Übung 5 – Hausaufgaben

Formale Techniken in der Software-Entwicklung

Christian Kroiß

QID-Queue (2/2)

```
fmod QID-QUEUE is
  including QID .

  sort Queue .

  op empty : -> Queue [ctor] .
  op enq : Queue Qid -> Queue [ctor] .

  op top : Queue -> Qid .
  op deq : Queue -> Queue .

  op undefined : -> [Qid] [ctor] .
  op undefined : -> [Queue] [ctor] .
```

QID-Queue (2/2)

```
var x : Qid .
  var q : Queue .

  eq top(empty) = undefined              [label top-empty] .

  eq top(enq(empty,x)) = x               [label top-enq1] .
  eq top(enq(q,x)) = top(q)              [label top-enq2] .

  eq deq(empty) = undefined              [label deq-empty] .

  eq deq(enq(empty,x)) = empty           [label deq-enq1] .
  eq deq(enq(q,x)) = enq(deq(q),x)       [label deq-enq2] .
endfm
```

Aufgabe 4-1

Ausprobieren:

```
red enq(enq(empty, 'a), 'b) .
red enq(enq(enq(empty, 'a), 'b), 'c) .
red top(enq(enq(enq(empty, 'a), 'b), 'c)) .
red deq(enq(enq(enq(empty, 'a), 'b), 'c)) .
```

Aufgabe 4-1

Queue by List (1/2):

```
fmod QID-QUEUE-BY-LIST is
  including LIST{Qid} .

  var q : List{Qid} .
  var x  : Qid .

  op enq : List{Qid} Qid -> List{Qid} [ctor] .
  eq enq(q, x) = q x .
endfm
```

Queue by List (2/2):

```
fmod QID-QUEUE-IMPL is
  including QID-QUEUE-BY-LIST * (
    sort List{Qid} to Queue,
    sort NeList{Qid} to NeQueue,
    op nil to empty,
    op head to top,
    op tail to deq
  ) .
endfm
```

Aufgabe 4-1 a)

4-1 a) Beweisen Sie, dass jedes Modell von QID-QUEUE-BY-LIST ein Modell von `QID-QUEUE` ist.

Zu zeigen: `QID-QUEUE-IMPL` ist eine FRI-Implementierung von `QID-QUEUE`

Wegen

```
Rep : Queue -> Bool . Rep = true .
_ ~ _ : Queue Queue -> Bool .
P ~ Q = P == Q .
```

Bleibt zu zeigen: Aus den Axiomen in `QID-QUEUE-BY-LIST` folgen die Axiome in `QID-QUEUE`

Aufgabe 4-1 a)

[top-empty] und [deq-empty]:

- `top(empty) = undefined [label top-empty]`
  Zu zeigen: `head(nil) = undefined`
  Gilt, da es für `nil` keine Regel im Modul `LIST` gibt.
- `deq(empty) = undefined [label deq-empty]`
  Zu zeigen: `tail(nil) = undefined`
  Gilt, da es für `nil` keine Regel im Modul `LIST` gibt.

Aufgabe 4-1 a)

[deq-enq1] und [top-enq2]

- `deq(enq(empty,x)) = empty [label deq-enq2]`
  Zu zeigen: `tail(enq(nil,x)) = nil`
  l.s. = `tail(nil x) = tail(x) = tail(x nil) = nil`
- `deq(enq(q,x)) = enq(deq(q),x) [label deq-enq2]`
  Zu zeigen: `tail(enq(q,x)) = enq(deq(q),x)`
  - l.s. = `tail(q x) = tail(e es x) = es x`
    (`q = e es`, n. Vorr.)
  - r.s. = `enq(tail(e es),x) = enq(es,x) = es x` = l.s.

Aufgabe 4-1 b)

4-1 b) Simulieren Sie Schlangen durch ein Array mit zwei Zeigern. Geben Sie dazu eine Spezifikation QID-QUEUE-BY-ARRAYPOINTER, ggf. eine geeignete Umbenennung, ein Repräsentationsprädikat Rep und eine Kongruenz $\sim$ an.

Aufgabe 4-1 b)

```
fmod QID-QUEUE-BY-ARRAYPOINTER is
  protecting NAT-ARRAY{Qid} .
  protecting NAT .

  sort ArrayQueue .
  op queue : NatArray{Qid} Nat Nat -> ArrayQueue [ctor] .
  op empty : -> ArrayQueue .
  --- constructors different from QUEUE!
  op enq : ArrayQueue Qid -> ArrayQueue .
  op top : ArrayQueue -> [Qid] .
  op deq : ArrayQueue -> [ArrayQueue] .

  op undefined : -> [Qid] [ctor] .
  op undefined : -> [ArrayQueue] [ctor] .
```

Aufgabe 4-1 b)

```
var x : Qid .
var A : NatArray{Qid} .
var Q : ArrayQueue .
vars front tail : Nat .
eq empty = queue(empty, 0, 0) .
eq enq(queue(A, front, tail), x) = queue(insert(tail,x,A), front,
       tail + 1) .
ceq top(queue(A, front, tail)) = A[front]
      if front < tail .
eq top(Q) = undefined [owise] .

ceq deq(queue(A, front, tail)) = queue(A, front + 1, tail)
      if front < tail .
eq deq(Q) = undefined [owise] .

endfm
```

Rep und ∼ in Pseudo-Maude

```
eq Rep_E(x:Qid) = true .
ceq Rep_Q(queue(A,front,tail)) =
      (front <= tail)
      and forall(k : Nat . k < sd(tail,front)
            => isDefined(A[front+k])) .
eq x ~E x' = (x == x') .
eq queue(A,front,tail) ~Q queue(A',front',tail') =
      sd(tail,front) == sd(tail',front')
      and forall(k : Nat . k < sd(tail,front)
            => A[front+k] == A'[front'+k]) .
```

Aufgabe 4-1 c)

4-1 c) Weisen Sie die Kongruenzeigenschaft von $\sim$ für die Operation `enq` nach.

Behauptung: Für alle Queues Q, Q' und alle Elemente x gilt

$$Q \sim Q' \Rightarrow enq(Q, x) \sim enq(Q', x)$$

Beweis: Sei $Q := queue(A, f, t)$ und $Q' := queue(A', f', t')$.

Sei

$$Q_1 = enq(Q, x) \quad = queue(ins(t, x, A), f, t + 1)$$
$$Q'_1 = enq(Q', x) \quad = queue(ins(t', x, A'), f', t' + 1)$$

Aufgabe 4-1 c)

Zu zeigen: $Q_1 \sim Q_1'$. Es gilt $t_1 - f_1 = t_1' - f_1'$, also bleibt noch zu zeigen

$$\forall k \in Nat \;.\; k < (t+1) - f \Rightarrow ins(t, x, A)[f + k] = ins(t', x, A')[f' + k]$$

- Gilt (laut Vor.) für $k < (t - f)$ und A, A'. Aber $ins()$ verändert hier nichts.
- $k = t - f$: zu vergleichende Indizes sind $t = f + (t - f)$ und t'.
  Also ist noch zu zeigen:

$$ins(t, x, A)[t] = ins(t', x, A')[t']$$

- Gilt wegen Definition von insert.

Aufgabe 4-2)

Gegeben sei folgendes Modul:

```
fmod NATURAL is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _+_ : Nat Nat -> Nat .
  op _*_ : Nat Nat -> Nat .
  vars N M : Nat .
  eq N + 0 = N .
  eq N + s(M) = s(N + M) .
  eq N * 0 = 0 .
  eq N * s(M) = (N * M) + N .
endfm
```

Beweisen Sie, dass die Gleichungen terminieren. Verwenden Sie dazu die in der Vorlesung besprochene Lexikographische Pfadordnung.

Aufgabe 4-2)

Sei S eine endliche Signatur und $>$ eine Ordnung ihrer Funktionssymbole. Dann ist die **lexikographische Pfadordnung** $>_{lpo}$ über $T(\Sigma, V)$ rekursiv folgendermaßen definiert:

1) Seien $t \in T(\Sigma, V), x \in Var(t)$ und x verschieden von t, dann $t >_{lpo} x$.

2) $f(t_1, \ldots, t_n) >_{lpo} g(t'_1, \ldots, t'_m)$ wenn

entweder 2.1) $t_i >_{lpo} g(t'_1, \ldots, t'_m)$ für ein i,

oder 2.2) $f > g$ und $f(t_1, \ldots, t_n) >_{lpo} t'_j$, für alle j,

oder 2.3) $f = g, f(t_1, \ldots, t_n) >_{lpo} t'_j$, für alle j und es gibt ein i mit $t_j = t'_j$, für alle $j < i$, und $t_i >_{lpo} t'_i$.

Aufgabe 4-2)

$$
\begin{array}{ll}
(1) & +(N,0) = N \\
(2) & +(N,s(M)) = s(+(N,M)) \\
(3) & *(N,0) = 0 \\
(4) & *(N,s(M)) = +(*(N,M),N)
\end{array}
$$

Ordnung: $* > + > s > 0$

$$
\begin{array}{lll}
(1) & +(N,0) = N & \\
 & +(N,0) > N & [1] \\
(2) & + > s \quad \text{Zeige:} \quad +(N,s(M)) > +(N,M) & [2.2] \\
 & \quad + = + \quad \text{Zeige:} \quad N = N \wedge s(M) > M & [2.3] \\
 & \quad\quad s(M) > M & [1] \\
(3) & * > 0 & [2.2] \\
(4) & * > + \quad \text{Zeige:} \quad *(N,s(M)) > *(N,M) & \\
 & \quad\quad \wedge *(N,s(M)) > N & [2.2] \\
 & *(N,s(M)) > *(N,M) & [\text{analog zu } (2)] \\
 & *(N,s(M)) > N & [1]
\end{array}
$$