Übung 7 – Hausaufgaben + UPPAAL

Formale Techniken in der Software-Entwicklung

Christian Kroiß

Aufgabe 6-1

```
(omod COMM is
    protecting NAT-LIST .
    protecting QID .
    sort Nat? .
    subsort Nat < Nat? .
    subsort Qid < Oid .

    class Buffer | q : NatList, reader : Oid .
    class Sender | cell : Nat?, cnt : Nat, receiver : Oid .
    class Receiver | cell : Nat?, cnt : Nat .

    op mt : -> Nat? .

    msg to_from_val_cnt_ : Oid Oid Nat Nat -> Msg .
    op start : Oid Oid Oid NatList -> Configuration .
```

Aufgabe 6-1

```
vars E N M : Nat .
    var L : NatList .
    vars X Y Z : Oid .

    --- Beachte: Nicht immer tauchen alle Attribute auf

    rl [read] : < X : Buffer | q : L E, reader : Y >
                < Y : Sender | cell : mt, cnt : N >
        => < X : Buffer | q : L, reader : Y >
           < Y : Sender | cell : E, cnt : N + 1 > .
    rl [send] : < Y : Sender | cell : E, cnt : N, receiver : Z >
        => < Y : Sender | cell : mt, cnt : N, receiver : Z >
           to Z from Y val E cnt N .
    rl [receive] : < Z : Receiver | cell : mt, cnt : N >
                   to Z from Y val E cnt M
        => < Z : Receiver | cell : E, cnt : N + 1 > .
```

Aufgabe 6-1

```
eq start(X, Y, Z, L) =
        < X : Buffer | q : L, reader : Y >
        < Y : Sender | cell : mt, cnt : 0, receiver : Z >
        < Z : Receiver | cell : mt, cnt : 0 > .
endom)
```

Aufgabe 6-1

a) Geben Sie eine Regel `write` an, mit deren Hilfe der Empfang gepuffert wird.

Gebraucht wird zusätzlich ein Puffer für empfangene Nachrichten:

```
class RBuffer | q : NatList .
```

Die Regel write soll nun jeweils den Wert im Speicher des Receivers nehmen und in den Rbuffer schreiben:

```
rl [write] : < R : Receiver | cell : E, rbuffer : RB >
                < RB : RBuffer | q : Es >
          => < R : Receiver | cell : mt >
             < RB : RBuffer | q : E Es > .
```

Aufgabe 6-1

b) Modifizieren Sie das Modul so, dass die Pakete immer in der richtigen Reihenfolge in den Empfangspuffer gestellt werden. Beschreiben Sie in einem Kommentar, wie Sie das bewerkstelligen.

Es reicht bei receive darauf zu achten, dass die Nachricht mit der korrekten Sequenznummer gelesen wird:

```
crl [receive] : < R : Receiver | cell : mt, cnt : N >
                to R from S val E cnt M
     => < R : Receiver | cell : E, cnt : N + 1 >
     if M = N + 1 .
```

Aufgabe 6-2

a) nicht objektorientiertes Systemmodul

```
mod HANOI is
  protecting NAT .
  protecting BOOL .
  sorts Tower Disc Config .
  sort Disc < Tower .
  op empty : -> Tower [ctor] .
  op _ _ : Tower Tower -> Tower [assoc comm ctor id: empty] .
  op <_,_,_> : Tower Tower Tower -> Config [ctor] .
  op d(_) : Nat -> Disc [ctor] .
  op init_tower : Nat -> Tower .
  op init : Nat -> Config .
  op maxdisc : Tower -> Nat .
```

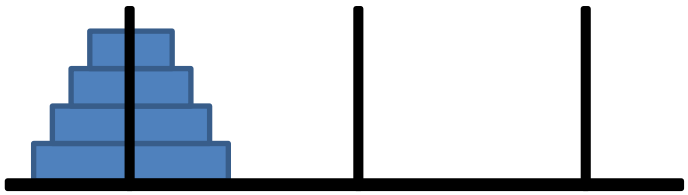
Aufgabe 6-2

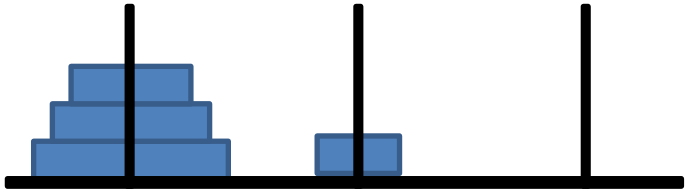
```
var N : Nat .
vars T1 T2 T3 : Tower .
var D : Disc .

eq init_tower(1) = d(1) .
ceq init_tower(N) = init_tower(sd(N,1)) d(N) if N > 1 .
eq init(N) = < init_tower(N), empty, empty > .

eq maxdisc(empty) = 0 .
eq maxdisc(T1 d(N)) = max(maxdisc(T1),N) .
```

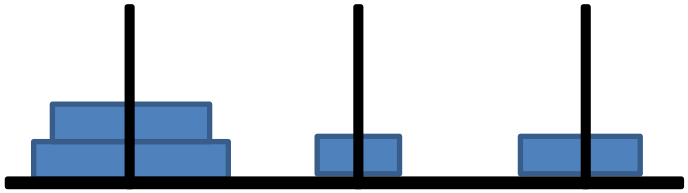
Aufgabe 6-2

```
crl [A_B] : < T1 d(N), T2, T3 > => < T1, T2 d(N), T3 >
      if maxdisc(T1) < N /\ maxdisc(T2) < N .
crl [A_C] : < T1 d(N), T2, T3 > => < T1, T2, T3 d(N) >
      if maxdisc(T1) < N /\ maxdisc(T3) < N .
crl [B_A] : < T1, T2 d(N), T3 > => < T1 d(N), T2, T3 >
      if maxdisc(T2) < N /\ maxdisc(T1) < N .
crl [B_C] : < T1, T2 d(N), T3 > => < T1, T2, T3 d(N) >
      if maxdisc(T2) < N /\ maxdisc(T3) < N .
crl [C_A] : < T1, T2, T3 d(N) > => < T1 d(N), T2, T3 >
      if maxdisc(T3) < N /\ maxdisc(T1) < N .
crl [C_B] : < T1, T2, T3 d(N) > => < T1, T2 d(N), T3 >
      if maxdisc(T3) < N /\ maxdisc(T2) < N .

endm
```

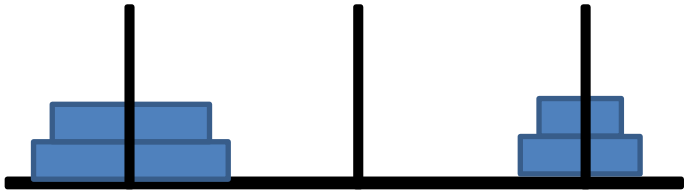
Aufgabe 6-2

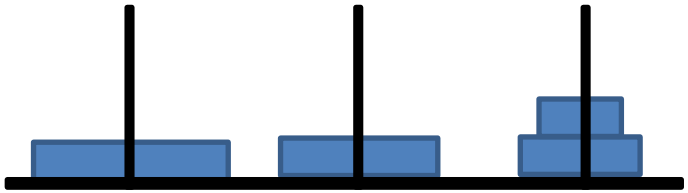


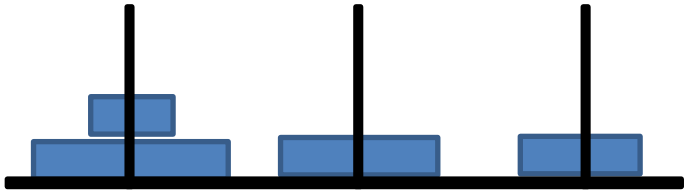




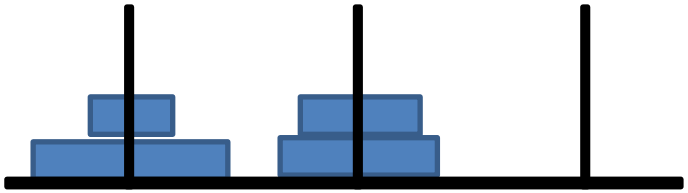
1) A→B
2) A → C
3) B → C
4) A → B
5) C → A

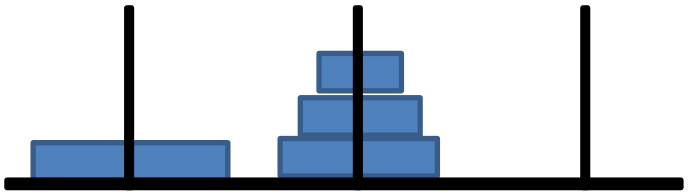


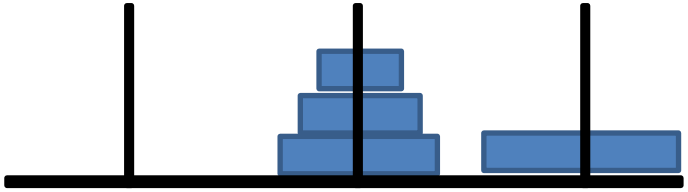




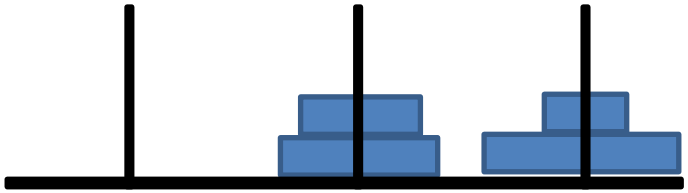
Aufgabe 6-2

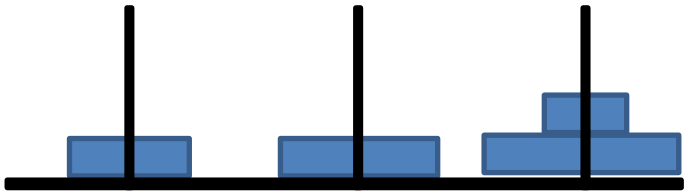


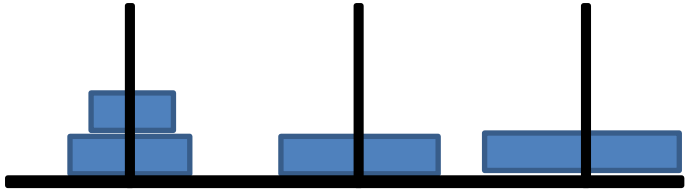




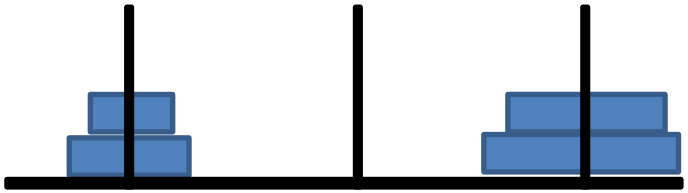
6) C → B
7) A → B
8) A → C
9) B → C
10) B → A
11) C → A



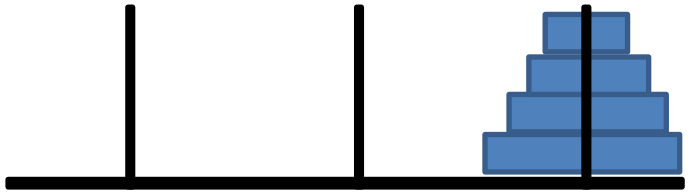


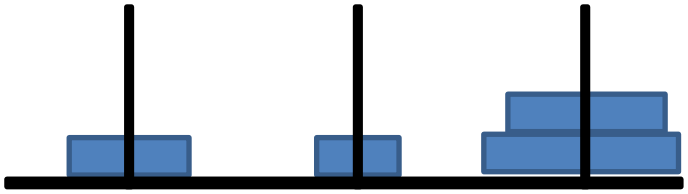


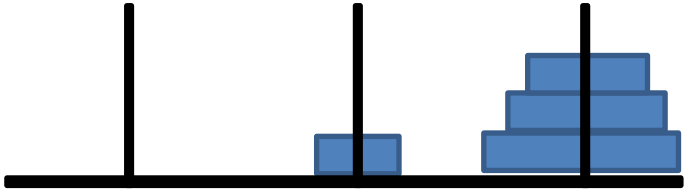
Aufgabe 6-2



12) B → C
13) A → B
14) A → C
15) B → C







Aufgabe 6-2

b) Objektorientiertes Modul

```
(omod OHANOI is
        protecting QID .
        protecting INT .

        protecting SORTABLE-LIST{Int<} .

        subsort Qid < Oid .

        class Tower | c : List{Int<} .
```

Aufgabe 6-2

```
msg move : Oid Oid -> Msg .
op initial : -> Configuration .

eq initial = < 'A : Tower | c : 1 2 3 4 >
       < 'B : Tower | c : nil > < 'C : Tower | c : nil > .

op max : List{Int<} -> Nat .

vars A B : Oid .
var N : Nat .
vars L1 L2 : List{Int<} .

eq max(nil) = 0 .
eq max(L1 N) = max(max(L1),N) .
```

Aufgabe 6-2

```
crl [move] :
        move(A,B)
        < A : Tower | c : L1 N >
        < B : Tower | c : L2 >
        =>
        < A : Tower | c : L1 >
        < B : Tower | c : L2 N >
        if max(L1) < N /\ max(L2) < N .

endom)
```

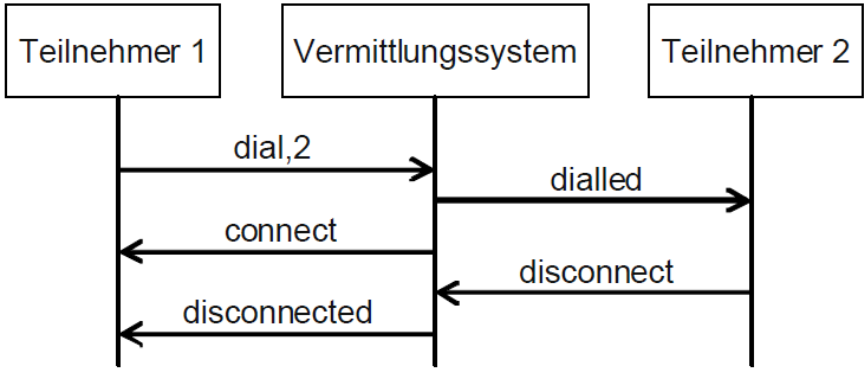
UPPAAL-Beispiel aus Buch „Formale Modelle der Softwareentwicklung“: Telefonsystem

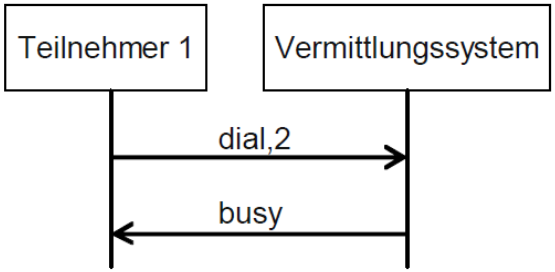
Modelliert werden soll ein Telefonsystem mit N Teilnehmern, die miteinander durch ein Vermittlungssystem verbunden werden können.

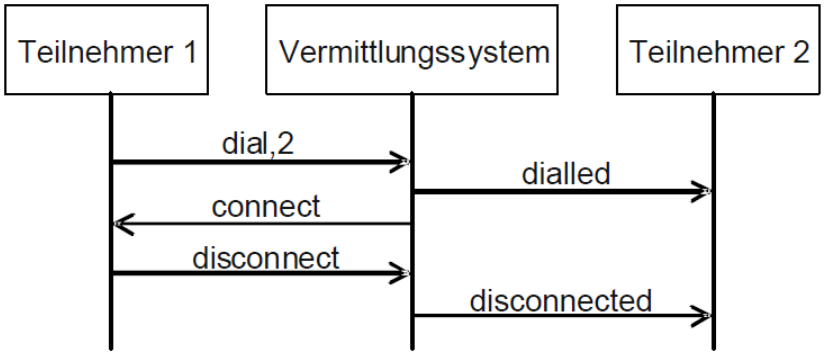
Anforderungen:

- jeder Teilnehmer hat immer wieder die Möglichkeit, jeden anderen Teilnehmer zu erreichen
- Es werden minimal 4 und maximal 7 Zeiteinheiten benötigt, um festzustellen, dass ein Teilnehmer nicht erreichbar ist.
- Ein Telefonat (zwischen `connect` und `disconnect`, bzw. `dialled` und `disconnect`) dauert minimal 20 und maximal 40 Zeiteinheiten. Handelt es sich z. B. um Gratisanrufmöglichkeiten, gibt es in den ersten 20 Einheiten Werbung und dann 20 Einheiten zur freien Verfügung.

Mögliche Telefonatsabläufe







Modell: siehe TelSys.xml