

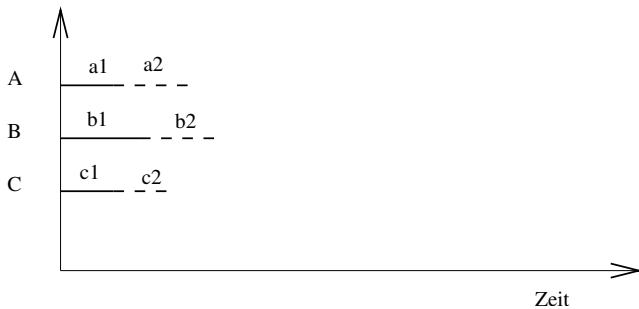
Kapitel 3

Parallele Prozesse

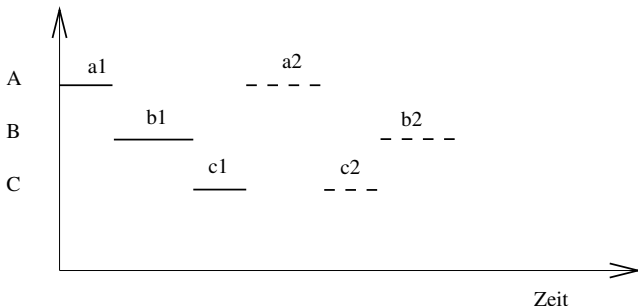
Prof. Dr. Rolf Hennicker

18.05.2017

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 eines Prozesses ist dieselbe wie bei echter Parallelität.

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

6. Parallele Komposition von Prozessen

6. FSP-Konstrukt

Bisher wurden 5 grundlegende Konstrukte für FSP Prozessausdrücke definiert.

Definition:

Sind E_1, \dots, E_n Prozessausdrücke, dann ist

$$(E_1 \parallel E_2 \parallel \dots \parallel E_n)$$

ein Prozessausdruck (*parallele Komposition* von E_1, \dots, E_n)

mit $FV((E_1 \parallel E_2 \parallel \dots \parallel E_n)) = FV(E_1) \cup \dots \cup FV(E_n)$.

Wirkung:

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

Deklaration paralleler Prozesse:

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

$$\textcircled{\parallel} P = (E_1 \parallel \dots \parallel E_n).$$

~~$$\parallel P = (E \parallel P).$$~~

~~$$\text{rec}(P = E \parallel P)$$~~

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

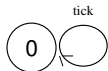
Beispiel:

CLOCK = (tick \rightarrow CLOCK).

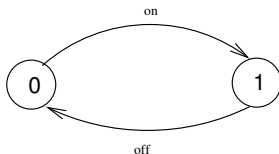
RADIO = (on \rightarrow off \rightarrow RADIO).

\parallel CLOCK_RADIO = (CLOCK \parallel RADIO).

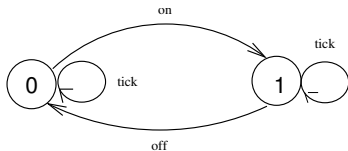
Zugehörige LTSe:



CLOCK



RADIO



(CLOCK \parallel RADIO)

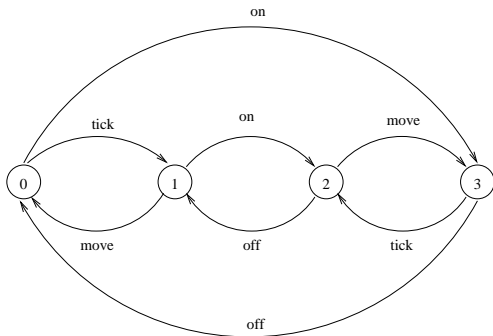
Beispiel:

CLOCK2 = (tick \rightarrow move \rightarrow CLOCK2).

RADIO = (on \rightarrow off \rightarrow RADIO).

\parallel CLOCK2_RADIO = (CLOCK2 \parallel RADIO).

Its(CLOCK2):



Prozessinteraktionen

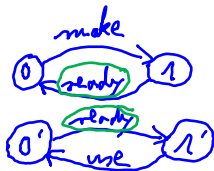
- ▶ Prozessinteraktionen werden durch *gemeinsame* Aktionen ("shared actions") modelliert.
- ▶ Parallele Prozesse, die gemeinsame Aktionen haben, müssen diese gemeinsam ausführen, d.h. sie müssen sich synchronisieren.
- ▶ Die Synchronisation schränkt die möglichen Abläufe der parallelen Komposition ein.

Beispiel:

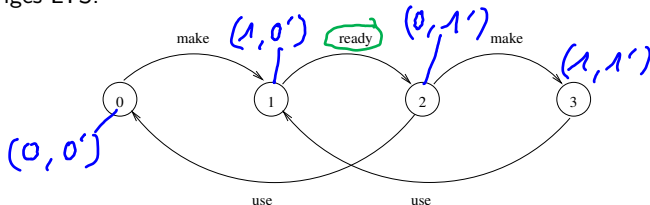
MAKER = (make \rightarrow ready \rightarrow MAKER).

USER = (ready \rightarrow use \rightarrow USER).

||MAKER_USER = (MAKER || USER).



Zugehöriges LTS:



Variante:

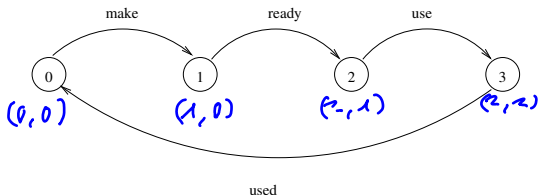
MAKER produziert erst dann weiter, wenn der USER die Benutzung bestätigt hat.

MAKER2 = (make \rightarrow ready \rightarrow used \rightarrow MAKER2).

USER2 = (ready \rightarrow use \rightarrow used \rightarrow USER2).

\parallel MAKER_USER2 = (MAKER2 \parallel USER2).

Zugehöriges LTS:



Es gibt etliche nicht-reachable Zustände
im kartesischen Produkt $S \times S'$,
z. B. $(1, 1)$

7. Umbenennung von Aktionen

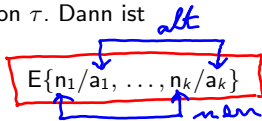
Die Umbenennung von Aktionen dient (vor allem)

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

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

Definition:

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



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

Wirkung:

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

Erstellung von Prozess-Kopien durch Umbenennung

Beispiel:

CLIENT = (call → wait → continue → CLIENT).

||TWOCLIENTS = (a:CLIENT || b:CLIENT). ← beliebiges Interleaving

Dabei ist a:CLIENT eine abkürzende Schreibweise für
 CLIENT{a.call/call, a.wait/wait, a.continue/continue}.
 der Aktionennamen
 von a: CLIENT
 und b: CLIENT

Man nennt dies Process Labelling.

Abkürzende Schreibweisen für parallele Kompositionen von Prozesskopien

```
const N = ...
range ID = 1..N
```

Die Ausdrücke

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

bezeichnen alle den Prozess

$$(a[1]:E \parallel \dots \parallel a[N]:E)$$

Der Bezeichner a kann auch weggelassen werden.

Zum Beispiel bezeichnet $[ID]:E$ den Prozess

z.B. [ID]: CLIENT

$$([1]:E \parallel \dots \parallel [N]:E)$$

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

```
forall[i:1 ...N] E für (E[1/i] || ... || E[N/i])
```

später

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

Synchronisation von Prozessen durch Umbenennung

$$(E_1 \parallel E_2 \parallel \dots \parallel E_n) / \{n_1/a_1, \dots, n_k/a_k\} =_{\text{def}}$$

$$(E_1 \{n_1/a_1, \dots, n_k/a_k\} \parallel \dots \parallel E_n \{n_1/a_1, \dots, n_k/a_k\})$$

Beispiel:

CLIENT = (call → wait → continue → CLIENT).

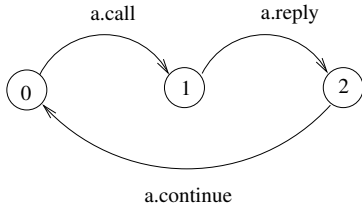
SERVER = (request → service → reply → SERVER).

|| CLIENT_SERVER = (CLIENT || SERVER) / {call/request, reply/wait}.

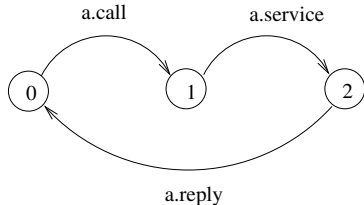
(CLIENT { call/request, reply/wait } ||
 SERVER { call/request, reply/wait })
 ohne Winkung

Beispiel für Synchronisation von Prozess-Kopien:

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

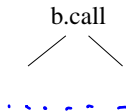
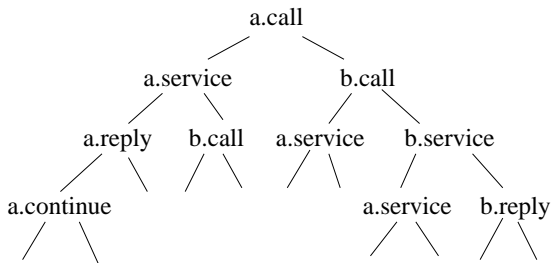


(a:CLIENT){... a.reply/a.wait ...}



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

— Synchronisation —
 vor der parall. Komposition



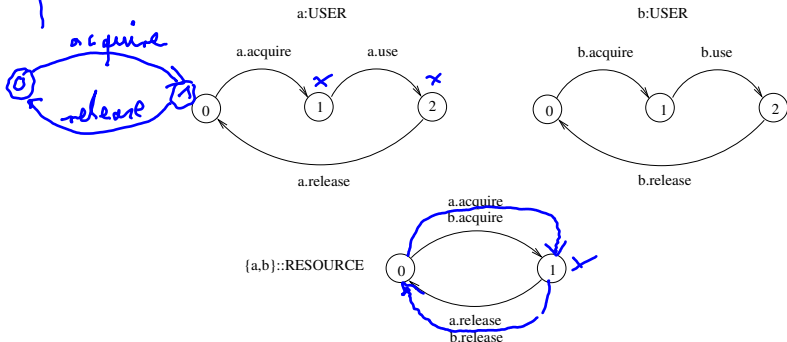
Beispiel (Resource-Sharing):

kritischer Bereich

USER = (acquire → use → release → USER).

RESOURCE = (acquire → release → RESOURCE)

|| RESOURCE_SHARE = (a:USER || b:USER || {a,b}::RESOURCE).

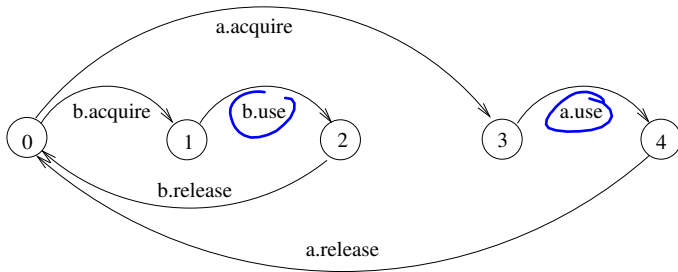


$\{a,b\}::RESOURCE$ steht für die Umbenennung

$RESOURCE\{\{a.acquire, b.acquire\}/acquire, \{a.release, b.release\}/release\}$

Umbenennung ist hier keine Funktion

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.

Systemen

Definition:

Sei E ein (event. paralleler) Prozessausdruck und sei $H \subseteq \text{Labels}$ eine *endliche* Menge von Aktionsnamen ($\neq \tau$).

Dann ist $E \setminus H$ ein Prozessausdruck ("Hiding") mit $FV(E \setminus H) = FV(E)$.

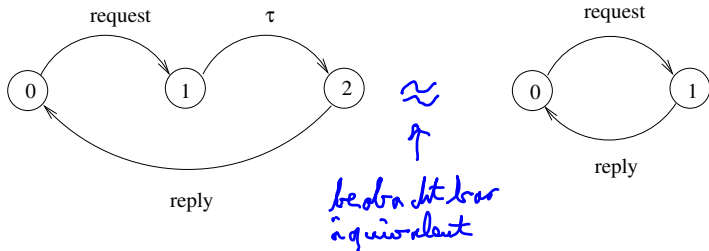
konkret $E \setminus \{a_1, \dots, a_m\}$

Wirkung:

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

Beispiel:

$\|\text{SERVER2} = \text{SERVER} \setminus \{\text{service}\}.$ $\text{SERVER} @ \{\text{request}, \text{reply}\}$



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 nicht bzgl. τ synchronisiert werden; τ ist intern und daher keine gemeinsame Aktion parallel laufender Prozesse.

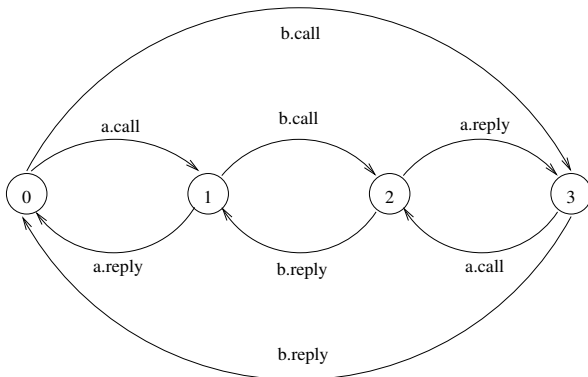
Beispiel:

\parallel TCLIENTS_SERVER =

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

a. continue, b. continue

Bezüglich beobachtbarer Äquivalenz minimalisiertes LTS:



$$E @ I \quad E @ \{a_1, \dots, a_m\}$$

wobei E ein Prozessausdruck und $I \subseteq \text{Labels}$ 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:
 $(P \parallel Q) / \{\text{neu/alt}\} @ \{a_1, \dots, a_k\}$

Beispiel:

$(\text{MAKER} \parallel \text{USER}) @ \{\text{make, use}\}$ entspricht $(\text{MAKER} \parallel \text{USER}) \setminus \{\text{ready}\}$

9. Alphabeterweiterung

Definition:

(1) Sei $T = (S, A, \Delta, q)$ ein LTS.

Dann heißt die Menge $\alpha T =_{\text{def}} A \setminus \{\tau\}$ das *Alphabet* von T .

(2) Sei E ein Prozessausdruck mit $\text{Its}(E) = T$.

Dann heißt die Menge $\alpha E =_{\text{def}} \alpha T$ das *Alphabet* von E .

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

Definition:

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

Beispiel:

$\text{FMAKER} = (\text{make} \rightarrow \text{ready} \rightarrow \text{FMAKER}) + \{\text{use}\}$.

$\text{USER} = (\text{ready} \rightarrow \text{use} \rightarrow \text{USER})$.

$\parallel \text{FMAKER_USER} = (\text{FMAKER} \parallel \text{USER})$.

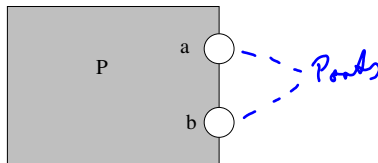


Strukturdiagramme

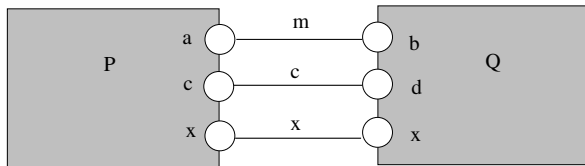
Architektur

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

Strukturdiagramm eines Prozesses mit Alphabet $\{a,b\}$

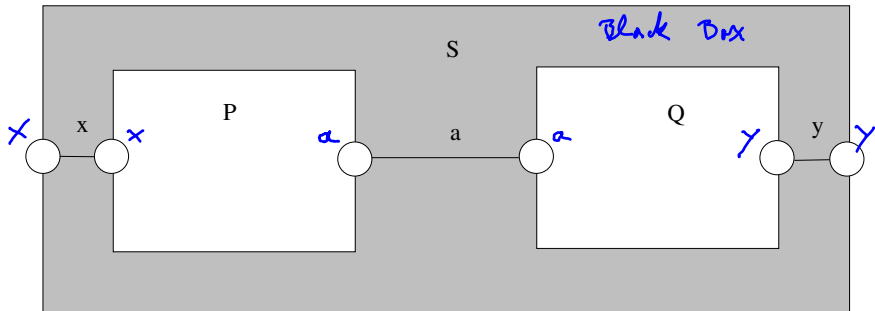


Strukturdiagramm von zwei parallelen Prozessen



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

Strukturdiagramm von zwei parallelen Prozessen mit Schnittstelle



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

$$(P \parallel Q) \setminus \{a\}$$

$$\alpha S = \{x, y\}$$

$$\alpha (P \parallel Q) = \{x, a, y\}$$

3.2 Semantik von parallelen Prozessen

Die induktive Definition der Funktion $\text{Its}: \mathcal{E} \longrightarrow \mathcal{T}$ 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 $_ \parallel_{lts} _ : \mathcal{T} \times \mathcal{T} \rightarrow \mathcal{T}$.

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

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

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

Seien $E_1, \dots, E_n \in \mathcal{E}$ Prozessausdrücke.

Dann definieren wir:

$$lts(\boxed{E_1 \parallel \dots \parallel E_n}) =_{def} \underline{\text{Reach}}(lts(E_1) \parallel_{lts} \dots \parallel_{lts} lts(E_n))$$

7. Umbenennung:

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

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

Dann definieren wir:

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

wobei

$$B_1 = \{a \in A \mid \exists n \in \text{Labels} : (n, a) \in R\} \text{ alte Aktionen}$$

$$B_2 = \{n \in \text{Labels} \mid \exists a \in \alpha T : (n, a) \in R\} \text{ neue Aktionen}$$

$$\Delta_1 = \{(q, a, p) \in \Delta \mid a \in B_1\}$$

$$\Delta_2 = \{(q, n, p) \mid (q, a, p) \in \Delta, (n, a) \in R\}$$

Seien $E \in \mathcal{E}$ ein Prozessausdruck und $a_1, \dots, a_k, n_1, \dots, n_k \in \text{Labels}$.

Dann definieren wir:

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

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

neu ↑ *alt* ↑

8. Hiding:

Für jede Menge $H \subseteq \text{Labels}$ definieren wir den semantischen Operator

$$_ \backslash_{\text{Its}} H : \mathcal{T} \rightarrow \mathcal{T}.$$

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

Dann definieren wir:

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

wobei

$$A \backslash_{\text{Its}} H = (A \setminus H) \cup \{\tau\}$$

$$\Delta \backslash_{\text{Its}} H = \{(q, a, p) \in \Delta \mid a \notin H\} \cup \{(q, \tau, p) \mid (q, a, p) \in \Delta, a \in H\}$$

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

Dann definieren wir:

$$\text{Its}(E \setminus H) =_{\text{def}} \text{Its}(E) \backslash_{\text{Its}} H.$$

9. Alphabeterweiterung:

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

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

Dann definieren wir:

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

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

Dann definieren wir:

$$\text{Its}(\underline{E + B}) =_{\text{def}} \text{Its}(E) +_{\text{Its}} B.$$

Bemerkung: Für alle FSP Prozessausdrücke E ist $\text{Its}(E)$ reachable.

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 $T = (S, A, \Delta, q_0)$ 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 $q \xRightarrow{a}_{\Delta} p$, wenn es eine Folge

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

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 $q \xRightarrow{\epsilon}_{\Delta} p$ für $q \xrightarrow{\tau^*}_{\Delta} p$.

$$q \xRightarrow{\epsilon} q$$

Definition (Schwache Bisimulation):

Seien $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$ mit $\alpha T = \alpha T'$.
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 $q \xrightarrow{a}_{\Delta} p$, dann existiert $p' \in S'$ mit $q' \xrightarrow{a}_{\Delta'} p'$ und $(p, p') \in R$.
- (2) Falls $q' \xrightarrow{a}_{\Delta'} p'$, dann existiert $p \in S$ mit $q \xrightarrow{a}_{\Delta} p$ 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 $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$.

T und T' sind **beobachtbar äquivalent (schwach bisimilar)**, geschrieben $T \approx T'$, wenn gilt:

- (a) T und T' haben dasselbe Alphabet, d.h. $\alpha T = \alpha T'$.
- (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 LTSen sind auch beobachtbar äquivalent.
Die Umkehrung gilt jedoch nicht!

Lemma:

\approx ist eine Äquivalenzrelation auf \mathcal{T} .

(Beweis analog zur starken Äquivalenz.)

Definition (Beobachtbare Äquivalenz von Prozessen):

Zwei Prozesse $E, F \in \mathcal{E}$ sind **beobachtbar äquivalent (schwach bisimilar)**, geschrieben $E \approx F$, wenn gilt: $\text{Its}(E) \approx \text{Its}(F)$.

Beispiele:

- Seien $P = (a \rightarrow \text{STOP})$. $Q = (a \rightarrow Q)$.
Dann gilt: $\text{STOP} \approx P \setminus \{a\} \approx Q \setminus \{a\}$.
- $(a \rightarrow \text{STOP} \mid b \rightarrow k \rightarrow \text{STOP}) \setminus \{k\} \approx (a \rightarrow \text{STOP} \mid b \rightarrow \text{STOP})$
- Die folgenden Prozesse sind **nicht** beobachtbar äquivalent:
 $(a \rightarrow \text{STOP} \mid k \rightarrow b \rightarrow \text{STOP}) \setminus \{k\}$ und
 $(a \rightarrow \text{STOP} \mid b \rightarrow \text{STOP})$

Algebraische Gesetze für beobachtbare Äquivalenz:

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

- ▶ $(a \rightarrow E \mid b \rightarrow F) \approx (b \rightarrow F \mid a \rightarrow E)$
- ▶ $(a \rightarrow E \mid a \rightarrow E) \approx (a \rightarrow E)$
- ▶ $(E \parallel F) \approx (F \parallel E)$
- ▶ $((E \parallel F) \parallel G) \approx (E \parallel (F \parallel G))$
- ▶ $(E \parallel \text{STOP}) \approx E$
- ▶ $(\tau \rightarrow E) \approx E$
- ▶ $(a \rightarrow E) \setminus \{a\} \approx E$ falls $a \notin \alpha E$.

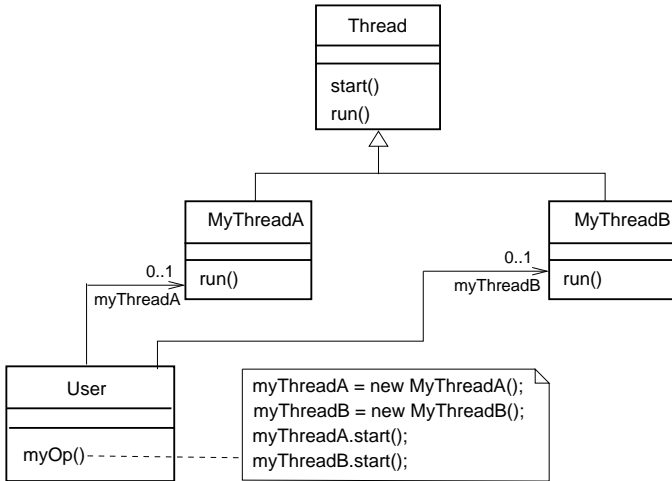
Kongruenzeigenschaften:

- ▶ $E \approx F \implies (a \rightarrow E) \approx (a \rightarrow F)$
- ▶ $E \approx F \implies (E \parallel G) \approx (F \parallel G)$
- ▶ $E \approx F \implies E\{R\} \approx F\{R\}$
- ▶ $E \approx F \implies E \setminus H \approx F \setminus H$

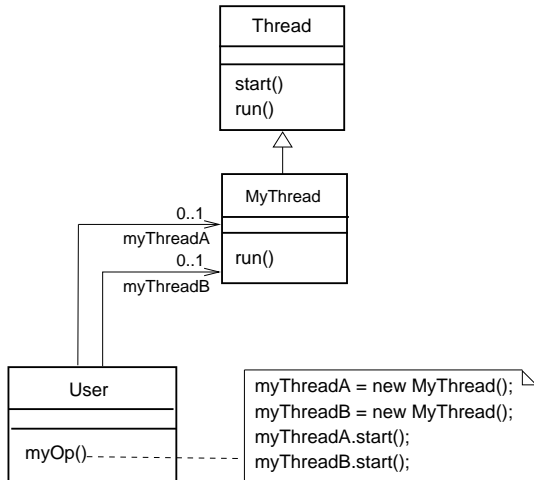
Beachte: \approx ist keine Kongruenz bzgl. der Auswahl.

3.3 Java-Programme mit mehreren Threads

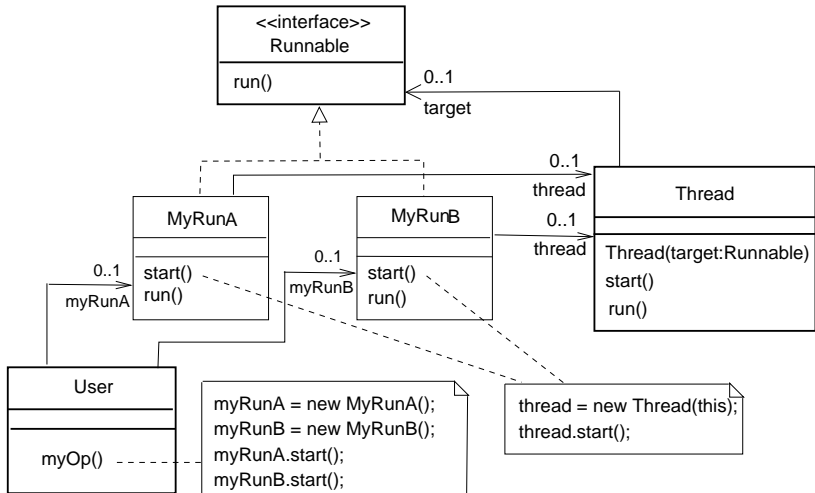
Realisierung mittels Vererbung



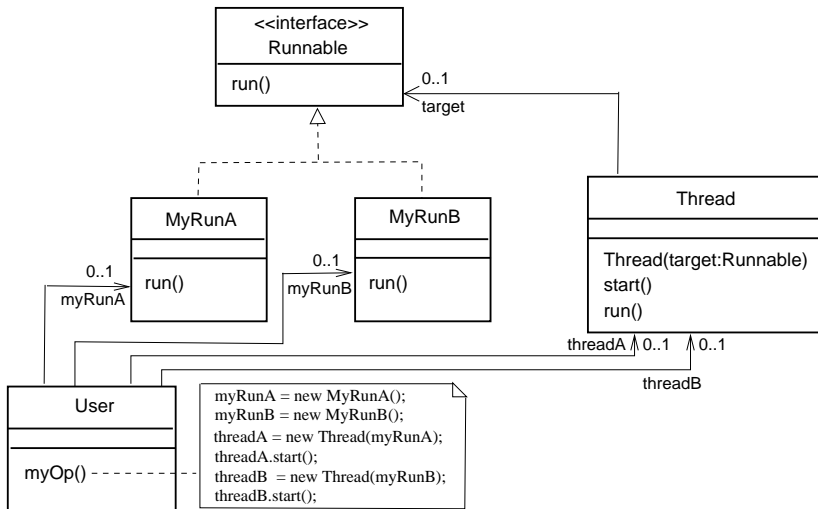
Realisierung mittels Vererbung (Variante für "Prozesskopien")



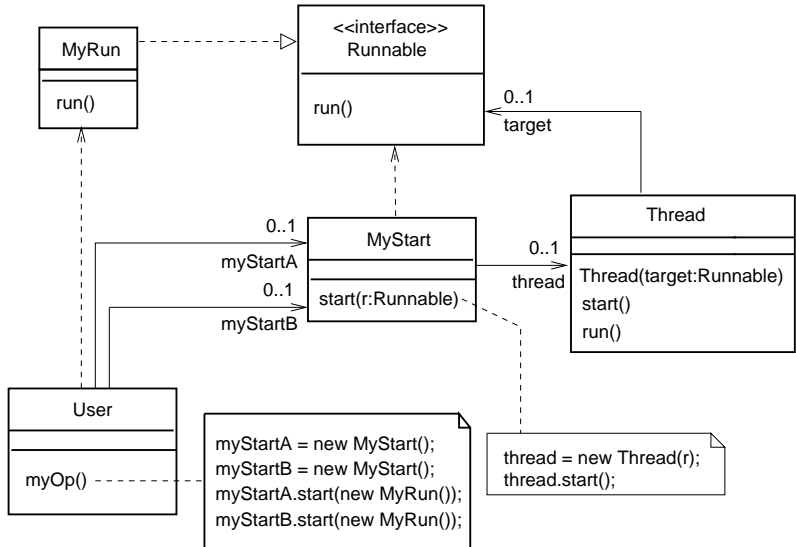
Realisierung durch Verwendung des Interfaces "Runnable"



Realisierung mit "Runnable" (Variante 1)



Realisierung mit "Runnable" (Variante 2 für "Prozesskopien")



Beispiel (Rotierende Segmente):

vgl. [Magee, Kramer]

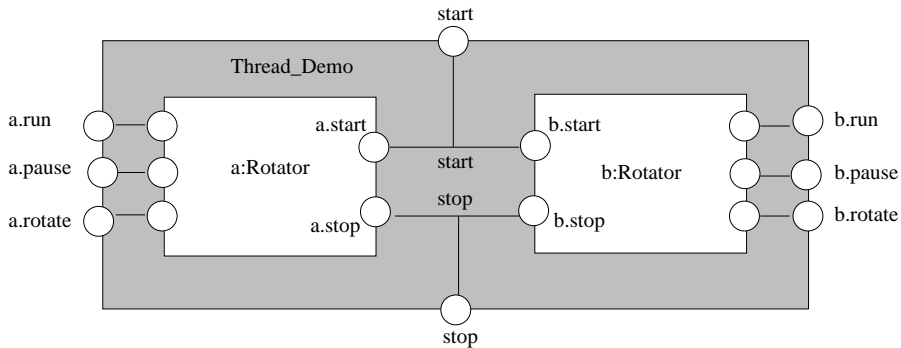
Zwei voneinander unabhängige Threads rotieren ein Kreissegment.

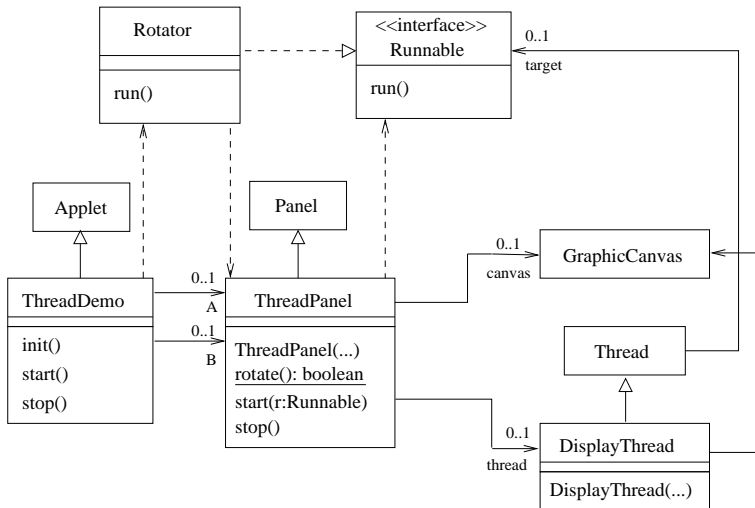
Modellierung:

ROTATOR = (start → PAUSED),
 PAUSED = (run → RUN
 | pause → PAUSED
 | stop → STOP),
 RUN = ({rotate, run} → RUN
 | pause → PAUSED
 | stop → STOP).

||THREAD_DEMO =

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





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) {}
    }
}
```