *Zahl*`)`
`mov` *Reg Zahl*	*Reg* `=` *Zahl*
`je` *Adresse*	Bedingter Sprung: `if (zero) PC =` *Adresse*
`jmp` *Adresse*	Unbedingter Sprung: `PC =` *Adresse*

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov   dx   0
2:  cmp   ax   0
3:  je     7
4:  add   dx   ax
5:  dec   ax
6:  jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
1	3		
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov   dx   0
2:  cmp   ax   0
3:  je     7
4:  add   dx   ax
5:  dec   ax
6:  jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
2	3	0	
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
3	3	0	false
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	
5	1	6	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov   dx   0
2:  cmp   ax   0
3:  je     7
4:  add   dx   ax
5:  dec   ax
6:  jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
4	3	0	
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	
5	1	6	
6	0	6	

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov   dx   0
2:  cmp   ax   0
3:  je     7
4:  add   dx   ax
5:  dec   ax
6:  jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
5	3	3	
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	
5	1	6	
6	0	6	
2	0	6	

Aufgabe 2: Assemblerprogramm

Code

```
1:   mov   dx   0
2:   cmp   ax   0
3:   je     7
4:   add   dx   ax
5:   dec   ax
6:   jmp    2
7:
```

Programmausführung

PC	ax	dx	zero
6	2	3	
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	
5	1	6	
6	0	6	
2	0	6	
3	0	6	true

Aufgabe 2: Assemblerprogramm

Code

```
1:  mov  dx  0
2:  cmp  ax  0
3:  je    7
4:  add  dx  ax
5:  dec  ax
6:  jmp   2
7:
```

Programmausführung

PC	ax	dx	zero
2	2	3	
3	2	3	false
4	2	3	
5	2	5	
6	1	5	
2	1	5	
3	1	5	false
4	1	5	
5	1	6	
6	0	6	
2	0	6	
3	0	6	true
7			

Aufgabe 3: EBNF-Notation

EBNF-Notation

- $\{A\}$: beliebig oft (inklusive nie) A, d.h., leer, A, AA, $AA \ldots A$
- $[A]$: optional A, d.h., leer oder A
- $A|B$: A oder B

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

-0

-0: Auf ["-"] folgt keine `ZifferOhneNull`.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

0251

0251: Eine Zahl muss mit `ZifferOhneNull` beginnen.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

-12

-12: Auf ["-"] folgt eine `ZifferOhneNull`, darauf eine `Ziffer`.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

`A + -12`

`A + -12`: A ist keine `ZifferOhneNull`.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

`-1 + -1`

`-1 + -1`: -1 ist eine `Zahl` und somit ein `Wert`. `Wert + Wert` ist ein Ausdruck.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

74 - 99

`74 - 99`: 74 ist eine `Zahl`, 99 und -99 sind `Zahlen`. Aber `Wert Wert` oder `Wert - Wert` sind keine Ausdrücke.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

957 + 123 * -32

957 + 123 * -32: `Wert + Wert * Wert` ist kein `Ausdruck`.

Aufgabe 3: Grammatik

EBNF-Notation

Ausdruck	=	Wert { "+" Wert }
	|	Wert { "*" Wert }
Wert	=	Zahl
	|	"(" Ausdruck ")"
Zahl	=	["-"] ZifferOhneNull { Ziffer }
Ziffer	=	"0" | ZifferOhneNull
ZifferOhneNull	=	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

`12 + ( 642 * 984 * 0 ) + 65`

`12 + ( 642 * 984 * 0 ) + 65`: `Wert + ( Ausdruck ) + Wert` ist ein `Ausdruck`, `Wert * Wert * Wert` ebenso, aber `0` ist keine `Zahl` und somit kein `Wert`.

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z)$

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$
  - aber $\lambda x.(x + z) \neq_\alpha \lambda z.(z + z)$.

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$
  - aber $\lambda x.(x + z) \neq_\alpha \lambda z.(z + z)$.

η-Regel - Extensionalität

- $\lambda v.(fv) =_\eta f$ wobei $v \notin FV(f)$.

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$
  - aber $\lambda x.(x + z) \neq_\alpha \lambda z.(z + z)$.

η-Regel - Extensionalität

- $\lambda v.(fv) =_\eta f$ wobei $v \notin FV(f)$.
- Beispiel: $\lambda x.(\sin\ x)$

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$
  - aber $\lambda x.(x + z) \neq_\alpha \lambda z.(z + z)$.

η-Regel - Extensionalität

- $\lambda v.(fv) =_\eta f$ wobei $v \notin FV(f)$.
- Beispiel: $\lambda x.(\sin x) =_\eta \sin$

Aufgabe 4: Regeln des Lambda-Kalküls

α-Regel - Umbenennen von gebundenen Variablen

- $\lambda v.f =_\alpha \lambda u.[u/v]f$ wobei $u \notin FV(f)$.
- Beispiel:
  - $\lambda x.(x + z) =_\alpha \lambda y.(y + z)$
  - aber $\lambda x.(x + z) \neq_\alpha \lambda z.(z + z)$.

η-Regel - Extensionalität

- $\lambda v.(fv) =_\eta f$ wobei $v \notin FV(f)$.
- Beispiel: $\lambda x.(\sin\ x) =_\eta \sin$
- aber $\lambda x.(x\ x) \neq_\eta x$.

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x + y)))\ 1)\ 2$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x + y)))\ 1)\ 2$
    - $((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.(x + y))}_{f}\ \underbrace{1}_{e})\ 2$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x + y)))\ 1)\ 2$
    - $((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.(x + y))}_{f})\ \underbrace{1}_{e})\ 2 = ((\lambda v.f)e)\ 2$

      wobei $v = x, f = \lambda y.(x + y)$ und $e = 1$.

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x + y)))\ 1)\ 2$
    - $((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.(x + y))}_{f}\ \underbrace{1}_{e})\ 2 = ((\lambda v.f)e)\ 2$

      wobei $v = x, f = \lambda y.(x + y)$ und $e = 1$.
    - $((\lambda x.(\lambda y.(x + y)))\ 1)\ 2 =_\beta ([e/v]f)\ 2 = (\lambda y.(1 + y))\ 2$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x+y)))\ 1)\ 2$
    - $((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.(x+y))}_{f})\ \underbrace{1}_{e})\ 2 = ((\lambda v.f)e)\ 2$

      wobei $v = x, f = \lambda y.(x+y)$ und $e = 1$.
    - $((\lambda x.(\lambda y.(x+y)))\ 1)\ 2 =_\beta ([e/v]f)\ 2 = (\lambda y.(1+y))\ 2$
    - $(\lambda \underbrace{y}_{v}.\underbrace{(1+y)}_{f})\ \underbrace{2}_{e} =_\beta [2/y](1+y) = 1+2$

Lambda-Kalkül - β-Regel

β-Regel - Berechnungsschritt

- $(\lambda v.f)e =_\beta [e/v]f$: Die Funktion $f(v)$ wird mit dem Parameter e ausgeführt.
- Beispiel:
  - $(\lambda x.x)\ 5 =_\beta [5/x]x = 5$
  - $(\lambda x.0)\ 5 =_\beta [5/x]0 = 0$
  - $((\lambda x.(\lambda y.(x+y)))\ 1)\ 2$
    - $((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.(x+y))}_{f}\ \underbrace{1}_{e})\ 2 = ((\lambda v.f)e)\ 2$

      wobei $v = x, f = \lambda y.(x+y)$ und $e = 1$.
    - $((\lambda x.(\lambda y.(x+y)))\ 1)\ 2 =_\beta ([e/v]f)\ 2 = (\lambda y.(1+y))\ 2$
    - $(\lambda \underbrace{y}_{v}.\underbrace{(1+y)}_{f}\ \underbrace{2}_{e} =_\beta [2/y](1+y) = 1+2$
    - $1 + 2$ ist in β-Normalform.

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v} . \underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v}.\underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$

$$=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v} . \underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$

  $=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$

- $\lambda a.(\lambda b.(((\lambda \underbrace{x}_{v} . \underbrace{(\lambda y.x)}_{f})\ \underbrace{b}_{e})\ a))$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v}.\underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$
  $=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$
- $\lambda a.(\lambda b.(((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.x)}_{f})\ \underbrace{b}_{e})\ a))$ $=_\beta \lambda a.(\lambda b.((\lambda y.b)\ a))$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v}.\underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$
  $=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$
- $\lambda a.(\lambda b.(((\lambda \underbrace{x}_{v}.\underbrace{(\lambda y.x)}_{f})\ \underbrace{b}_{e})\ a))$ $=_\beta \lambda a.(\lambda b.((\lambda y.b)\ a))$
- $\lambda a.(\lambda b.((\lambda \underbrace{y}_{v}.\underbrace{b}_{f})\ \underbrace{a}_{e}))$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v} . \underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$
  $=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$
- $\lambda a.(\lambda b.(((\lambda \underbrace{x}_{v} . \underbrace{(\lambda y.x)}_{f})\ \underbrace{b}_{e})\ a))$ $=_\beta \lambda a.(\lambda b.((\lambda y.b)\ a))$
- $\lambda a.(\lambda b.((\lambda \underbrace{y}_{v} . \underbrace{b}_{f})\ \underbrace{a}_{e}))$ $=_\beta \lambda a.(\lambda b.b)$

Aufgabe 4 - Lösung

$(\lambda p.(\lambda a.(\lambda b.((p\ b)\ a))))\ (\lambda x.(\lambda y.x))$

β-Regel: $(\lambda v.f)e =_\beta [e/v]f$.

- $(\lambda \underbrace{p}_{v} . \underbrace{(\lambda a.(\lambda b.((p\ b)\ a)))}_{f})\ \underbrace{(\lambda x.(\lambda y.x))}_{e}$
  $=_\beta \lambda a.(\lambda b.(((\lambda x.(\lambda y.x))\ b)\ a))$
- $\lambda a.(\lambda b.(((\lambda \underbrace{x}_{v} . \underbrace{(\lambda y.x)}_{f})\ \underbrace{b}_{e})\ a))$ $=_\beta \lambda a.(\lambda b.((\lambda y.b)\ a))$
- $\lambda a.(\lambda b.((\lambda \underbrace{y}_{v} . \underbrace{b}_{f})\ \underbrace{a}_{e}))$ $=_\beta \lambda a.(\lambda b.b)$
- $\lambda a.(\lambda b.b)$ ist in β-Normalform.