Kapitel 3

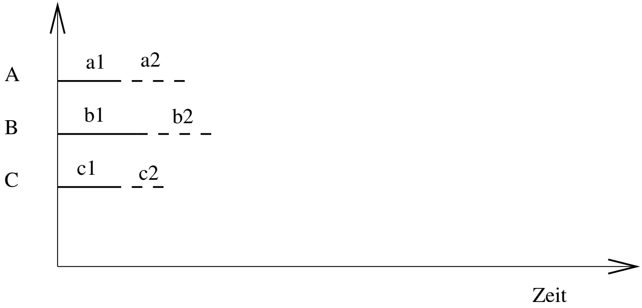
Parallele Prozesse

Prof. Dr. Rolf Hennicker

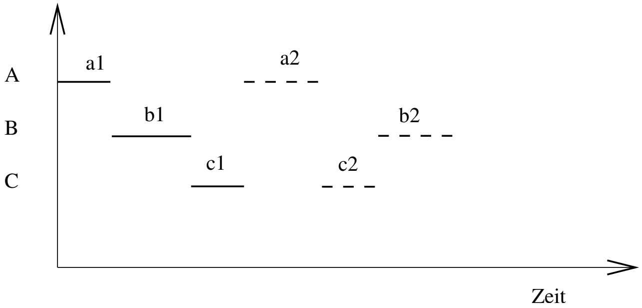
08.05.2014

3.1 Modellierung paralleler Prozesse

Echte Parallelität



Quasi-(Pseudo-)Parallelität



Die Aktionen der einzelnen Prozesse werden bei Quasi-(Pseudo-)Parallelität miteinander "verzahnt" ausgeführt. Wir sprechen dann von **"Interleaving"**.

Beachte:

- Alle möglichen Verzahnungen müssen berücksichtigt werden.
- Die Reihenfolge der Aktionen <u>eines</u> Prozesses ist dieselbe wie bei echter Parallelität.

Die parallele Komposition von Prozessen wird im Folgenden durch Interleaving modelliert.

6. Parallele Komposition von Prozessen

Bisher wurden 5 grundlegende Konstrukte für FSP Prozessausdrücke definiert.

Definition:

Sind $E_1, \ldots, E_n$ Prozessausdrücke, dann ist

$$(E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n)$$

ein Prozessausdruck (*parallele Komposition* von $E_1, \ldots, E_n$)
mit $FV((E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n)) = FV(E_1) \cup \ldots \cup FV(E_n)$.

Wirkung:

Die (disjunkten) Aktionen von $E_1, \ldots, E_n$ werden verzahnt ausgeführt.

Deklaration paralleler Prozesse:

Seien $E_1, \ldots, E_n$ Prozessausdrücke und sei $P \in PID$ ein Prozessidentifikator mit $P \notin FV((E_1 \,||\, \ldots \,||\, E_n))$. Eine Deklaration mit paralleler Komposition wird dann angegeben durch

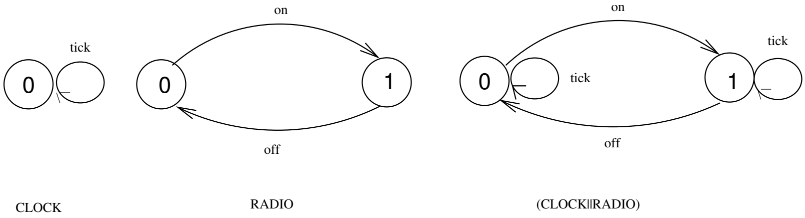
$$||\, P = (E_1 \,||\, \ldots \,||\, E_n).$$

Achtung: In rekursiven Deklarationen und in rekursiven Prozessausdrücken darf der Paralleloperator nicht verwendet werden.

Beispiel:

$$\begin{aligned}
&\text{CLOCK} = (\text{tick} \rightarrow \text{CLOCK}).\\
&\text{RADIO} \ = (\text{on} \rightarrow \text{off} \rightarrow \text{RADIO}).\\
&\|\text{CLOCK_RADIO} = (\text{CLOCK}\|\text{RADIO}).
\end{aligned}$$

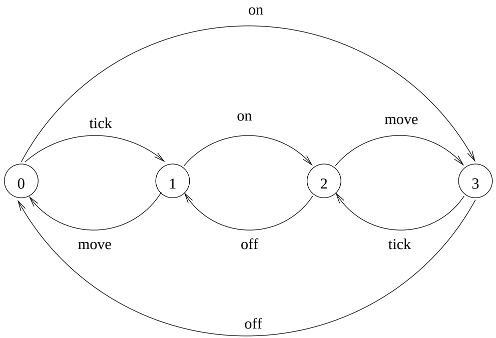
Zugehörige LTSe:



Beispiel:

$\text{CLOCK2} = (\text{tick} \rightarrow \text{move} \rightarrow \text{CLOCK2}).$
$\text{RADIO} = (\text{on} \rightarrow \text{off} \rightarrow \text{RADIO}).$
$\|\text{CLOCK2\_RADIO} = (\text{CLOCK2}\|\text{RADIO}).$

lts(CLOCK2):



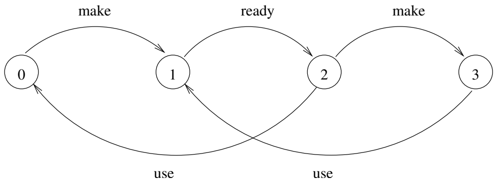
Prozessinteraktionen

- Prozessinteraktionen werden durch *gemeinsame* Aktionen ("shared actions") modelliert.
- Parallele Prozesse, die gemeinsame Aktionen haben, müssen diese gemeinsam ausführen, d.h. sie müssen sich synchronisieren.
- Die Synchronisation schränkt die möglichen Abläufe der parallelen Komposition ein.

Beispiel:

$$
\begin{aligned}
&\text{MAKER} = (\text{make} \rightarrow \text{ready} \rightarrow \text{MAKER}).\\
&\text{USER} \quad = (\text{ready} \rightarrow \text{use} \rightarrow \text{USER}).\\
&\|\text{MAKER\_USER} = (\text{MAKER} \parallel \text{USER}).
\end{aligned}
$$

Zugehöriges LTS:

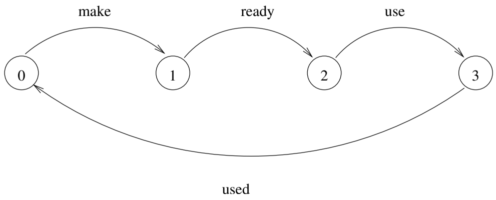


Variante:

MAKER produziert erst dann weiter, wenn der USER die Benutzung bestätigt hat.

$$\begin{array}{ll}\mathrm{MAKER2} & = (\mathrm{make} \to \mathrm{ready} \to \mathrm{used} \to \mathrm{MAKER2}).\\ \mathrm{USER2} & = (\mathrm{ready} \to \mathrm{use} \to \mathrm{used} \to \mathrm{USER2}).\\ \multicolumn{2}{l}{\|\mathrm{MAKER\_USER2} = (\mathrm{MAKER2} \parallel \mathrm{USER2}).}\end{array}$$

Zugehöriges LTS:



7. Umbenennung von Aktionen

Die Umbenennung von Aktionen dient (vor allem)

- zur Erstellung verschiedener Kopien eines Prozesses,
- als Hilfsmittel zur Synchronisation paralleler Prozesse.

Allgemeine Voraussetzung: $\mathsf{ACT} = \mathsf{Labels} \cup \{\tau\}$

Definition:

Sei E ein (evt. paralleler) Prozessausdruck und seien $a_1, \ldots, a_k$ und $n_1, \ldots, n_k$ Aktionsnamen verschieden von τ. Dann ist

$$E\{n_1/a_1, \ldots, n_k/a_k\}$$

ein Prozessausdruck ("Relabelling") mit $FV(E\,\{n_1/a_1, \ldots, n_k/a_k\}) = FV(E)$.

Wirkung:

Im LTS von E werden die Aktionsnamen $a_1, \ldots, a_k$ ersetzt durch $n_1, \ldots, n_k$.

Erstellung von Prozess-Kopien durch Umbenennung

Beispiel:

CLIENT $=$ (call $\rightarrow$ wait $\rightarrow$ continue $\rightarrow$ CLIENT).
$\|$TWOCLIENTS $=$ (a:CLIENT $\|$ b:CLIENT).

Dabei ist a:CLIENT eine abkürzende Schreibweise für
CLIENT{a.call/call, a.wait/wait, a.continue/continue}.

Man nennt dies **Process Labelling**.

Abkürzende Schreibweisen für parallele Kompositionen von Prozesskopien

Sei `range ID = 1..N`

Die Ausdrücke

```
a[i:1 ...N]:E
a[ID]:E
```

bezeichnen alle den Prozess

$(\texttt{a[1]:E} \parallel \ldots \parallel \texttt{a[N]:E})$

Der Bezeichner a kann auch weggelassen werden.
Zum Beipiel bezeichnet `[ID]:E` den Prozess

$(\texttt{[1]:E} \parallel \ldots \parallel \texttt{[N]:E})$

Falls E ein Prozessausdruck ist, in dem der Index i vorkommt, dann steht

`forall[i:1 ...N] E` für $(\texttt{E[1/i]} \parallel \ldots \parallel \texttt{E[N/i]})$

wobei `E[1/i]` die Substitution von `i` in `E` durch `1` bezeichnet, usw.

Synchronisation von Prozessen durch Umbenennung

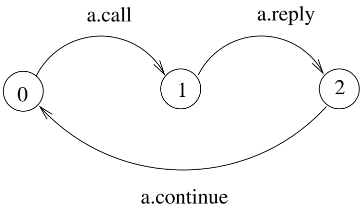
$$(E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n) \,/\, \{n_1/a_1, \ldots, n_k/a_k\} =_{def}$$
$$(E_1 \,\{n_1/a_1, \ldots, n_k/a_k\} \,||\, \ldots \,||\, E_n \,\{n_1/a_1, \ldots, n_k/a_k\})$$

Beispiel:

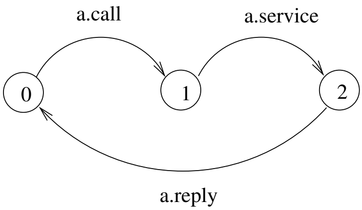
CLIENT = (call $\rightarrow$ wait $\rightarrow$ continue $\rightarrow$ CLIENT).
SERVER = (request $\rightarrow$ service $\rightarrow$ reply $\rightarrow$ SERVER).
$\|$CLIENT_SERVER = (CLIENT $\|$ SERVER) / {call/request, reply/wait}.

Beispiel für Synchronisation von Prozess-Kopien:

$\|$TWOCLIENTS_SERVER $=$ (a:CLIENT $\|$ b:CLIENT $\|$ a:SERVER $\|$ b:SERVER) / $\{$a.call/a.request, b.call/b.request, a.reply/a.wait, b.reply/b.wait$\}$.

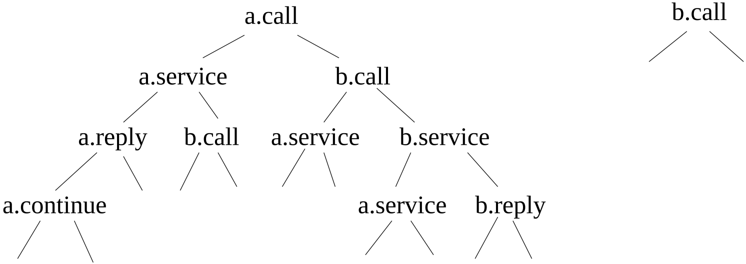


(a:CLIENT)$\{$... a.reply/a.wait ...$\}$



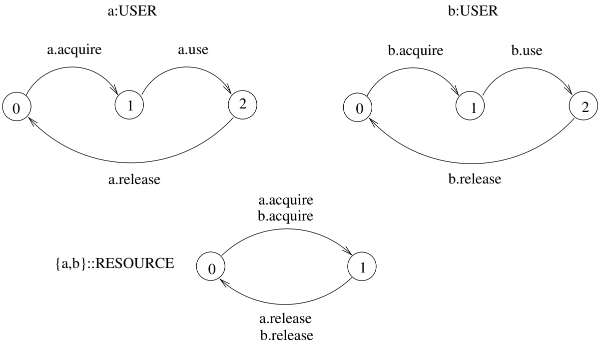
(a:SERVER)$\{$a.call/a.request ...$\}$

Abläufe:


a.call
a.service
b.call
a.reply
b.call
a.service
b.service
a.continue
a.service
b.reply
b.call

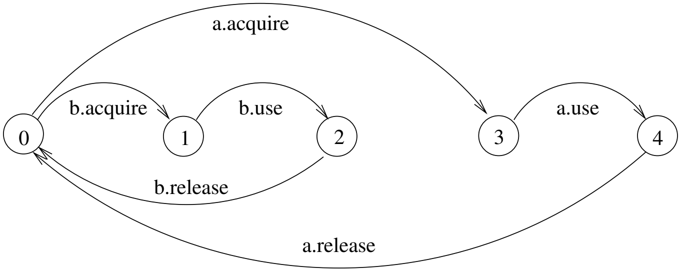
Beispiel (Resource-Sharing):

USER $=$ (acquire $\to$ use $\to$ release $\to$ USER).
RESOURCE $=$ (acquire $\to$ release $\to$ RESOURCE).
$\|$RESOURCE_SHARE $=$ (a:USER $\|$ b:USER $\|$ {a,b}::RESOURCE).



`{a,b}::RESOURCE` steht für die Umbenennung
`RESOURCE{{a.aquire,b.aquire}/aquire,{a.release,b.release}/release}`

RESOURCE_SHARE



8. Verbergen von Aktionen

Das Verbergen von Aktionen ("Hiding") dient zur Abstraktion von Aktionen, die unter einem bestimmten Gesichtspunkt "nicht relevant" (häufig im Sinne von nicht beobachtbar") sind. Dadurch kann man das "Black-Box" Verhalten von Komponenten modellieren.

Definition:

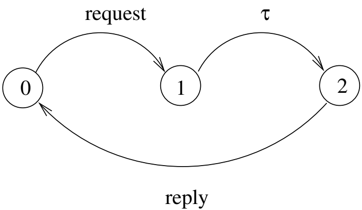
Sei E ein (event. paralleler) Prozessausdruck und sei H $\subseteq$ Labels eine Menge von Aktionsnamen ($\neq \tau$).
Dann ist E $\setminus$ H ein Prozessausdruck ("Hiding") mit FV(E $\setminus$ H) = FV(E).

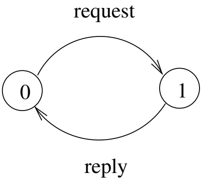
Wirkung:

Die Aktionen aus H werden verborgen und im LTS von E in eine spezielle Aktion τ (tau) umbenannt. τ heißt "unsichtbare" (unbeobachtbare, stille, interne) Aktion.

Beispiel:

$\|$SERVER2 $=$ SERVER $\setminus$ $\{$service$\}$.





Neben dem LTS kann auch das *minimale, beobachtbar äquivalente* LTS berechnet werden.

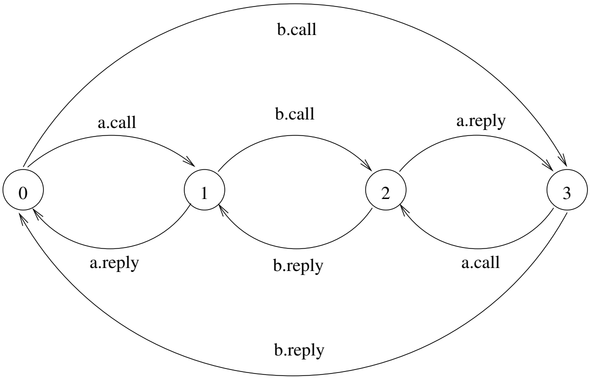
Bemerkung:

- Häufig wird "Hiding" nach der parallelen Komposition angewandt, um von der Komplexität eines parallelen Systems zu abstrahieren.
- Wird es vorher angewandt, dann darf bei der parallelen Komposition <u>nicht</u> bzgl. τ synchronisiert werden; τ ist intern und daher keine gemeinsame Aktion parallel laufender Prozesse.

Beispiel:

$\|$TCLIENTS_SERVER =
TWOCLIENTS_SERVER $\setminus$ {{a,b}.continue, {a,b}.service}.

Bezüglich beobachtbarer Äquivalenz minimalisiertes LTS:



Schnittstellen-Operator

$$\mathsf{E}\,@\,\mathsf{I}$$

wobei E ein Prozessausdruck und I $\subseteq$ Labels eine Menge von Aktionen ($\neq \tau$) ist.

Wirkung:

Alle sichtbaren Aktionen von E, die nicht in I vorkommen, werden verborgen.

Bemerkungen:

- I heißt *Schnittstelle* ("Interface") des Prozesses.
- Schnittstellen werden meist zur Beschreibung der von einem komplexen (parallelen) System angebotenen Dienste verwendet, wobei gemeinsame Aktionen der Komponenten verborgen werden.
- Häufige Form von parallelen Prozessen mit Schnittstellen: $(\mathsf{P}\|\mathsf{Q})/\{\mathsf{neu}/\mathsf{alt}\}@\{\mathsf{a}_1,\ldots,\mathsf{a}_k\}$

Beispiel:

$(\mathsf{MAKER}\|\mathsf{USER})@\{\mathsf{make},\mathsf{use}\}$ entspricht $(\mathsf{MAKER}\|\mathsf{USER})\backslash\{\mathsf{ready}\}$

9. Alphabeterweiterung

Definition:

(1) Sei $\mathsf{T} = (\mathsf{S}, \mathsf{A}, \Delta, \mathsf{q})$ ein LTS.
Dann heißt die Menge $\alpha\mathsf{T} =_{def} \mathsf{A} \setminus \{\tau\}$ das *Alphabet* von T.

(2) Sei E ein Prozessausdruck mit $\mathsf{lts(E)} = \mathsf{T}$.
Dann heißt die Menge $\alpha\mathsf{E} =_{def} \alpha\mathsf{T}$ das *Alphabet* von E.

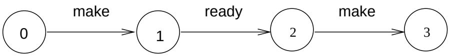
Beispiel: $\alpha((\mathsf{MAKER} \,\|\, \mathsf{USER})\,@\,\{\mathsf{make}, \mathsf{use}\}) = \{\mathsf{make}, \mathsf{use}\}$

Definition:

Sei E ein Prozessausdruck und $\mathsf{B} \subseteq$ Labels eine Menge von Aktionen. Dann ist die *Alphabeterweiterung* $\mathsf{E} + \mathsf{B}$ ein Prozessausdruck mit $\mathsf{FV(E+B)} = \mathsf{FV(E)}$.

Beispiel:

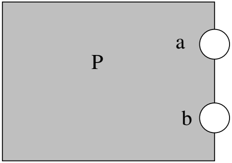
FMAKER = (make → ready → FMAKER) + {use}.
USER = (ready → use → USER).
||FMAKER_USER = (FMAKER || USER).



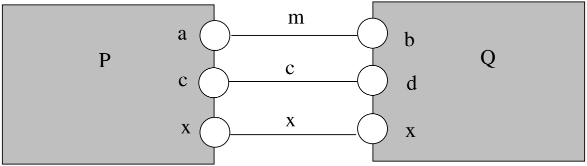
Strukturdiagramme

Strukturdiagramme zeigen den *strukturellen Aufbau* komplexer Systeme (Prozesse) mit Schnittstellen und (internen) Verbindungen zwischen Komponenten.

Strukturdiagramm eines Prozesses mit Alphabet {a,b}

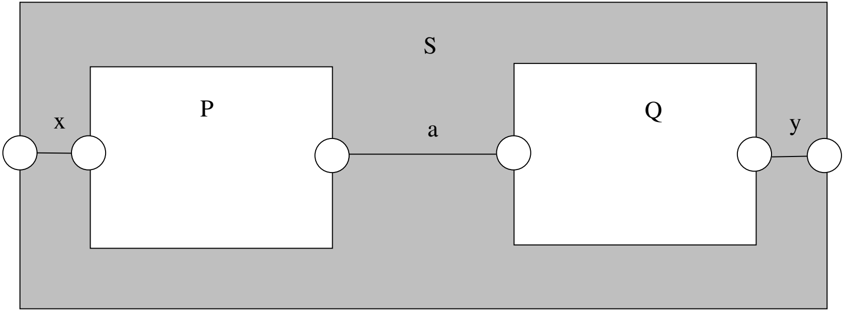


Strukturdiagramm von zwei parallelen Prozessen



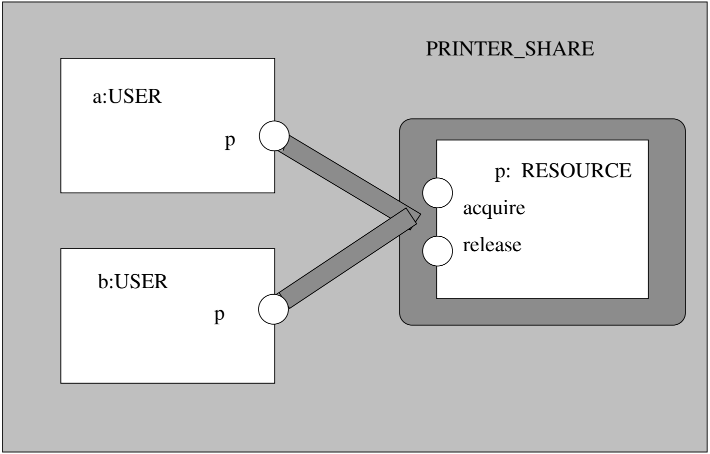
(P||Q) / {m/a, m/b, c/d}

Strukturdiagramm von zwei parallelen Prozessen mit Schnittstelle



$\| \; S = (P \| Q) @ \{x, y\}$

Strukturdiagramm von Prozessen mit Ressource-Sharing



RESOURCE = (acquire –> release –> RESOURCE).
USER = (p.acquire –> use –> p.release –> USER).
||PRINTER_SHARE = (a:USER||b:USER||{a,b}::p:RESOURCE).

3.2 Semantik von parallelen Prozessen

Die induktive Definition der Funktion lts: $\mathcal{E} \longrightarrow \mathcal{T}$ wird folgendermaßen erweitert auf:

6. Parallele Komposition von Prozessen
7. Umbenennung
8. Hiding (Verbergen von Aktionen) und
9. Alphabeterweiterung

6. Parallele Komposition:

Zunächst definieren wir den semantischen Operator $\_ \parallel_{lts} \_ : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$.

Seien $T = (S, A, \Delta, q_0) \in \mathcal{T}$ und $T' = (S', A', \Delta', q_0') \in \mathcal{T}$.
Dann definieren wir:

$$T \parallel_{lts} T' =_{def} (S \times S', A \cup A', \Delta_{T \parallel_{lts} T'}, (q_0, q_0')),$$

wobei $\Delta_{T \parallel_{lts} T'} =$

$$\begin{array}{rll} & \{((q, q'), a, (p, q')) \mid & (q, a, p) \in \Delta, a \notin \alpha T'\} \\ \cup & \{((q, q'), a, (q, p')) \mid & (q', a, p') \in \Delta', a \notin \alpha T\} \\ \cup & \{((q, q'), a, (p, p')) \mid & (q, a, p) \in \Delta, (q', a, p') \in \Delta', \\ & & a \in \alpha T \cap \alpha T'\} \end{array}$$

Bemerkung: $\parallel_{lts}$ ist assoziativ bis auf eine Bijektion zwischen Zuständen.

Seien $E_1, \ldots, E_n \in \mathcal{E}$ Prozessausdrücke.
Dann definieren wir:

$$\text{lts}\,((E_1 \parallel \ldots \parallel E_n)) =_{def} \text{Reach}\,(\text{lts}(E_1) \parallel_{lts} \ldots \parallel_{lts} \text{lts}(E_n))$$

7. Umbenennung:

Für jede Umbenennungsrelation $R \subseteq \text{Labels} \times \text{Labels}$ definieren wir den semantischen Operator $\_\{R\}_{lts} : \mathcal{T} \to \mathcal{T}$.

Sei $T = (S, A, \Delta, q_0) \in \mathcal{T}$.
Dann definieren wir:

$$T\{R\}_{lts} =_{def} (S, (A \backslash B_1) \cup B_2, (\Delta \backslash \Delta_1) \cup \Delta_2, q_0),$$

wobei

$$\begin{aligned} B_1 &= \{a \in A \mid \exists n \in \text{Labels} : (n, a) \in R\} \\ B_2 &= \{n \in \text{Labels} \mid \exists a \in \alpha T : (n, a) \in R\} \\ \Delta_1 &= \{(q, a, p) \in \Delta \mid a \in B_1\} \\ \Delta_2 &= \{(q, n, p) \mid (q, a, p) \in \Delta, (n, a) \in R\} \end{aligned}$$

Seien $E \in \mathcal{E}$ ein Prozessausdruck und $a_1, \ldots, a_k, n_1, \ldots, n_k \in \text{Labels}$.
Dann definieren wir:

$$\text{lts}(E\{n_1/a_1, \ldots, n_k/a_k\}) =_{def} \text{lts}(E)\{R\}_{lts},$$

wobei $R = \{(n_1, a_1), \ldots, (n_k, a_k)\}$.

8. Hiding:

Für jede Menge $H \subseteq$ Labels definieren wir den semantischen Operator $\_\backslash_{lts} H : \mathcal{T} \to \mathcal{T}$.

Sei $T = (S, A, \Delta, q_0) \in \mathcal{T}$.

Dann definieren wir:

$$T \backslash_{lts} H =_{def} (S, A \backslash_{lts} H, \Delta \backslash_{lts} H, q_0),$$

wobei

$$\begin{aligned} A \backslash_{lts} H &= (A \backslash H) \cup \{\tau\} \\ \Delta \backslash_{lts} H &= \{(q, a, p) \in \Delta \mid a \notin H\} \cup \{(q, \tau, p) \mid (q, a, p) \in \Delta, a \in H\} \end{aligned}$$

Seien $E \in \mathcal{E}$ ein Prozessausdruck und $H \subseteq$ Labels.

Dann definieren wir:

$$\text{lts}(E \backslash H) =_{def} \text{lts}(E) \backslash_{lts} H.$$

9. Alphabeterweiterung:

Für jede Menge $B \subseteq$ Labels definieren wir den semantischen Operator $\_ +_{lts} B : \mathcal{T} \to \mathcal{T}$.

Sei $T = (S, A, \Delta, q_0) \in \mathcal{T}$.
Dann definieren wir:

$$T +_{lts} B =_{def} (S, A \cup B, \Delta, q_0).$$

Seien $E \in \mathcal{E}$ ein Prozessausdruck und $B \subseteq$ Labels.
Dann definieren wir:

$$\mathrm{lts}(E + B) =_{def} \mathrm{lts}(E) +_{lts} B.$$

Beobachtbare Äquivalenz

Zwei Prozesse sind beobachtbar äquivalent, wenn sie für einen (externen) Beobachter, der keine τ-Aktionen sehen kann, nicht unterschieden werden können.

Definition:

Sei $\mathsf{T} = (\mathsf{S}, \mathsf{A}, \Delta, \mathsf{q}_0)$ ein LTS, seien q, p $\in$ S zwei Zustände und sei a $\in \alpha$T.
q *geht mit* a *modulo* τ *über in* p, geschrieben $\mathsf{q} \stackrel{\mathsf{a}}{\Longrightarrow}_\Delta \mathsf{p}$, wenn es eine Folge

$$\mathsf{q} \xrightarrow{\tau^*}_\Delta \mathsf{u} \xrightarrow{\mathsf{a}}_\Delta \mathsf{v} \xrightarrow{\tau^*}_\Delta \mathsf{p}$$

von Transitionen in Δ gibt, wobei „$\xrightarrow{\tau^*}_\Delta$“ für eine beliebige (endliche) Anzahl von τ-Übergängen in Δ steht (eventuell auch keinen, d.h. q $=$ u oder v $=$ p).

Wir schreiben $\mathsf{q} \stackrel{\epsilon}{\Longrightarrow}_\Delta \mathsf{p}$ für $\mathsf{q} \xrightarrow{\tau^*}_\Delta \mathsf{p}$.

Definition (Schwache Bisimulation):

Seien T, T' $\in \mathcal{T}$, T = (S, A, Δ, q_0), T' = (S', A', Δ', q_0') mit αT = αT'.
Eine **schwache Bisimulation** zwischen T und T' ist eine Relation R $\subseteq$ S $\times$ S',
so dass für alle (q, q') $\in$ R und für alle a $\in \alpha T \cup \{\epsilon\}$ gilt:

(1) Falls q $\stackrel{a}{\Longrightarrow}_{\Delta}$ p, dann existiert p' $\in$ S' mit q' $\stackrel{a}{\Longrightarrow}_{\Delta'}$ p' und (p, p') $\in$ R.

(2) Falls q' $\stackrel{a}{\Longrightarrow}_{\Delta'}$ p', dann existiert p $\in$ S mit q $\stackrel{a}{\Longrightarrow}_{\Delta}$ p und (p, p') $\in$ R.

Bemerkung:

Jede starke Bisimulation zwischen zwei LTSen T und T' ist auch eine schwache Bisimulation zwischen T und T'. Die Umkehrung gilt jedoch nicht!

Definition (Beobachtbare Äquivalenz von LTSen):

Seien T, T' $\in \mathcal{T}$, T = (S, A, Δ, q_0), T' = (S', A', Δ', q_0').
T und T' sind **beobachtbar äquivalent (schwach bisimular)**, geschrieben T $\approx$ T', wenn gilt:

(a) T und T' haben dasselbe Alphabet, d.h. αT = αT'.

(b) Es gibt eine schwache Bisimulation R $\subseteq$ S $\times$ S' zwischen T und T', so dass (q_0, q_0') $\in$ R.

Bemerkung:

Stark äquivalente LTSe sind auch beobachtbar äquivalent.
Die Umkehrung gilt jedoch nicht!

Lemma:

$\approx$ ist eine Äquivalenzrelation auf $\mathcal{T}$.
(Beweis analog zur starken Äquivalenz.)

Beispiele:

Definition (Beobachtbare Äquivalenz von Prozessen):
Zwei Prozesse E, F $\in \mathcal{E}$ sind **beobachtbar äquivalent (schwach bisimular)**, geschrieben E $\approx$ F, wenn gilt: lts(E) $\approx$ lts(F).

Beispiele:

1. Seien $P = (a \to STOP)$. $Q = (a \to Q)$.
   Dann gilt: $STOP \approx P\backslash\{a\} \approx Q\backslash\{a\}$.

2. $(a \to STOP \mid b \to k \to STOP)\backslash\{k\} \approx (a \to STOP \mid b \to STOP)$

3. Die folgenden Prozesse sind **nicht** beobachtbar äquivalent:
   $(a \to STOP \mid k \to b \to STOP)\backslash\{k\}$ und
   $(a \to STOP \mid b \to STOP)$

Algebraische Gesetze für beobachtbare Äquivalenz:

Seien a, b Aktionen, E, F, G Prozessausdrücke, R eine Umbenennung und H eine Menge von Labels.

- $(a \to E \mid b \to F) \approx (b \to F \mid a \to E)$
- $(a \to E \mid a \to E) \approx (a \to E)$
- $(E \| F) \approx (F \| E)$
- $((E \| F) \| G) \approx (E \| (F \| G))$
- $(E \| STOP) \approx E$
- $(\tau \to E) \approx E$
- $(a \to E) \setminus \{a\} \approx E \quad$ falls $a \notin \alpha E$.

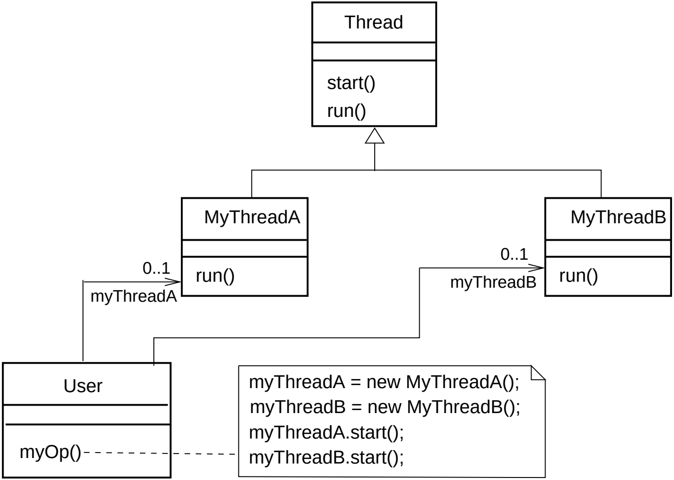
Kongruenzeigenschaften:

- $E \approx F \implies (a \to E) \approx (a \to F)$
- $E \approx F \implies (E \| G) \approx (F \| G)$
- $E \approx F \implies E\{R\} \approx F\{R\}$
- $E \approx F \implies E \setminus H \approx F \setminus H$

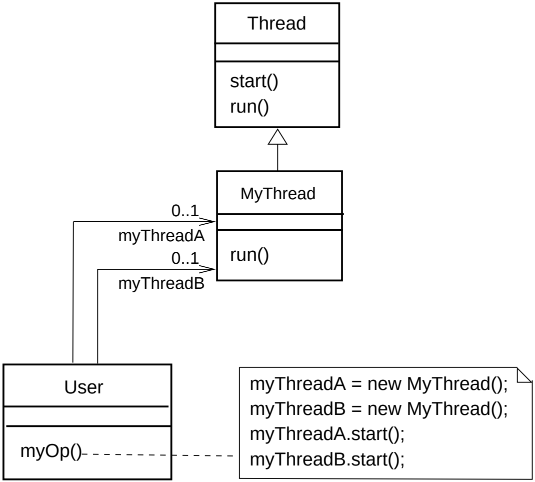
Beachte: $\approx$ ist keine Kongruenz bzgl. der Auswahl.

3.3 Java-Programme mit mehreren Threads

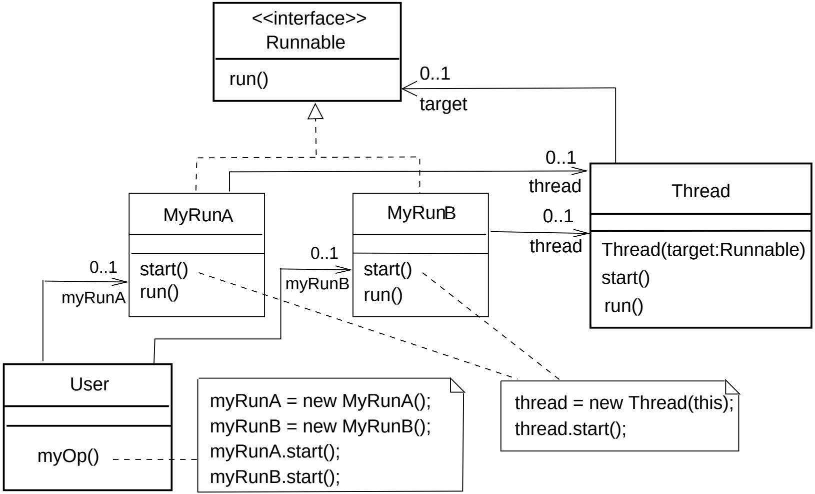
Realisierung mittels Vererbung



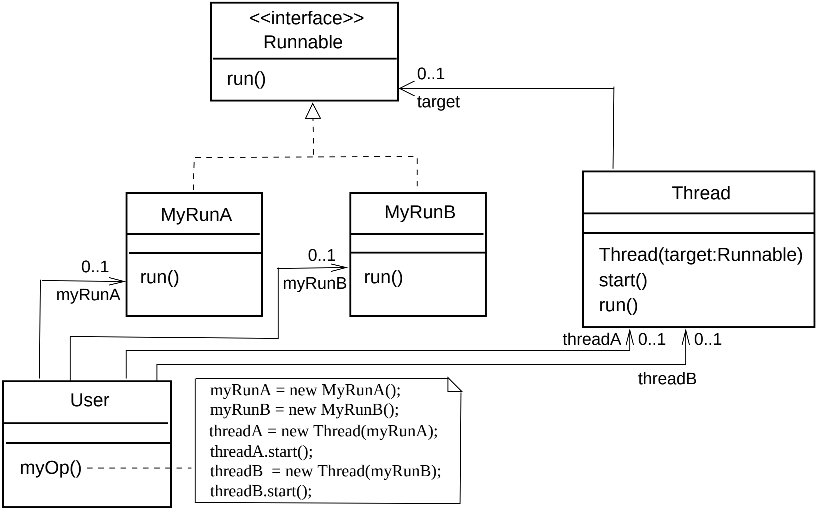
Realisierung mittels Vererbung (Variante)



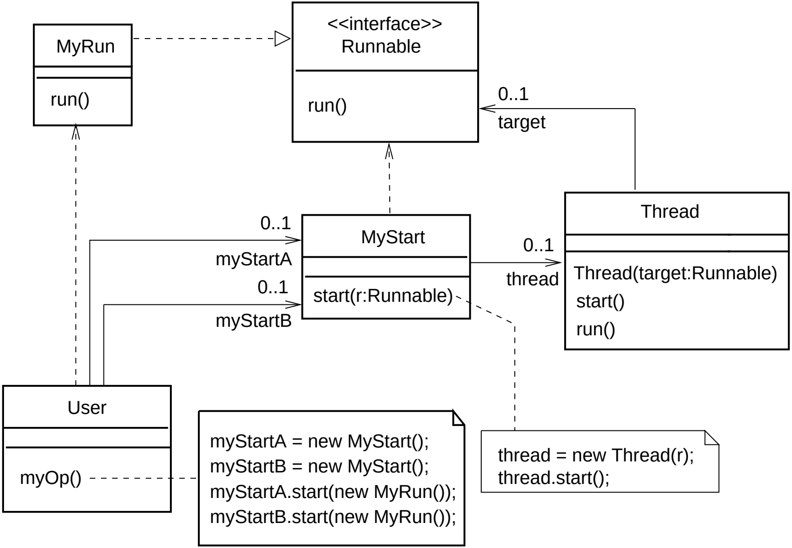
Realisierung durch Verwendung des Interfaces "Runnable"



Realisierung mit “Runnable“ (Variante 1)



Realisierung mit “Runnable“ (Variante 2)



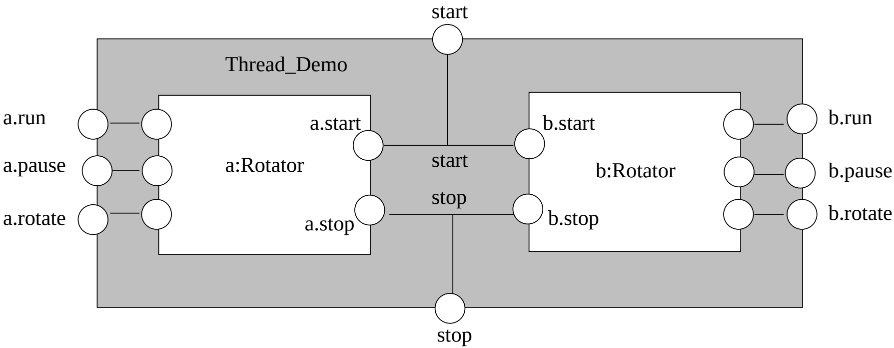
Beispiel (Rotierende Segmente):
vgl. [Magee, Kramer]

Zwei voneinander unabhägige Threads rotieren ein Kreissegment.

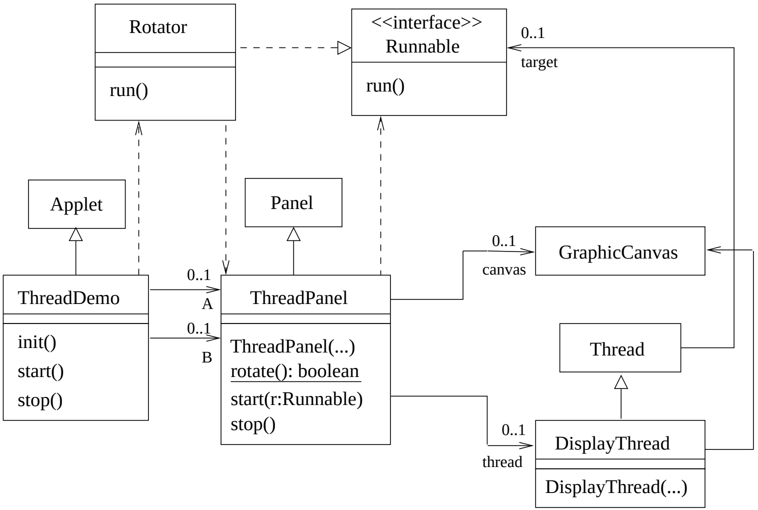
Modellierung:

$$
\begin{array}{lcl}
\mathrm{ROTATOR} & = & (\mathrm{start} \rightarrow \mathrm{PAUSED}), \\
\mathrm{PAUSED} & = & (\mathrm{run} \rightarrow \mathrm{RUN} \\
 & & \mid \mathrm{pause} \rightarrow \mathrm{PAUSED} \\
 & & \mid \mathrm{stop} \rightarrow \mathrm{STOP}), \\
\mathrm{RUN} & = & (\{\mathrm{rotate}, \mathrm{run}\} \rightarrow \mathrm{RUN} \\
 & & \mid \mathrm{pause} \rightarrow \mathrm{PAUSED} \\
 & & \mid \mathrm{stop} \rightarrow \mathrm{STOP}).
\end{array}
$$

$\|$THREAD_DEMO $=$
(a:ROTATOR$\|$b:ROTATOR)$/\{$start$/\{$a,b$\}$.start,stop$/\{$a,b$\}$.stop$\}$.


start
Thread_Demo
a.run
a.start
b.start
b.run
a.pause
a:Rotator
start
b:Rotator
b.pause
stop
a.rotate
a.stop
b.stop
b.rotate
stop

Implementierung des Modells:



Java-Code:

```
public class ThreadDemo extends Applet {
  ThreadPanel A,B;

  public void init() {
    A = new ThreadPanel("Thread A", Color.blue);
    B = new ThreadPanel("Thread B", Color.blue);
    add(A); add(B);
  }
  public void start() {  // synchronisation
    A.start(new Rotator());
    B.start(new Rotator());
  }
  public void stop() {
    A.stop();
    B.stop();
  }
}
```

```
public class ThreadPanel extends Panel {
   DisplayThread thread;
   GraphicCanvas canvas;
   // construct display with title and segment color c
   public ThreadPanel(String title, Color c) {...}
   // rotate display of currently running thread 6 degrees
   // return value not used in this example
   public static boolean rotate() throws InterruptedException {...}

   // create a new thread with target r and start it running
   public void start(Runnable r) {
      thread = new  DisplayThread(canvas, r,...);
      thread.start();
   }
   // stop the thread using interrupt()
   public void stop() {thread.interrupt(); }
}
public class Rotator implements Runnable {
   public void run() {
      try {
         while(true)  ThreadPanel.rotate();
      } catch(InterruptedException e) {}
   }
}
```