

ÜBUNGEN ZU ORGANISATION UND ARCHITEKTUR VON RECHNERN SS 2002

SERIE 1 — MUSTERLÖSUNG

Aufgabe 1+2

(6+6 Punkte)

```
1 (define-struct zero ())
2 (define-struct succ (arg))
3 (define-struct add (arg1 arg2))
4 (define-struct mul (arg1 arg2))
5
6 ;; Transforms a number into the abstract representation
7 (define (num2ast n)
8   (if (> n 0)
9       (make-succ (num2ast (- n 1)))
10      (make-zero)))
11
12 ;; Adds arguments which contain only 'zero' and 'succ'
13 ;; (+ e1 e2) = (+ 1 (+ (- 1 e1) e2))
14 (define (eval-add e1 e2)
15   (cond ((zero? e1) e2)
16         ((succ? e1) (make-succ (eval-add (succ-arg e1)
17                                         e2))))))
18
19
20 ;; Multiplies arguments which contain only 'zero' and 'succ'
21 ;; (* e1 e2) = (+ e2 (* (- 1 e1) e2))
22 (define (eval-mul e1 e2)
23   (cond ((zero? e1) e1)
24         ((succ? e1) (eval-add e2
25                               (eval-mul (succ-arg e1)
26                                         e2))))))
27
28 ;; Replaces all 'add' and 'mul' in the argument by 'zero' and 'succ'
29 (define (eval expr)
30   (cond ((zero? expr) expr)
31         ((succ? expr) (make-succ (eval (succ-arg expr))))
32         ((add? expr) (eval-add (eval (add-arg1 expr))
33                                (eval (add-arg2 expr))))
34         ((mul? expr) (eval-mul (eval (mul-arg1 expr))
35                                (eval (mul-arg2 expr))))
36         (else "eval: illegal constructor found!")))
37
38 ;; Transforms an abstract representation into a number
39 (define (ast2num expr)
40   (cond ((zero? expr) 0)
41         ((succ? expr) (+ 1 (ast2num (succ-arg expr))))
42         ((add? expr) (+ (ast2num (add-arg1 expr))
43                         (ast2num (add-arg2 expr))))
44         ((mul? expr) (* (ast2num (mul-arg1 expr))
45                         (ast2num (mul-arg2 expr))))
46         (else "ast2num: illegal constructor found!")))
47
48 ;;
49 ;; Example: (+ 1 (+ (+ 4 (* (+ 2 0) 3)) 0))
50 ;;
51 (let ((ast (make-succ (make-add (make-add (num2ast 4)
52                                       (make-mul (make-add (num2ast 2)
53                                                         (num2ast 0))
54                                                         (num2ast 3)))
55                                       (num2ast 0))))))
56   (list (ast2num ast) ;; Should evaluate to a list with two identical numbers
57         (ast2num (eval ast))))
```

Aufgabe 3

(8 Punkte)

```

1  (require-library "match.ss")
2  (require-library "defstru.ss")
3
4  (define-structure (zero))
5  (define-structure (succ arg))
6  (define-structure (add arg1 arg2))
7  (define-structure (mul arg1 arg2))
8
9  ;; Transforms a number into the abstract representation
10 (define (num2ast n)
11   (if (> n 0)
12       (make-succ (num2ast (- n 1)))
13       (make-zero)))
14
15 ;; Replaces all 'add' and 'mul' in the argument by 'zero' and 'succ'
16 ;; Note: The functions 'eval-add' and 'eval-mul' have been eliminated by using pattern match :-)
17 (define (eval expr)
18   (match expr [($ zero)          expr]
19               [($ succ arg)      (make-succ (eval arg))]
20               ;;
21               ;; First argument is 'zero' -> Evaluate second argument
22               [($ add ($ zero) arg2) (eval arg2)]
23               ;;
24               ;; First argument is 'succ' -> Shift 'succ' to the front
25               [($ add ($ succ arg1) arg2) (make-succ (eval (make-add arg1
26                                                           arg2)))]
27               ;;
28               ;; First argument is neither 'zero' nor 'succ' -> Evaluate it
29               ;; Note: There is no need to evaluate the second argument here!
30               [($ add arg1 arg2) (eval (make-add (eval arg1)
31                                                           arg2))]
32               ;;
33               [($ mul ($ zero) arg2) (make-zero)]
34               [($ mul ($ succ arg1) arg2) (eval (make-add arg2
35                                                           (make-mul arg1
36                                                           arg2)))]
37               [($ mul arg1 arg2) (eval (make-mul (eval arg1)
38                                                           arg2))]
39               [_ "eval: illegal constructor found!"]))
40
41 ;; Transforms an abstract representation into a number
42 (define (ast2num
43   (match-lambda [($ zero) 0]
44                 [($ succ arg) (+ 1 (ast2num arg))]
45                 [($ add arg1 arg2) (+ (ast2num arg1)
46                                       (ast2num arg2))]
47                 [($ mul arg1 arg2) (* (ast2num arg1)
48                                       (ast2num arg2))]
49                 [_ "ast2num: illegal constructor found!"]))
50
51 ;;
52 ;; Example: (+ 1 (+ (+ 4 (* (+ 2 0) 3)) 0))
53 ;;
54 (let ((ast (make-succ (make-add (make-add (num2ast 4)
55                                       (make-mul (make-add (num2ast 2)
56                                                           (num2ast 0))
57                                                           (num2ast 3)))
58                                       (num2ast 0))))
59   (list (ast2num ast)          ;; Should evaluate to a list with two identical numbers
60         (ast2num (eval ast))))

```