Kapitel 2

Prozesse und Java-Threads

Prof. Dr. Rolf Hennicker

14.04.2016

2.1 Prozessbegriff

Prozess:
Programm in Ausführung

Prozesszustand (zu einem Zeitpunkt):
Wird charakterisiert durch die Werte von

- explizite Variablen (vom Programmierer deklariert)
- impliziten Variablen (Befehlszähler, organisatorische Daten)

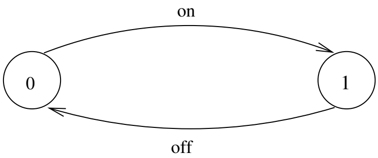
Zustandsübergang (eines Prozesses):
Wird von einer Aktion bewirkt. Aktionen sind elementar, d.h. nicht unterbrechbar.

Bemerkung:
Im Folgenden abstrahieren wir von den konkreten Zustandsdarstellungen, d.h. von den konkreten Werten der expliziten und impliziten Variablen.

Wir interessieren uns dafür, welche Aktionen in einem Prozesszustand als nächstes möglich sind.

2.2 Modellierung durch endliche Zustandsmaschinen

Beispiel: Lichtschalter als Zustandsmaschine



Grafische Darstellung mit dem Tool LTSA (Labelled Transition System Analyser)**:**
Zustände werden von 0 beginnend durchnummeriert.
0 ist der Anfangszustand.

Ablauf: on off on off on off $\cdots$

Ein **Ablauf** ist eine Aktionsfolge, die ein Prozess ausführen kann, die entweder unendlich ist oder in einem Zustand endet, in dem der Prozess nicht fortgesetzt werden kann.

Beachte:

- Wir betrachten nur Prozesse mit endlich vielen Zuständen und endlicher Menge von Aktionen.
- Das Verhalten eines Prozesses kann aber unendlich sein (nicht terminierend).

Die hier betrachteten Zustandsmaschinen sind formal *endliche markierte Transitionssysteme* („Labelled Transition Systems"), abgekürzt LTS.

Definition:

Sei `States` eine universelle, abzählbar unendliche Menge von Zuständen und `ACT` eine universelle, abzählbar unendliche Menge von (sichtbaren) Aktionen. Ein endliches LTS ist ein Quadrupel

$$(S, A, \Delta, q_0),$$

wobei

- $S \subseteq$ `States` eine endliche Menge von Zuständen ist,
- $A \subseteq$ `ACT` eine endliche Menge von Aktionen ist,
- $\Delta \subseteq S \times A \times S$ eine Übergangsrelation ist,
- $q_0 \in S$ ein Anfangszustand ist.

Notation: Für $(q, a, p) \in \Delta$ schreiben wir auch $q \xrightarrow{a}_{\Delta} p$

oder grafisch

$$q \xrightarrow{a} p$$

Beispiel: Lichtschalter (formal)

Das LTS (S, A, Δ, q_0) besteht aus

$$
\begin{aligned}
S &= \{0, 1\} \\
A &= \{\mathrm{on}, \mathrm{off}\} \\
\Delta &= \{(0, \mathrm{on}, 1), (1, \mathrm{off}, 0)\} \\
q_0 &= 0
\end{aligned}
$$

2.3 Prozessausdrücke

Prozesse werden kompakt beschrieben durch Ausdrücke der Sprache FSP (Finite State Processes) [Magee, Kramer].

FSP ist eine Variante einer "Prozessalgebra".

FSP orientiert sich

- syntaktisch an CSP [Hoare] (Communicating Sequential Processes)
- semantisch an CCS [Milner] (Calculus of Communicating Systems)

Die *Semantik* eines Prozessausdrucks E wird durch Übersetzung in ein LTS gegeben.

- Im Folgenden werden FSP-Prozessausdrücke induktiv (über deren strukturellen Aufbau) definiert.
- Dabei wird jedem Prozessausdruck E eine Menge von freien Variablen FV(E) zugeordnet. Die Variablen stellen Prozessidentifikatoren dar.

1. Konstante Prozessausdrücke und 2. Prozessidentifikatoren

Sei PID eine universelle, abzählbar unendliche Menge von Prozessidentifikatoren (Bezeichnern).

Definition:

1. STOP ist ein (konstanter) Prozessausdruck mit $\mathrm{FV}(\mathrm{STOP}) = \emptyset$.
2. Jeder Prozessidentifikator $\mathrm{P} \in \mathrm{PID}$ ist ein Prozessausdruck mit $\mathrm{FV}(\mathrm{P}) = \{\mathrm{P}\}$.

Wirkung:

1. STOP bezeichnet den Prozess, der keine Aktion ausführen kann.
2. Die Wirkung von $\mathrm{P} \in \mathrm{PID}$ kann nur im Zusammenhang mit einer Prozessdeklaration "$\mathrm{P} = \mathrm{E}$." beschrieben werden (vgl. unten).

3 Aktionspräfix

Definition:

Ist a $\in$ ACT eine Aktion und E ein Prozessausdruck,
dann ist das *Aktionspräfix* $\boxed{(a \to E)}$ ebenfalls ein Prozessausdruck
mit $FV((a \to E)) = FV(E)$.

Statt von Prozessausdrücken sprechen wir häufig kurz von „Prozessen“.

Wirkung:

Der Prozess $(a \to E)$ engagiert sich zunächst in die Aktion a
und verhält sich dann wie E.

Beispiele:

$(eat \to \mathtt{STOP})$

$(eat \to (drink \to \mathtt{STOP}))$

Abkürzende Schreibweise (eat → drink → STOP)

4. Auswahl

Definition:

Sind $a_1, \dots, a_n$ Aktionen und $E_1, \dots, E_n$ Prozessausdrücke mit $n \geq 2$, dann ist $\boxed{(a_1 \to E_1 \mid \dots \mid a_n \to E_n)}$ ein Prozessausdruck mit $FV((a_1 \to E_1 \mid \dots \mid a_n \to E_n)) = FV(E_1) \cup \dots \cup FV(E_n)$.

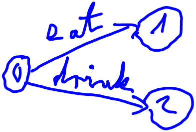
Wirkung:

Der Prozess engagiert sich entweder

- in a_1 und verhält sich danach wie E_1 oder
- in a_2 und verhält sich danach wie E_2 oder

  ⋮
- in a_n und verhält sich danach wie E_n.

Beispiel:



$(eat \to STOP \mid drink \to STOP)$

Kurznotation: $(\{eat, drink\} \to STOP)$

Allgemein:

$(\underbrace{\{a_1, \dots, a_n\}}_{\text{Auswahl}} \to E)$ statt für $(a_1 \to E \mid \dots \mid a_n \to E)$

5. Prozessausdrücke mit Rekursion

Werden später bei der Definition der Semantik von Prozessidentifikatoren P im Kontext einer rekursiven Prozessdeklaration "$P = E$." verwendet.

Definition:
Sei P ein Prozessidentifikator und E ein Prozessausdruck, so dass $P \in FV(E)$. Dann ist $\boxed{rec(P = E)}$ ein Prozessausdruck mit $FV(rec(P = E)) = FV(E) \setminus \{P\}$.

Beispiel:

$rec(H = \underbrace{(eat \to H)}_{E})$

$FV(rec(H = (eat \to H))) = \emptyset$

LTS:

$FV(rec(H = (eat \to H \mid drink \to K))) = \{K\}$

(Rekursive) Prozessdeklarationen

Definition:

Ist P ein Prozessidentifikator und E ein Prozessausdruck, dann ist

$$P = E.$$

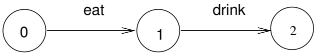
eine *Prozessdeklaration*. Die Deklaration ist *rekursiv*, wenn P in dem Ausdruck E frei vorkommt, d.h. $P \in FV(E)$.

Beispiele:

Prozessausdruck, vgl. 2.

1. $PERS = \overbrace{(eat \to drink \to STOP)}^{E}.$

   Das LTS von PERS ist gegeben durch das LTS des Prozessausdrucks $(eat \to drink \to STOP)$.

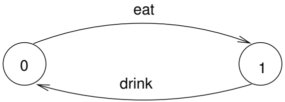


2. $PERSON = \overbrace{(eat \to drink \to PERSON)}^{E}.$

   Das LTS von PERSON ist gegeben durch das LTS des Prozessausdrucks

   PERSON = $rec(PERSON = (eat \to drink \to PERSON))$.

   vgl. 5.



Äquivalente Prozessbeschreibung mit **lokalen** Prozessdeklarationen:

global

PERSON = EATING

EATING = (eat → DRINKING)

DRINKING = (drink → PERSON).

lokal

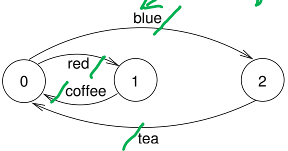
Beispiel (Getränkeautomat):

DRINKS = (red → coffee → DRINKS | blue → tea → DRINKS).

UML:

Ereignis in UML

Zugehöriges LTS:



Aktion in UML

State machines

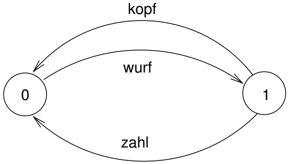
Bemerkungen:

- blue, red $\widehat{=}$ Input-Aktionen (der Automat empfängt)
- coffee, tea $\widehat{=}$ Output-Aktionen (der Automat gibt aus)
- Häufig beginnen die Alternativen einer Auswahl mit Input–Aktionen.
- Im Beispiel DRINKS gibt es unendlich viele mögliche Abläufe (die alle unendlich lang sind):
  - red coffee red coffee $\cdots$
  - red coffee blue tea blue $\cdots$

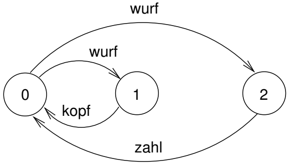
    $\ldots$
  - blue tea red coffee $\cdots$
  - $\ldots$

Beispiel (Münzwurf):

$$\text{MÜNZE1} = \quad (\text{wurf} \rightarrow \quad (\text{kopf} \rightarrow \text{MÜNZE1} \mid \text{zahl} \rightarrow \text{MÜNZE1})).$$



$$\text{MÜNZE2} = \quad (\text{wurf} \rightarrow \text{kopf} \rightarrow \text{MÜNZE2} \mid \text{wurf} \rightarrow \text{zahl} \rightarrow \text{MÜNZE2}).$$



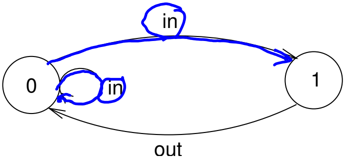
nicht

Beachte:

Beide Prozesse haben dieselben Abläufe, jedoch verschiedene (nicht äquivalente) LTSe.

Beispiel (Fehlerhafter Übertragungskanal):

$$\begin{aligned} \text{F\_CHAN} = \quad & (\text{in} \rightarrow \text{out} \rightarrow \text{F\_CHAN} \\ & \mid \text{in} \rightarrow \text{F\_CHAN}). \end{aligned}$$



Der Prozess ist nichtdeterministisch!

!

Im Folgenden betrachten wir **wichtige** abkürzende Schreibweisen für Prozessausdrücke, die sich alle auf die bisherigen 5 Konstrukte für Prozessausdrücke zurückführen lassen.

Indizierte Aktionen und indizierte Prozesse

Indizierte Aktionen:

Können zur Modellierung von Daten eines endlichen Datenbereichs (als Indizes von Aktionen) verwendet werden.

Beispiel (Korrekter Übertragungskanal für Daten):

Auswahl über {in[0], in[1], in[2]}

$$\mathrm{CHAN} = (\mathrm{in}[i{:}0..2] \rightarrow \mathrm{out}[i] \rightarrow \mathrm{CHAN}).$$

ist Kurzschreibweise für:

$$\begin{aligned}\mathrm{CHAN} = \quad & (\mathrm{in}[0] \rightarrow \mathrm{out}[0] \rightarrow \mathrm{CHAN} \\ & \mid \mathrm{in}[1] \rightarrow \mathrm{out}[1] \rightarrow \mathrm{CHAN} \\ & \mid \mathrm{in}[2] \rightarrow \mathrm{out}[2] \rightarrow \mathrm{CHAN}).\end{aligned}$$

Aktionen:

in[0], out[0], ...

Beachte:

Der Indexbereich muss beschränkt sein.

Indizierte Prozesse: sind immer lokal

Dienen zur Vereinfachung von Prozessdeklarationen (mit lokalen Prozessen).

Beispiel (Kanal):

$$\text{CHAN} = (\text{in}[i{:}0..2] \rightarrow \text{TRANSMIT}[i]\ ),$$
$$\text{TRANSMIT}[i{:}0..2] = (\text{out}[i] \rightarrow \text{CHAN}).$$

3 lokale Prozesse (für i = 0,..,2)

steht für:

$$\begin{aligned} \text{CHAN} = \quad & (\text{in}[0] \rightarrow \text{TRANSMIT}[0] \\ & \mid \text{in}[1] \rightarrow \text{TRANSMIT}[1] \\ & \mid \text{in}[2] \rightarrow \text{TRANSMIT}[2]), \end{aligned}$$

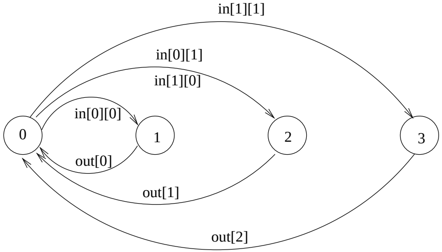
$$\begin{aligned} \text{TRANSMIT}[0] &= (\text{out}[0] \rightarrow \text{CHAN}), \\ \text{TRANSMIT}[1] &= (\text{out}[1] \rightarrow \text{CHAN}), \\ \text{TRANSMIT}[2] &= (\text{out}[2] \rightarrow \text{CHAN}). \end{aligned}$$

Mehrfache Indizes, arithmetische Ausdrücke und Deklarationen von Konstanten und Bereichen:

Beispiel (SUM):

const N = 1
range T = 0..N
range R = 0..2*N

SUM = (in[a:T][b:T] $\rightarrow$ TOTAL[a+b]),
TOTAL[s:R] = (out[s] $\rightarrow$ SUM).



Prozesse mit bewachten Aktionen

Boolescher Ausdruck
ähnlich zu Java

(when B a → E | b → F).

Die Aktion a kann nur dann gewählt werden, wenn die Bedingung B erfüllt ist.

Bemerkung:

- Bewachte Aktionen können bei der Deklaration indizierter, lokaler Prozesse verwendet werden:

  P[i:T][j:R] = (when B a → E | ...)

  lokalen

- Die Bedingung B darf an Variablen höchstens die Indizes der Prozessdeklaration und formale Parameter (von parametrisierten Prozessen) enthalten.
- Prozessdeklarationen mit bewachten Aktionen sind Kurzschreibweisen für Prozessdeklarationen ohne bewachte Aktionen.

Beispiel (Countdown):

$$\begin{aligned}
\text{COUNTDOWN}(\overbrace{N}^{=2}) &= (\text{start} \to \text{CD}[\cancel{2}^{\,N}]), \\
\text{CD}[i{:}0..\cancel{2}_{\,N}] &= (\text{when } (i > 0)\ \text{tick} \to \text{CD}[i\text{-}1] \\
&\quad\ |\ \text{when } (i == 0)\ \text{beep} \to \text{STOP} \\
&\quad\ |\ \text{stop} \to \text{STOP}).
\end{aligned}$$

ist <u>Kurzschreibweise</u> für:

$$\begin{aligned}
\text{COUNTDOWN} &= (\text{start} \to \text{CD}[2]), \\
\text{CD}[2] &= (\text{tick} \to \text{CD}[1] \\
&\quad\ |\ \text{stop} \to \text{STOP}), \\
\text{CD}[1] &= (\text{tick} \to \text{CD}[0] \\
&\quad\ |\ \text{stop} \to \text{STOP}), \\
\text{CD}[0] &= (\text{beep} \to \text{STOP} \\
&\quad\ |\ \text{stop} \to \text{STOP}).
\end{aligned}$$

Parametrisierte Prozesse

Wiederverwendung

- Parametrisierte Prozesse erlauben eine generische Definition von Prozessen.
- Der Prozessparameter muss bei der Deklaration einen "Defaultwert" erhalten (sonst kein endliches LTS).
- Der Prozess kann jedoch für einen beliebigen aktuellen Parameter in einer anderen Prozessdeklaration aufgerufen werden.

Beispiel (COUNTDOWN(N)):

$$
\begin{array}{lcl}
\text{COUNTDOWN(N=2)} & = & (\text{start} \rightarrow \text{CD[N]}), \\
\text{CD[i:0..N]} & = & (\text{when}\,(\text{i} > 0)\,\text{tick} \rightarrow \text{CD[i-1]} \\
 & & |\,\text{when}\,(\text{i} == 0)\,\text{beep} \rightarrow \text{STOP} \\
 & & |\,\text{stop} \rightarrow \text{STOP}).
\end{array}
$$

Beachte:

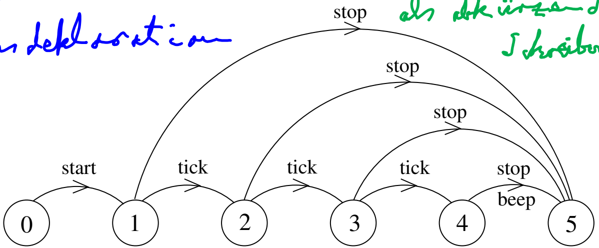
Parameter werden nur in *globalen* Prozessdeklarationen verwendet,
Indizes nur in *lokalen* Prozessdeklarationen.

Anwendung: ||MY_COUNTDOWN = COUNTDOWN(3)

aktueller Parameter

Prozessausdruck als abkürzende Schreibweise

Prozessdeklaration



Zusammenfassung abkürzender Notationen: (bis jetzt)

- indizierte Aktionen,
- indizierte Prozesse in lokalen Prozessdeklarationen,
- bewachte Aktionen,
- parametrisierte Prozessdeklarationen und deren Aufrufe

2.4 Semantik von Prozessausdrücken und starke Äquivalenz

Es bezeichne $\mathcal{E}$ die Menge aller Prozessausdrücke und $\mathcal{T}$ die Menge aller (endlichen) LTSe über `States` und `ACT`.

Die **Semantik von Prozessausdrücken** ist gegeben durch eine Funktion

$$\boxed{\text{lts}: \mathcal{E} \longrightarrow \mathcal{T},}$$

die gemäß der Struktur von Prozessausdrücken folgendermaßen induktiv definiert ist:

1. $\text{lts(STOP)} =_{def} (\{q_0\}, \emptyset, \emptyset, q_0),\ q_0 \in \texttt{States}$

2. Sei P $\in$ PID ein Prozessidentifikator.
   Zur Definition von lts(P) muss eine Prozessdeklaration $\boxed{\text{P} = \text{E.}}$ gegeben sein.
   Dann definieren wir:

$$\boxed{lts(P)} =_{def} \begin{cases} \text{lts(E)}, & \text{falls P} \notin \text{FV(E)} \\ \underbrace{\text{lts(rec(P=E))}}_{\text{vgl. 5.}}, & \text{falls P} \in \text{FV(E)} \end{cases}$$

3. Aktionspräfix:

Sei $\mathrm{lts}(\mathrm{E}) = (S, A, \Delta, q_0)$.

Dann definieren wir:

$\boxed{\mathrm{lts}(\boxed{(\mathrm{a} \to \mathrm{E})})} =_{def} (\underline{S \cup \{p_0\}}, A \cup \{a\}, \Delta \cup \{\underbrace{(p_0, a, q_0)}\}, \underline{p_0})$,

wobei $\underline{p_0 \in \mathtt{States} \setminus S}$.

4. Auswahl:

Für $i = 1, \ldots, n$ sei $\mathrm{lts}(\mathrm{E}_i) = (S_i, A_i, \Delta_i, q_i)$ mit paarweise disjunkten S_i.

Dann definieren wir:

$$\mathrm{lts}(\boxed{(\mathrm{a}_1 \to \mathrm{E}_1 \mid \ldots \mid \mathrm{a}_n \to \mathrm{E}_n)}) =_{def}$$

$$\begin{aligned}
&(S_1 \cup \ldots \cup S_n \cup \{p_0\},\\
&A_1 \cup \ldots \cup A_n \cup \{a_1, \ldots, a_n\},\\
&\Delta_1 \cup \ldots \cup \Delta_n \cup \{\underbrace{(p_0, a_1, q_1)}, \ldots, \underbrace{(p_0, a_n, q_n)}\},\\
&p_0),
\end{aligned}$$

wobei $\underline{p_0 \in \mathtt{States} \setminus (S_1 \cup \ldots \cup S_n)}$.

5. Rekursive Prozessausdrücke:

Sei P ein Prozessidentifikator und E ein Prozessausdruck, so dass P $\in$ FV(E).
Sei lts(E) $= (S, A, \Delta, q_0)$ wobei wir annehmen lts(P) $= (\{q_P\}, \emptyset, \emptyset, q_P)$.
Dann definieren wir:

$$\text{lts}(\boxed{\text{rec}(\text{P} = \text{E})}) =_{def} (S \setminus \{q_P\}, A, \Delta_{rec}, q_0),$$

wobei

$$\Delta_{rec} = \{(q, a, q') \mid (q, a, q') \in \Delta \text{ und } q, q' \neq q_P\} \cup \{(q, a, q_0) \mid (q, a, q_P) \in \Delta\}.$$

Dadurch wird in Δ_{rec} ein Zyklus eingeführt

Falls E = P, def. wir

lts(rec(P = P)) $=_{def}$ lts(STOP)

Definition (Starke Bisimulation):

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

(1) $q \xrightarrow{a}_{\Delta} p \implies \exists\, p' \in S'$ mit $q' \xrightarrow{a}_{\Delta'} p'$ und $(p, p') \in R$.

(2) $q' \xrightarrow{a}_{\Delta'} p' \implies \exists\, p \in S$ mit $q \xrightarrow{a}_{\Delta} p$ und $(p, p') \in R$.

Bemerkung:

Sei T = (S, A, Δ, q_0) $\in \mathcal{T}$.
Die Identität $= \subseteq S \times S$ ist eine starke Bisimulation zwischen T und T.

$\{ (s, s) \mid s \in S \}$

Definition (Starke 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 **stark äquivalent**, geschrieben $\boxed{T \sim T'}$ wenn gilt:

(a) T und T' haben dieselben Aktionen, d.h. $A = A'$.

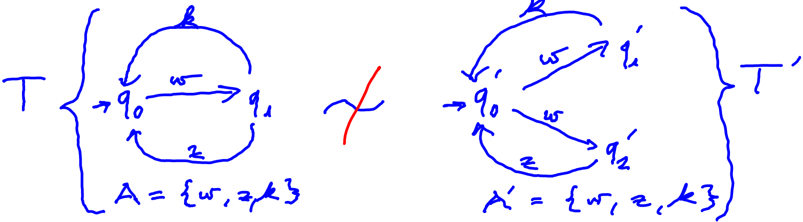
(b) Es gibt eine starke Bisimulation $R \subseteq S \times S'$ zwischen T und T', so dass $(q_0, q_0') \in R$.

Lemma:

$\sim$ ist eine Äquivalenzrelation auf $\mathcal{T}$. Bew: ÜZ

Bemerkung:

Stark äquivalente LTSe haben dieselben Abläufe.
Die Umkehrung gilt jedoch nicht (vgl. Beispiel Münzwurf von oben).



Beispiele: (Werden in der Vorlesung eingetragen.)

Definition (Reach(T)):

Sei $T = (S, A, \Delta, q_0)$ ein LTS.
Das **reachable Sub-LTS** von T ist gegeben durch $\text{Reach}(T) = (S_r, A, \Delta_r, q_0)$, wobei

- $S_r \subseteq S$ die kleinste Teilmenge von S ist, so dass gilt:
  (0) $q_0 \in S_r$,
  (1) $q \in S_r$ und $q \xrightarrow{a}_{\Delta} p \implies p \in S_r$,
- $\Delta_r = \{(q,a,p) \in \Delta \mid q, p \in S_r\}$.

T heißt reachable, wenn $T = \text{Reach}(T)$.

Bemerkung:

Für alle FSP-Ausdrücke E ist lts(E) reachable.

Lemma:

Seien $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$. Es gilt:

1. $T \sim \text{Reach}(T)$,
2. $T \sim T' \iff \text{Reach}(T) \sim \text{Reach}(T')$.

Definition (Starke Äquivalenz von Prozessen):

Zwei Prozesse E, F $\in \mathcal{E}$ sind **stark äquivalent** (stark bisimular), geschrieben E $\sim$ F, wenn gilt: lts(E) $\sim$ lts(F).

Beispiele (Algebraische Gesetze):

Seien a, b $\in$ $\mathtt{ACT}$ und E, F Prozessausdrücke.

- $(a \to E \mid b \to F) \sim (b \to F \mid a \to E)$ *(Kommutativgesetz für Auswahl)*
- $(a \to E \mid a \to E) \sim (a \to E)$ *(Idempotenzgesetz für Auswahl)*
- $E \sim F \implies (a \to E) \sim (a \to F)$ *(Kongruenzregel bzgl. Aktionspräfix)*
- $E \sim E'$ und $F \sim F' \implies (a \to E \mid b \to F) \sim (a \to E' \mid b \to F')$ *(Kongruenzregel bzgl. Auswahl)*

2.5 Implementierung von Prozessen

Betriebssystem-Prozesse und Threads

Ein *BS-Prozess* besitzt einen eigenen Adressraum und wird repräsentiert durch

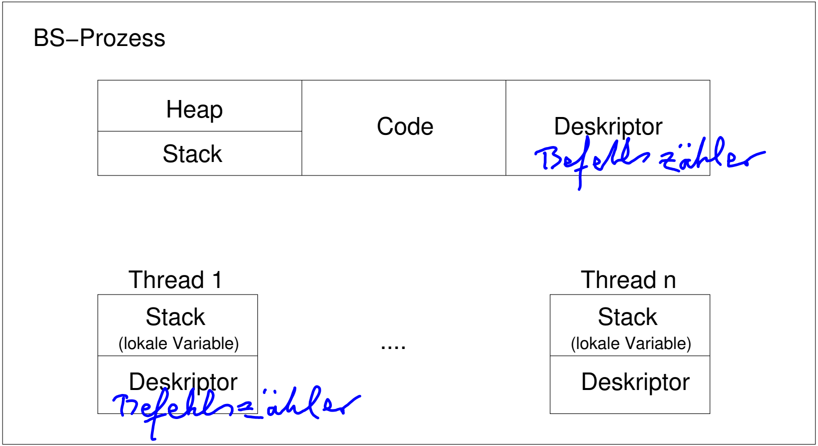
- Daten (globale und lokale Variable);
  die lokalen Variablen sind in einem Keller organisiert,
  die globalen Variablen in einem Heap
- Code (Befehle)
- Deskriptor (organisatorische Daten und Werte der Maschinenregister)

Ein BS-Prozess ist ein "schwergewichtiger Prozess" (z.B. Ausführung eines Anwendungsprogramms).

Ein *Thread* ist ein "leichtgewichtiger Prozess", der innerhalb eines BS-Prozesses (evt. parallel zu anderen Threads) abläuft.

- Jeder Thread besitzt einen eigenen Stack für seine lokalen Variablen und einen eigenen Deskriptor.
- Der Thread-Code ist im Code-Segment des BS-Prozesses enthalten.
- Jeder Thread hat Zugriff auf die globalen Variablen des BS-Prozesses.

Betriebssystem-Prozess



Realisierung von Threads in Java

Threads werden in Java durch Objekte der Klasse “Thread“ (im Paket java.lang) realisiert.

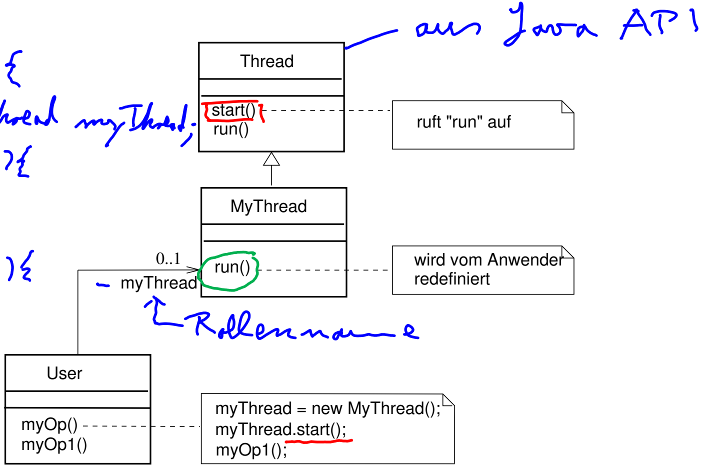
Thread
start() run() ...

Es bezeichne `t` ein Objekt der Klasse `Thread` oder einer Subklasse von `Thread`.

- Der Methodenaufruf `t.start();` bewirkt, dass das Thread-Objekt `t` aktiviert wird und seine `run`-Methode aufgerufen wird.
- Der aufrufende Thread setzt dann seine Tätigkeit parallel zur Ausführung der `run`-Methode des Threads `t` fort.

1. Realisierung von Threads mittels Vererbung

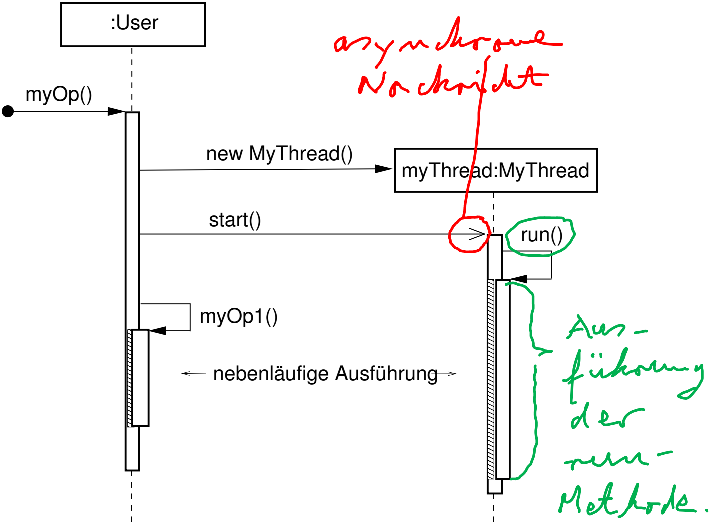
```
class User {
  private MyThread myThread;
  void myOp(){
    ...
  }
  void myOp1(){
    ...
  }
}
```



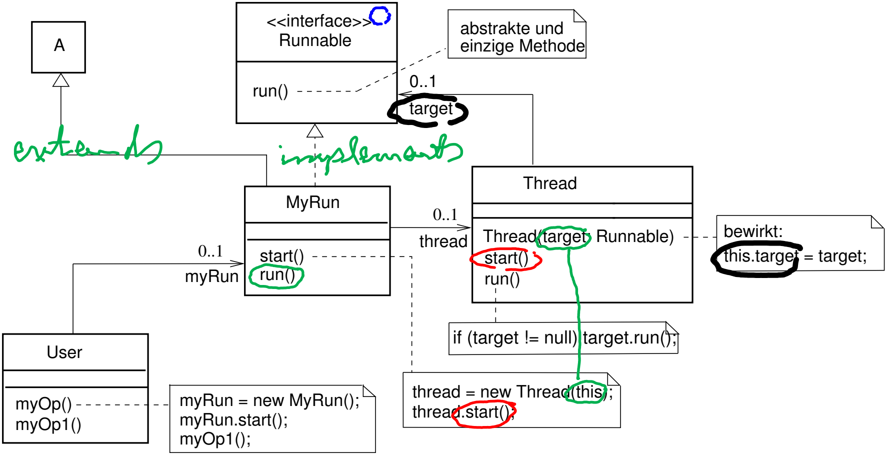
Beachte:

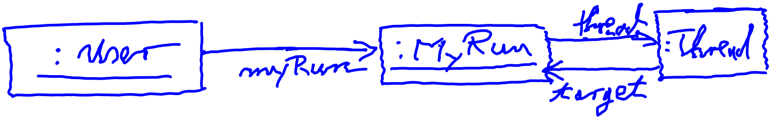
"MyThread" kann <u>nicht</u> Erbe einer weiteren Klasse sein,
da in Java Mehrfachvererbung nicht möglich ist!

Sequenzdiagramm mit Objekt der Klasse MyThread

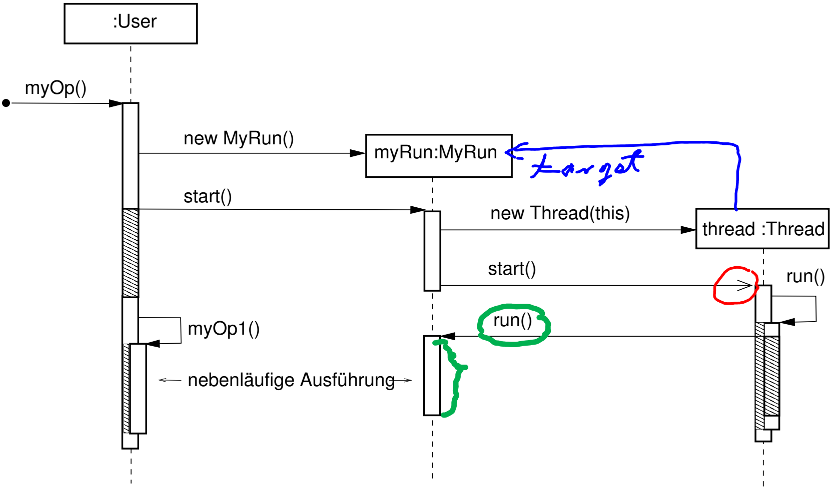


2. Realisierung von Threads durch Verwendung des Interfaces “Runnable“

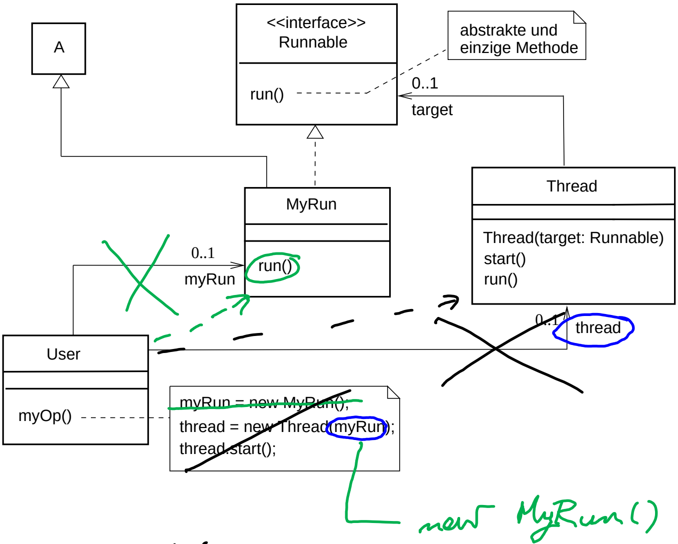




Sequenzdiagramm mit Objekt der Klasse MyRun



Klassendiagramm mit Interface Runnable (Variante)



new Thread(new MyRun()).start();

Beispiel (Implementierung des Countdown-Prozesses):

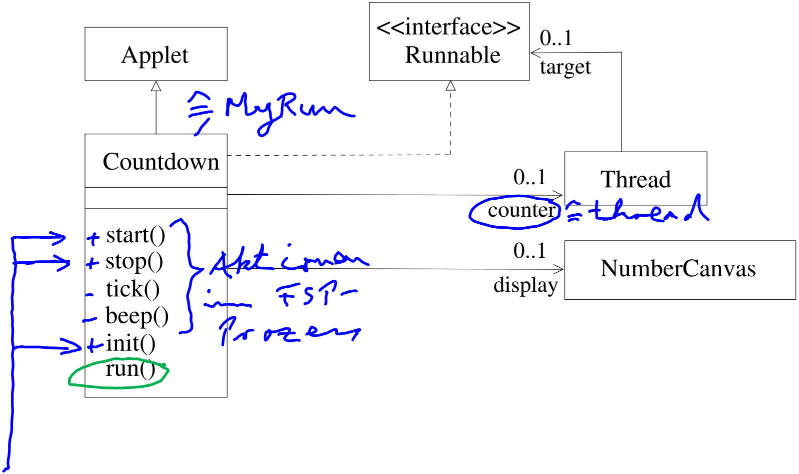
$$
\begin{aligned}
\text{COUNTDOWN(N=10)} \quad &= \quad (\text{start} \rightarrow \text{CD[N]}),\\
\text{CD[i:0..N]} \quad &= \quad (\text{when}\,(\text{i} > 0)\,\text{tick} \rightarrow \text{CD[i-1]}\\
&\qquad \mid \text{when}\,(\text{i} == 0)\,\text{beep} \rightarrow \text{STOP}\\
&\qquad \mid \text{stop} \rightarrow \text{STOP}).
\end{aligned}
$$

return in Java

Aktionen:

- externe: start, stop ≙ input
- interne: tick, beep ≙ output

Klassendiagramm der Implementierung



überschriebene Applet-Methoden

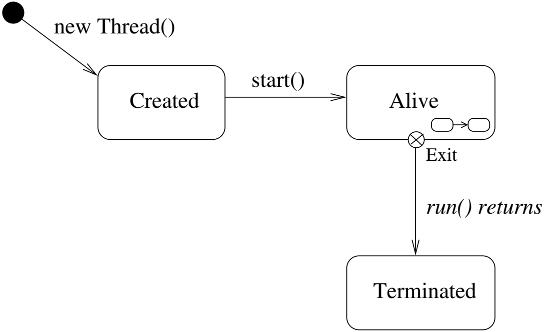
Java-Implementierung

```
public class Countdown extends Applet implements Runnable {
   final static int N = 10;
   int i;
   Thread counter;
   AudioClip beepsound, ticksound;
   NumberCanvas display;

   public void start() {
     i = N;
     counter = new Thread(this);   vgl. Schema
     counter.start();
   }
   public void stop() {
      counter = null;
   }
   private void tick() {...}
   private void beep() {...}
   public void init() {...}

   public void run() {
      while (true) {
         if (counter == null) return;
         if (i == 0) {beep(); return;}
         if (i  > 0) {tick(); i = i-1;}
      }
   }
}
```

Lebenszyklus eines Java-Threads



Unterzustände des Alive-Zustands