

Übungen zu “Grundlagen der Programm- und Systementwicklung”

Aufgabe 1 Lösungsvorschlag

Es ergibt sich folgende algebraische Spezifikation für beblätterte Binärbäume:

SPEC SEQ =

```
{  based_on Bool,
    sort Seq  $\alpha$ ,
     $\langle \rangle$  : Seq  $\alpha$ ,
     $\langle - \rangle$  :  $\alpha \rightarrow$  Seq  $\alpha$ ,
     $\circ$  : Seq  $\alpha$ , Seq  $\alpha$  : Seq  $\alpha$ ,
    iseseq : Seq  $\alpha \rightarrow$  Bool,
    first, last : Seq  $\alpha \rightarrow \alpha$ ,
    rest, head : Seq  $\alpha \rightarrow$  Seq  $\alpha$ ,
    Seq  $\alpha$  generated_by  $\langle \rangle, \langle - \rangle, \circ$ ,
    iseseq( $\langle \rangle$ ) = true,
    iseseq( $\langle a \rangle$ ) = false,
    iseseq( $t_1 \circ t_2$ ) = and(iseseq( $t_1$ ), iseseq( $t_2$ )),
    first( $\langle a \rangle$ ) = a,
    iseseq( $t_1$ ) = false  $\Rightarrow$  first( $t_1 \circ t_2$ ) = first( $t_1$ ),
    iseseq( $t_1$ ) = true  $\Rightarrow$  first( $t_1 \circ t_2$ ) = first( $t_2$ )
    last( $\langle a \rangle$ ) = a,
    iseseq( $t_2$ ) = false  $\Rightarrow$  last( $t_1 \circ t_2$ ) = last( $t_2$ ),
    iseseq( $t_2$ ) = true  $\Rightarrow$  last( $t_1 \circ t_2$ ) = last( $t_1$ ),
    rest( $\langle a \rangle$ ) =  $\langle \rangle$ ,
    iseseq( $t_1$ ) = false  $\Rightarrow$  rest( $t_1 \circ t_2$ ) = rest( $t_1$ )  $\circ$   $t_2$ 
    iseseq( $t_1$ ) = true  $\Rightarrow$  rest( $t_1 \circ t_2$ ) =  $t_1 \circ$  rest( $t_2$ )
    head( $\langle a \rangle$ ) =  $\langle \rangle$ ,
    iseseq( $t_2$ ) = false  $\Rightarrow$  head( $t_1 \circ t_2$ ) =  $t_1 \circ$  head( $t_2$ )
    iseseq( $t_2$ ) = true  $\Rightarrow$  head( $t_1 \circ t_2$ ) = head( $t_1$ )  $\circ$   $t_2$ 
}
```

*** Alternativ: zB: iseseq(rest($\langle a \rangle$)) = true ***
*** Ohne derartige Forderungen erfüllt auch *id* die Spezifikation zB. von *rest* ***

Aufgabe 2 Lösungsvorschlag

Die ganzen Zahlen INTEGER

SPEC INT =

```
{  based_on Bool,
    sort Int,
    0 : Int,
    succ, pred : Int → Int,
    ≥ : Int, Int → Int,
    Int generated_by 0, succ, pred,
    x ≥ x = true,
    x ≥ succ(x) = false,
    x ≥ y = false ⇒ x ≥ succ(y) = false,
    x ≥ y = true ⇒ x ≥ pred(y) = true,
    pred(succ(x)) = x,
    succ(pred(x)) = x,
    +, *, - : Int, Int → Int,
    0 + y = y,
    succ(x) + y = succ(x+y),
    pred(x) + y = pred(x+y),
    x - 0 = x,
    x - succ(y) = pred(x-y),
    x - pred(y) = succ(x-y),
    x * 0 = 0,
    x * succ(y) = (x*y)+x,
    x * pred(y) = (x*y)-x }
```

EXTENDEDSEQ

SPEC EXTENDEDSEQ =

```
{  based_on SEQ, NAT
    # : Seq α → Nat,
    # ⟨⟩ = 0,
    # ⟨a⟩ = succ(0),
    # x ∘ y = (# x) + (# y) }
```

STACK

SPEC STACK =
{ **based_on** Bool,
 sort Stack α ,
 estack : Stack α ,
 append : α , Stack α : Stack α ,
 isestack : Stack $\alpha \rightarrow$ Bool,
 first : Stack $\alpha \rightarrow \alpha$,
 rest : Stack $\alpha \rightarrow$ Stack α ,
 Stack α **generated_by** estack, append,
 isestack(estack) = true,
 isestack(append(d, s)) = false,
 first(append(d, s)) = s,
 rest(append(d, s)) = d, }

Double-Ended Queue, DEQUEUE

SPEC DEQUEUE =
{ **based_on** Bool,
 sort Deq α ,
 edeq : Deq α ,
 stock : Deq α , α : Deq α ,
 append : α , Deq α : Deq α ,
 isedeq : Deq $\alpha \rightarrow$ Bool,
 first, last : Deq $\alpha \rightarrow \alpha$,
 rest, prefix : Deq $\alpha \rightarrow$ Deq α ,
 Deq α **generated_by** edeq, append
 Deq α **generated_by** edeq, stock
 isedeq(edeq) = true,
 isedeq(stock(q, d)) = false,
 isedeq(append(d, q)) = false,
 stock(edeq, d) = append(d, edeq),
 append(d1, stock(q, d2)) = stock(append(d1, q), d2),
 first(append(d, q)) = d,
 last(stock(q, d)) = d,
 rest(append(d, q)) = q,
 prefix(stock(q, d)) = q }

TREE

SPEC TREE =

{ **based_on** Bool,
 sort Tree α ,
 entree : Tree α ,
 isetree : Tree $\alpha \rightarrow$ Bool,
 root : Tree $\alpha \rightarrow \alpha$,
 cons : Tree $\alpha, \alpha, \text{Tree } \alpha \rightarrow \text{Tree } \alpha$,
 left, right : Tree $\alpha \rightarrow \text{Tree } \alpha$,
 Tree α **generated_by** entree, cons,
 isetree(entree) = true,
 isetree(cons(t1, d, t2)) = false,
 left(cons(t1, d, t2)) = t1,
 right(cons(t1, d, t2)) = t2,
 root(cons(t1, d, t2)) = d }

Endliche Mengen, FINSET

SPEC FINSET =

{ **based_on** Bool,
 sort Fset α ,
 eset : Fset α ,
 add, del : Fset $\alpha, \alpha \rightarrow \text{Fset } \alpha$,
 iseset : Fset $\alpha \rightarrow$ Bool,
 isel : Fset $\alpha, \alpha \rightarrow$ Bool,
 Fset α **generated_by** eset, add,
 iseset(eset) = true,
 iseset(add(s, x)) = false,
 isel(eset, x) = false,
 isel(add(s, x), x) = true,
 isel(s, x) = true \Rightarrow isel(add(s, y), x) = true,
 $x \neq y \Rightarrow$ isel(add(s, y), x) = isel(s, x)
 del(eset, x) = eset,
 del(add(s, x), x) = del(s, x)
 $x \neq y \Rightarrow$ del(add(s, y), x) = add(del(s, x), y)
 add(add(s, x), y) = add(add(s, y), x)
 isel(s, x) = true \Rightarrow add(s, x) = s }

Multimengen, MULTISSET

SPEC MULTISSET =

```
{  based_on Bool,
    sort Mset  $\alpha$ ,
    emset : Mset  $\alpha$ ,
    add, del : Mset  $\alpha$ ,  $\alpha \rightarrow$  Mset  $\alpha$ ,
    isemset : Mset  $\alpha \rightarrow$  Bool,
    isel : Mset  $\alpha$ ,  $\alpha \rightarrow$  Bool,
    Mset  $\alpha$  generated_by emset, add,
    isemset(emset) = true,
    isemset(add(s, x)) = false,
    isel(emset, x) = false,
    isel(add(s, x), x) = true,
    isel(s, x) = true  $\Rightarrow$  isel(add(s, y), x) = true,
     $x \neq y \Rightarrow$  isel(add(s, y), x) = isel(s, x)
    del(emset, x) = emset,
    del(add(s, x), x) = s
     $x \neq y \Rightarrow$  del(add(s, y), x) = add(del(s, x), y)
    add(add(s, x), y) = add(add(s, y), x) }
```

Graphen, GRAPH

SPEC GRAPH =

```
{  based_on FINSET,
    sort Graph  $\alpha$ ,
    egraph : Graph  $\alpha$ ,
    put : Graph  $\alpha$ ,  $\alpha$ , Fset  $\alpha \rightarrow$  Graph  $\alpha$ ,
    neighbours : Graph  $\alpha$ ,  $\alpha \rightarrow$  Fset  $\alpha$ ,
    Graph  $\alpha$  generated_by egraph, put,
    neighbours(egraph, x) = eset,
    neighbours(put(g, x, s)) = s,
     $x \neq y \Rightarrow$  neighbours(put(g, x, s), y) = neighbours(g, y) }
```