Auf. 6.3

a)

(define (append x y)

(cond [ (empty? x) y]

[ (cons? x) (cons (first x) (append (rest x) y)) ] ))

a) (append empty l) ≡ l für alle Listen l

(append empty l)

EFUN

≡ (cond [ (empty? empty) l]

[ (cons? empty) (cons (first empty) (append (rest empty) l)) ] )

PRIM, ERED, EKONG

≡ (cond [#true l]

[ (cons? empty) (cons (first empty) (append (rest empty) l)) ] )

COND-true, ERED

≡ l

b)

(append l empty) ≡ l für alle Listen l.

Induktionsanfang (Basisfall): l ≡ empty

Zu zeigen: (append empty empty) ≡ empty

(append empty empty)

EFUN

≡ (cond [ (empty? empty) empty]

[ (cons? empty) (cons (first empty) (append (rest empty) empty)) ] )

PRIM, ERED, EKONG

≡ (cond [#true empty]

[ (cons? empty) (cons (first empty) (append (rest empty) empty)) ] )

COND-true, ERED

≡ empty

Induktionsannahme: (append lst1 empty) ≡ lst1, lst2 ≡ (cons head lst1)

Induktionschritt:

Zu zeigen: (append lst2 empty) ≡ lst2

(append lst2 empty)

EFUN, EKONG

≡ (cond [(empty? (cons head lst1)) empty]

[(cons? (cons head lst1)) (cons (first (cons head lst1)) (append (rest (cons head lst1)) empty)) ] ))

PRIM, ERED, EKONG

≡ (cond [#false empty]

[(cons? (cons head lst1)) (cons (first (cons head lst1)) (append (rest (cons head lst1)) empty)) ] ))

COND-false, ERED

≡ (cond [ (cons? (cons head lst1)) (cons (first (cons head lst1)) (append (rest (cons head lst1)) empty)) ] ))

STRUCT-predtrue, EKONG

≡ (cond [#true (cons (first (cons head lst1)) (append (rest (cons head lst1)) empty)) ] ))

COND-true, ERED

≡ (cons (first (cons head lst1)) (append (rest (cons head lst1)) empty))

PRIM, ERED, EKONG

≡ (cons head (append lst1 empty))

Induktionsannahme, EKONG

≡ (cons head lst1)

Induktionsannahme, PRIM

≡ lst2

Damit ist bewiesen: (append l empty) ≡ l für alle Listen l.

c)

(append (cons x l1) l2) ≡ (cons x (append l1 l2)) für alle x und alle Listen l1, l2.

Induktionsanfang (Basisfall): l1 ≡ empty

Zu zeigen: (append (cons x empty) l2) ≡ (cons x (append empty l2))

(append (cons x empty) l2)

EFUN

≡ (cond [ (empty? (cons x empty)) l2]

[ (cons? (cons x empty)) (cons (first (cons x empty)) (append (rest (cons x empty)) l2)) ] )

PRIM, ERED, EKONG

≡ (cond [#false l2]

[ (cons? (cons x empty)) (cons (first (cons x empty)) (append (rest (cons x empty)) l2)) ] )

COND-false, ERED

≡ (cond [ (cons? (cons x empty)) (cons (first (cons x empty)) (append (rest (cons x empty)) l2)) ] )

STRUCT-predtrue, EKONG

≡ (cond [#true (cons (first (cons x empty)) (append (rest (cons x empty)) l2)) ] )

COND-true, ERED

≡ (cons (first (cons x empty)) (append (rest (cons x empty)) l2))

PRIM, ERED, EKONG

≡ (cons x (append empty l2))

Induktionsannahme: (append (cons x l0) l2) ≡ (cons x (append l0 l2)), l1 ≡ (cons z l0)

Induktionsschritt:

Zu zeigen: (append (cons x l1) l2) ≡ (cons x (append l1 l2))

(append (cons x l1) l2)

EFUN

≡ (cond [ (empty? (cons x l1)) l2]

[ (cons? (cons x l1)) (cons (first (cons x l1))(append (rest (cons x l1)) l2))])

PRIM, ERED, EKONG

≡ (cond [#false l2]

[ (cons? (cons x l1)) (cons (first (cons x l1))(append (rest (cons x l1)) l2))])

COND-false, ERED

≡ (cond [ (cons? (cons x l1)) (cons (first (cons x l1))(append (rest (cons x l1)) l2))])

STRUCT-predtrue, EKONG

≡ (cond [#true (cons (first (cons x l1)) (append (rest (cons x l1)) l2)) ] )

COND-true, ERED

≡ (cons (first (cons x l1)) (append (rest (cons x l1)) l2))

PRIM, ERED, EKONG

≡(cons x (append l1 l2))

Damit ist bewiesen: (append (cons x l1) l2) ≡ (cons x (append l1 l2)) für alle x und alle Listen l1, l2.

d)

(append l1 (append l2 l3)) ≡ (append (append l1 l2) l3) für alle Listen l1, l2, l3

Induktionsanfang (Basisfall): l1 ≡ empty

Zu Zeigen: (append empty (append l2 l3)) ≡ (append (append empty l2) l3)

(i) (append empty (append l2 l3))

EFUN

≡ (cond [(empty? empty) (append l2 l3)]

[ (cons? empty) (cons (first empty) (append (rest empty) (append l2 l3))) ] )

PRIM, ERED, EKONG

≡ (cond [#true (append l2 l3)]

[ (cons? empty) (cons (first empty) (append (rest empty) (append l2 l3))) ] )

COND-true, ERED

≡ (append l2 l3)

(ii) (append (append empty l2) l3)

Teilaufgabe (a), ERED

≡ (append l2 l3)

ETRANS

(append empty (append l2 l3)) ≡ (append (append empty l2) l3) ≡ (append l2 l3)

Induktionsannahme: (append l0 (append l2 l3)) ≡ (append (append l0 l2) l3), l0 ≡ (cons z l1)

Induktionsschritt:

Zu zeigen: (append l0 (append l2 l3)) ≡ (append (append l0 l2) l3)

(i) (append l0 (append l2 l3))

EFUN, EKONG

≡ (cond [ (empty? (cons z l1)) (append l2 l3)]

[ (cons? (cons z l1)) (cons (first (cons z l1)) (append (rest (cons z l1)) (append l2 l3))) ] )

PRIM, ERED, EKONG

≡ (cond [#false (append l2 l3)]

[ (cons? (cons z l1)) (cons (first (cons z l1)) (append (rest (cons z l1)) (append l2 l3))) ] )

COND-false, ERED

≡ (cond [ (cons? (cons z l1)) (cons (first (cons z l1)) (append (rest (cons z l1)) (append l2 l3))) ] )

STRUCT-predtrue, EKONG

≡ (cond [#true (cons (first (cons z l1)) (append (rest (cons z l1)) (append l2 l3))) ] )

COND-true, ERED

≡ (cons (first (cons z l1)) (append (rest (cons z l1)) (append l2 l3)))

PRIM, ERED, EKONG

≡ (cons z (append l0 (append l2 l3)))

Induktionsannahme, EKONG

≡ (cons z (append (append l0 l2) l3))

(ii) (append (append l0 l2) l3)

EFUN, EKONG

≡ (cond [ (empty? (append (cons z l1) l2)) l3]

[ (cons? (append (cons z l1) l2)) (cons (first (append (cons z l1) l2))

(append (rest (append (cons z l1) l2)) l3)) ] )

PRIM, ERED, EKONG

≡ (cond [#false l3]

[ (cons? (append (cons z l1) l2)) (cons (first (append (cons z l1) l2))

(append (rest (append (cons z l1) l2)) l3)) ] )

COND-false, ERED

≡ (cond [ (cons? (append (cons z l1) l2)) (cons (first (append (cons z l1) l2))

(append (rest (append (cons z l1) l2)) l3)) ] )

STRUCT-predtrue, EKONG

≡ (cond [#true (cons (first (append (cons z l1) l2))

(append (rest (append (cons z l1) l2)) l3)) ] )

COND-true, ERED

≡ (cons (first (append (cons z l1) l2))

(append (rest (append (cons z l1) l2)) l3))

Teilaufgabe (c), EKONG

≡ (cons (first (cons z (append l0 l2)))

(append (rest (cons z (append l0 l2))) l3))

PRIM, ERED, EKONG

≡ (cons z (append (append l0 l2) l3))

ETRANS

(append l0 (append l2 l3)) ≡ (append (append l0 l2) l3) ≡ (cons z (append (append l0 l2) l3))

Damit ist bewiesen: (append l1 (append l2 l3)) ≡ (append (append l1 l2) l3) für alle Listen l1, l2, l3.