Übungen zu Organisation und Architektur von Rechnern
SS 2002

Serie 1 — Musterlösung

Aufgabe 1+2 (6+6 Punkte)

```
(define-struct zero ())
(define-struct succ (arg))
(define-struct add (arg1 arg2))
(define-struct mul (arg1 arg2))

;; Transforms a number into the abstract representation
(define (num2ast n)
        (if (> n 0)
            (make-succ (num2ast (- n 1)))
            (make-zero)))

;; Adds arguments which contain only 'zero' and 'succ'
;; (+ e1 e2) = (+ 1 (+ (- 1 e1) e2))
(define (eval-add e1 e2)
        (cond ((zero? e1) e2)
              ((succ? e1) (make-succ (eval-add (succ-arg e1)
                                               e2)))))


;; Multiplies arguments which contain only 'zero' and 'succ'
;; (* e1 e2) = (+ e2 (* (- 1 e1) e2))
(define (eval-mul e1 e2)
        (cond ((zero? e1) e1)
              ((succ? e1) (eval-add e2
                                    (eval-mul (succ-arg e1)
                                              e2)))))

;; Replaces all 'add' and 'mul' in the argument by 'zero' and 'succ'
(define (eval expr)
        (cond ((zero? expr) expr)
              ((succ? expr) (make-succ (eval (succ-arg expr))))
              ((add? expr)  (eval-add (eval (add-arg1 expr))
                                      (eval (add-arg2 expr))))
              ((mul? expr)  (eval-mul (eval (mul-arg1 expr))
                                      (eval (mul-arg2 expr))))
              (else         "eval: illegal constructor found!")))

;; Transforms an abstract representation into a number
(define (ast2num expr)
        (cond ((zero? expr) 0)
              ((succ? expr) (+ 1 (ast2num (succ-arg expr))))
              ((add?  expr) (+ (ast2num (add-arg1 expr))
                               (ast2num (add-arg2 expr))))
              ((mul?  expr) (* (ast2num (mul-arg1 expr))
                               (ast2num (mul-arg2 expr))))
              (else         "ast2num: illegal constructor found!")))

;;
;; Example: (+ 1 (+ (+ 4 (* (+ 2 0) 3)) 0))
;;
(let ((ast (make-succ (make-add (make-add (num2ast 4)
                                          (make-mul (make-add (num2ast 2)
                                                              (num2ast 0))
                                                    (num2ast 3)))
                                (num2ast 0)))))
     (list (ast2num ast)            ;; Should evaluate to a list with two identical numbers
           (ast2num (eval ast))))
```

Aufgabe 3 (8 Punkte)

```
(require-library "match.ss")
(require-library "defstru.ss")

(define-structure (zero))
(define-structure (succ arg))
(define-structure (add arg1 arg2))
(define-structure (mul arg1 arg2))

;; Transforms a number into the abstract representation
(define (num2ast n)
        (if (> n 0)
            (make-succ (num2ast (- n 1)))
            (make-zero)))

;; Replaces all 'add' and 'mul' in the argument by 'zero' and 'succ'
;; Note: The functions 'eval-add' and 'eval-mul' have been eliminated by using pattern match :-)
(define (eval expr)
        (match expr [($ zero)                   expr]
                    [($ succ arg)               (make-succ (eval arg))]
                    ;;
                    ;; First argument is 'zero' -> Evaluate second argument
                    [($ add ($ zero) arg2)      (eval arg2)]
                    ;;
                    ;; First argument is 'succ' -> Shift 'succ' to the front
                    [($ add ($ succ arg1) arg2) (make-succ (eval (make-add arg1
                                                                          arg2)))]
                    ;;
                    ;; First argument is neither 'zero' nor 'succ' -> Evaluate it
                    ;; Note: There is no need to evaluate the second argument here!
                    [($ add arg1 arg2)          (eval (make-add (eval arg1)
                                                                arg2))]
                    ;;
                    [($ mul ($ zero) arg2)      (make-zero)]
                    [($ mul ($ succ arg1) arg2) (eval (make-add arg2
                                                                (make-mul arg1
                                                                          arg2)))]
                    [($ mul arg1 arg2)          (eval (make-mul (eval arg1)
                                                                arg2))]
                    [_                          "eval: illegal constructor found!"]))

;; Transforms an abstract representation into a number
(define ast2num
        (match-lambda [($ zero) 0]
                      [($ succ arg)      (+ 1 (ast2num arg))]
                      [($ add arg1 arg2) (+ (ast2num arg1)
                                            (ast2num arg2))]
                      [($ mul arg1 arg2) (* (ast2num arg1)
                                            (ast2num arg2))]
                      [_                 "ast2num: illegal constructor found!"]))

;;
;; Example: (+ 1 (+ (+ 4 (* (+ 2 0) 3)) 0))
;;
(let ((ast (make-succ (make-add (make-add (num2ast 4)
                                          (make-mul (make-add (num2ast 2)
                                                              (num2ast 0))
                                                    (num2ast 3)))
                                (num2ast 0)))))
     (list (ast2num ast)            ;; Should evaluate to a list with two identical numbers
           (ast2num (eval ast))))
```