

## Theorie der Programmierung Wintersemester 2005/06

### Übungsblatt 1

#### Aufgabe 1

Bestimmen Sie die small step Semantik (d.h. die Folge aller small steps) für die folgenden Ausdrücke:

- a. **let** *sum\_of\_squares* =  $\lambda x. \lambda y. + (* x x) (* y y)$   
**in** *sum\_of\_squares* 3 4
- b. **let** *twice* =  $\lambda f. \lambda x. f (f x)$   
**in let** *square* =  $\lambda x. * x x$   
**in** *twice square* 5
- c.  $(\lambda x. x x) (\lambda x. x x)$
- d. **let** *fact* =  $\lambda x. \mathbf{if} = x 0 \mathbf{then} 1 \mathbf{else} * x (\mathit{fact} (- x 1))$   
**in** *fact* 3

#### Aufgabe 2

Überzeugen Sie sich davon, dass unsere small step Semantik deterministisch ist, d.h. dass es für jeden Ausdruck  $e$  höchstens einen small step  $e \rightarrow r$  (mit  $r \in \mathit{Exp} \cup \mathit{Expn}$ ) gibt. Überlegen Sie sich dazu, welche Regel in welcher Situation in Frage kommt.

### Aufgabe 3

Geben Sie jeweils die exakte Herleitung des ersten small steps für die folgenden Ausdrücke an.

- a.  $+1$  (/ 10)
- b.  $+$  (/ 10) 1
- c.  $(\lambda x. x)$  (/ 10)
- d.  $(\lambda x. 0)$  (/ 10)
- e. **let**  $x =$  (/ 10) **in** 0
- f. **if**  $=$  (/ 10) 0 **then** *true* **else** *true*
- g. **if** *true* **then** *true* **else** (/ 10)

### Aufgabe 4

In unserer Programmiersprache gibt es keine Operatoren, die auf booleschen Werten arbeiten. Man kann solche Operatoren als Abkürzungen (syntaktischen Zucker) für  $\lambda$ -Abstraktionen auffassen, z.B. die Negation

*not* für  $\lambda x. \mathbf{if} \ x \ \mathbf{then} \ \mathit{false} \ \mathbf{else} \ \mathit{true}$

- a. Geben Sie ähnliche Definitionen für Konjunktion (*and*), Disjunktion (*or*) und Gleichheit (*eq*) von booleschen Werten an.
- b. Welchen Nachteil haben die so definierten Funktionen *and* und *or*?