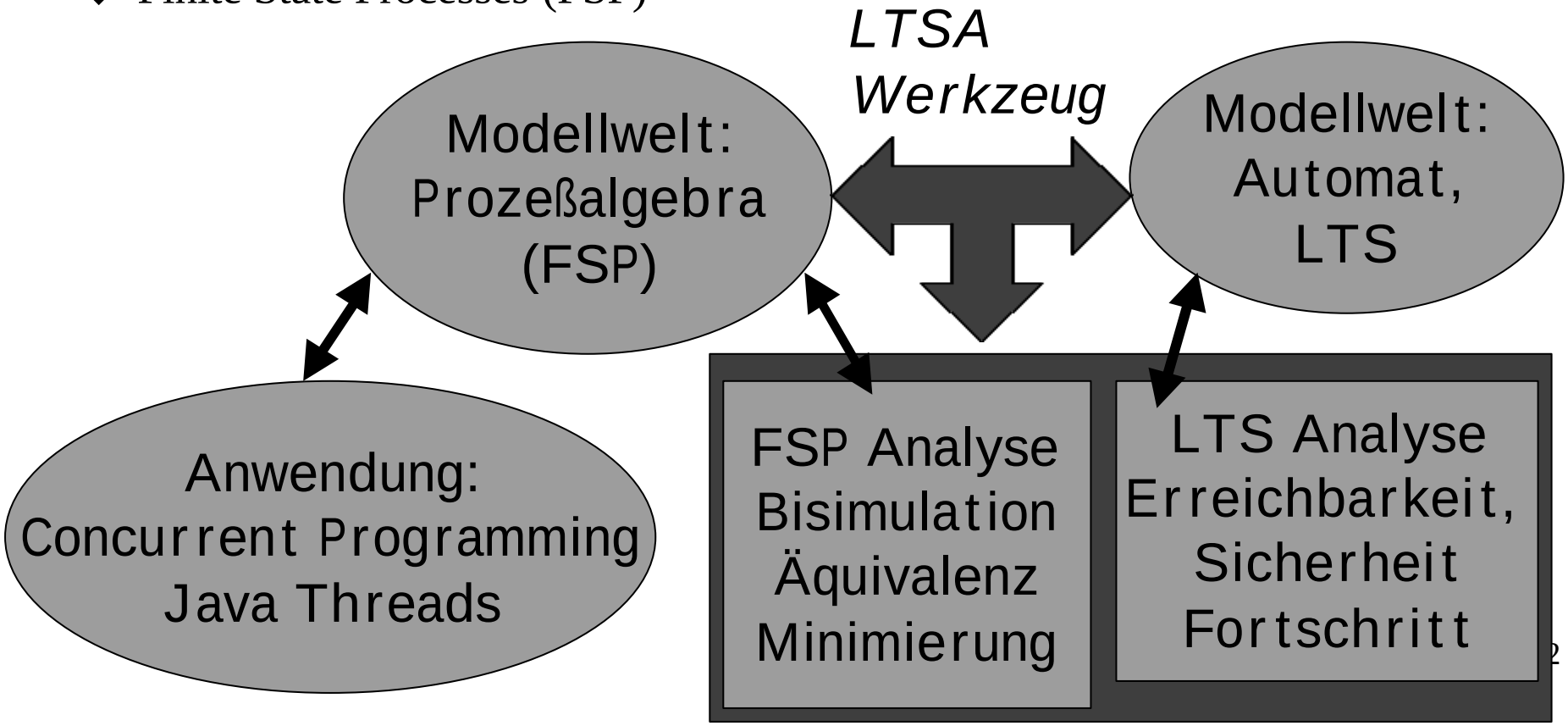
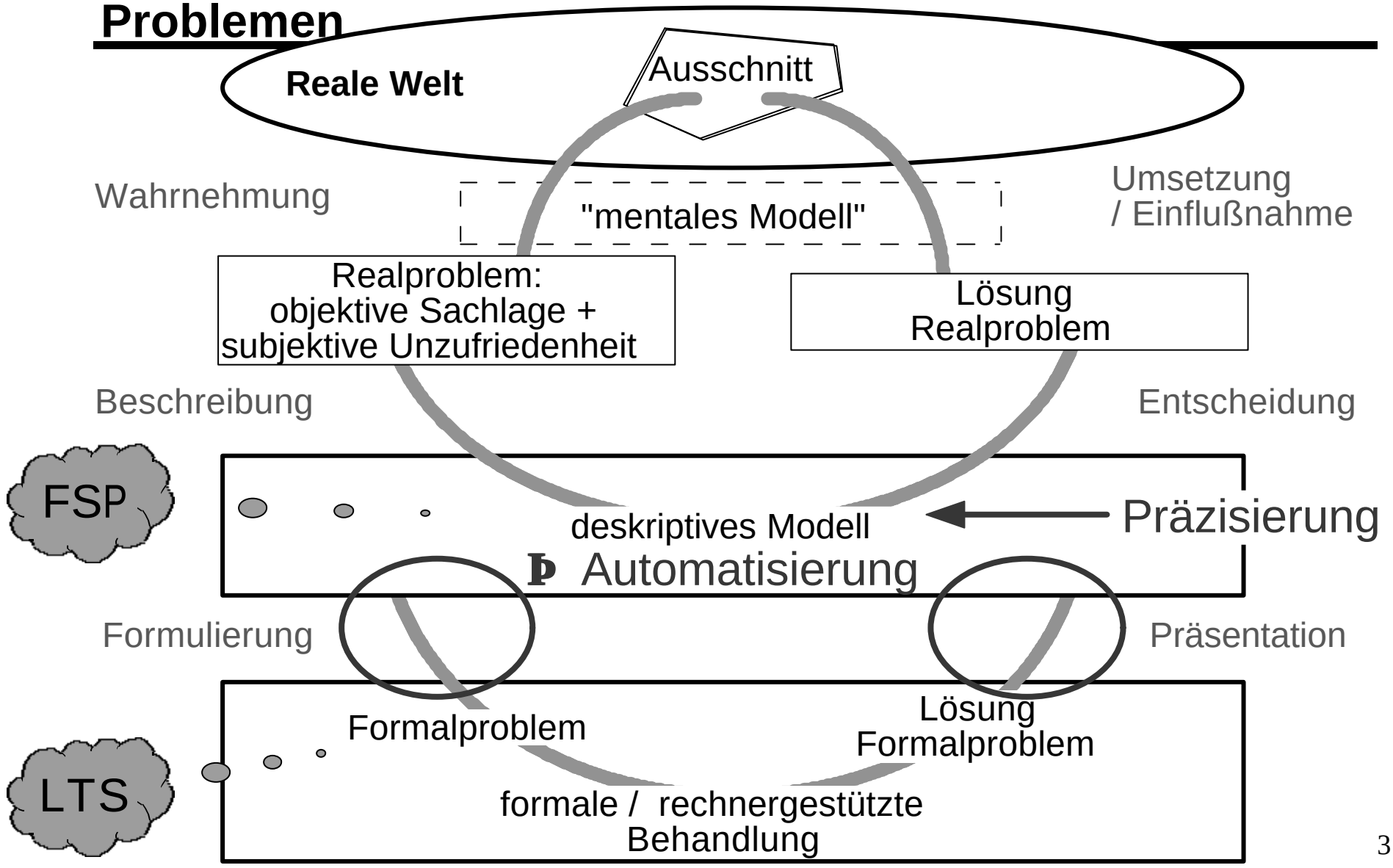
Modellierung von nebenläufigen Prozessen mit Labelled Transition Systems (LTS) und Finite State Processes (FSP)

Übersicht

- Modellierung
- bekannte Notation: endliche Automaten
- Labelled Transition Systems (LTS) vs endliche Automaten
- Finite State Processes (FSP)



Modellierung => Modellgestützte Lösung von Problemen



Hauptmerkmale eines Modells (Stachowiak 1973)

1) Abbildungsmerkmal

- Modelle sind Abbilder eines vorhandenen oder Vorbilder eines zu schaffenden Originals
- Originale können selbst Modelle sein
- zu jedem Modell gehört eine Abbildung, welche die Individuen und Attribute des Originals auf diejenigen des Modells abbildet
- es kann verschiedene Modelle eines Originals geben

2) Verkürzungsmerkmal, Abstraktion

- Modelle erfassen i.d.R. nicht alle Individuen und Attribute des Originals
- Es werden nur als relevant (wichtig/nützlich/notwendig) wahrgenommene Eigenschaften des Originals modelliert.
- Das Modell kann Individuen und Attribute enthalten, die keine Entsprechung im Original haben.

Hauptmerkmale eines Modells (Stachowiak 1973)

3) Pragmatisches Merkmal, Zweckorientierung

- Original und Modell sind einander nicht aus sich selbst heraus zugeordnet.
- Jedes Modell ist von einem/mehreren Menschen für einen spez. Zeitraum und einen spezifischen Verwendungszweck geschaffen.
- Modelle sind nicht a priori richtig oder falsch, sonder in Anbetracht der Zweckorientierung gut oder weniger gut geeignet.
- Es dürfen keine Modellattribute ausgewertet werden, die keine Entsprechung im Original haben. Entsprechende Auswertungen sind nicht wohldefiniert und können zu unrichtigen Aussagen führen.

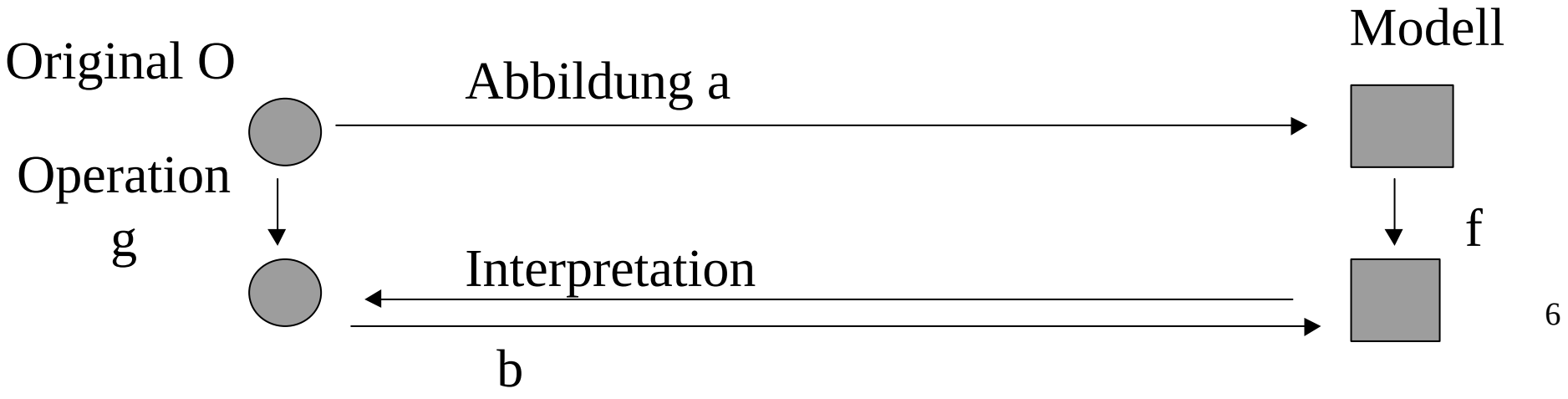
Operationen auf Modellen

Auf Modellen werden Operationen/Modifikationen/Experimente durchgeführt, um aus den resultierenden Modelleigenschaften Rückschlüsse auf Resultate im Originalsystem zu ziehen.

Nur solche Modelloperationen sind zulässig

- zu denen es eine entsprechende Operation am Original gibt
- deren resultierende Attribute auf entsprechende Attribute des Originals abbildbar sind.

Konsistenzbedingung: $M' = f(a(O)) = b(g(O))$



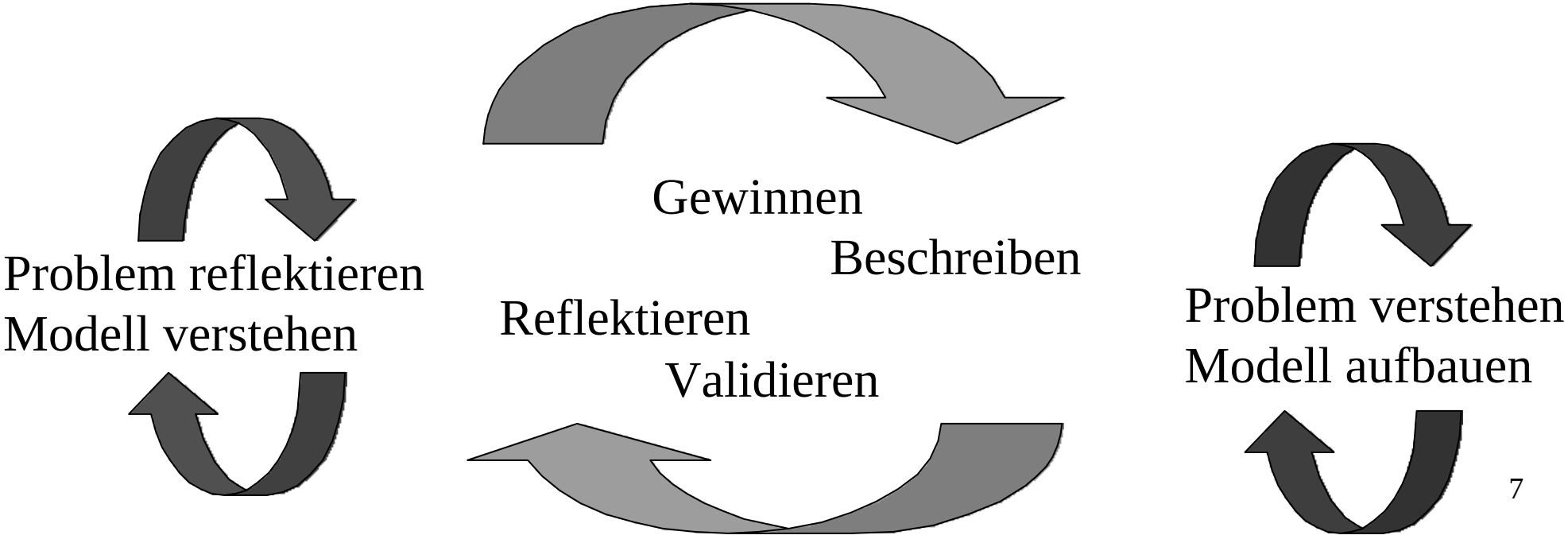
Modellbildung - Prozeß zur Erstellung eines Modells

Zwei Rollen:

Wissensträger - Person, welche das Wissen über das Originalsystem hat

Modellierer - Person, welche das Modell erstellt

möglich: je Rolle mehrere Personen, eine Person hat beide Rollen



Modellierung

Ein Modell ist Abbild/Vorbild für ein reales System.

Anwendung: Vertrauen in die Angemessenheit und Validität eines Entwurfs/Designs rechtfertigen/bilden.

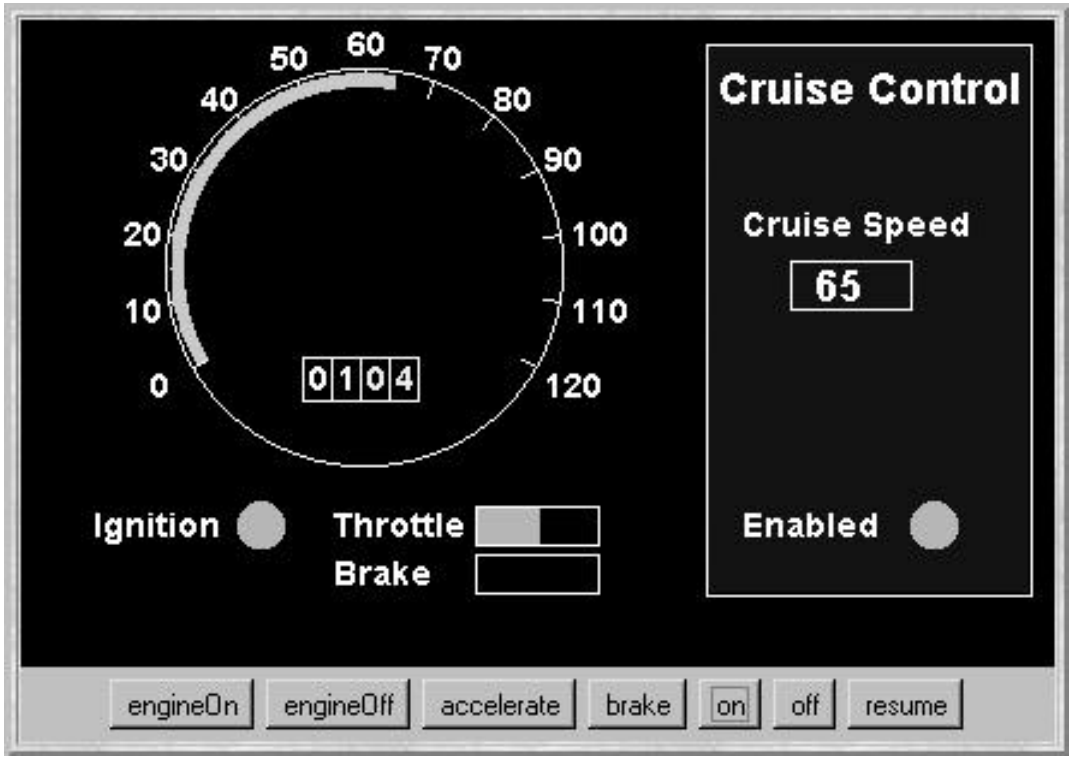
- Blickwinkel: Nebenläufigkeit
- Hilfsmittel: Animation zur Visualisierung des Verhaltens
- Ziel: automatische Verifikation von Eigenschaften (Sicherheit, Fortschritt)

Beschreibung von Modellen

durch Automaten/Labelled Transition (**LTS)** als Graph,
durch Finite State Processes (**FSP**) textuell.

SW Werkzeug (u.a.) zur Spezifikation & Analyse: ***LTSA***

Beispiel: System zur Geschwindigkeitssteuerung



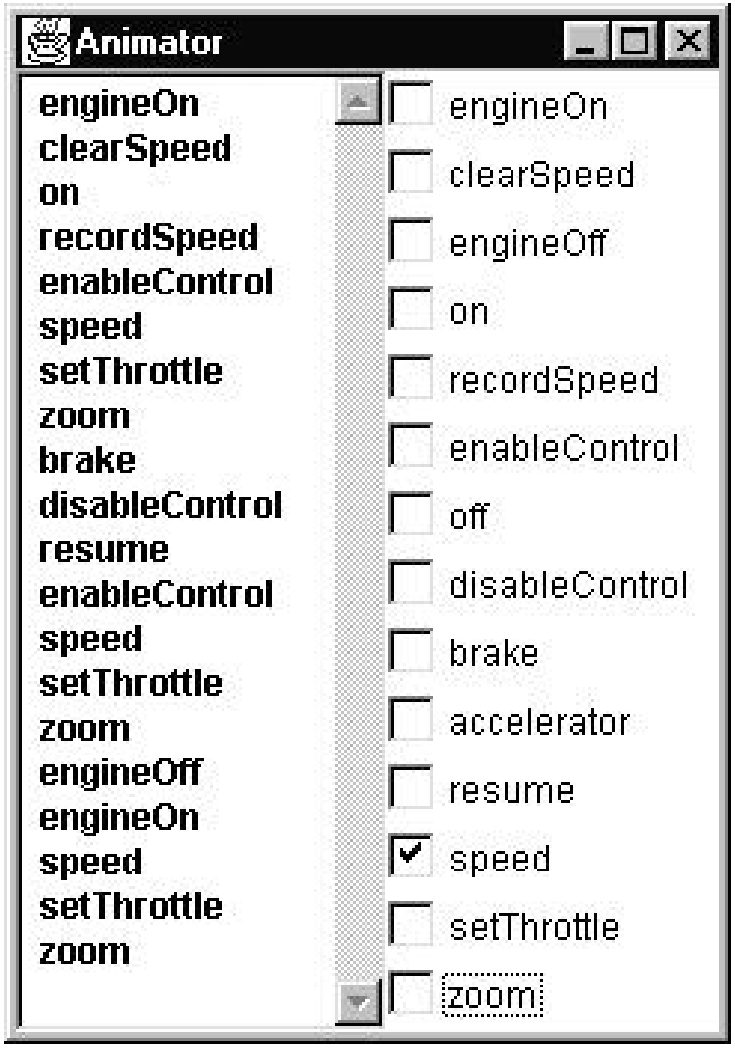
Reale Steuerung in einem Fahrzeug ist das (geplante) System.

Klassenhierarchie ist ein Modell der Softwarearchitektur für dieses System mit dem Ziel der Strukturierung in Komponenten und der Verdeutlichung der Abhängigkeiten.

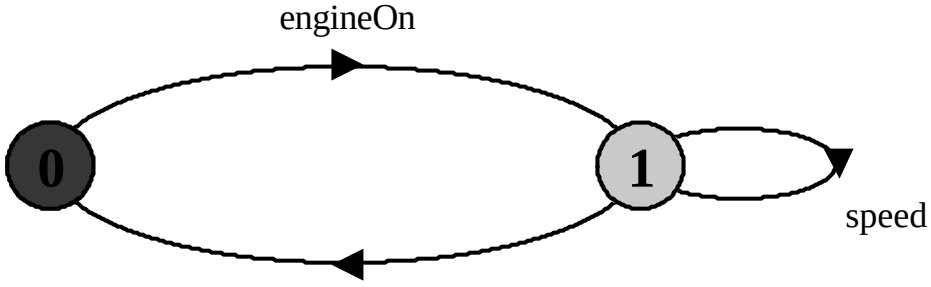
Modellierung der einzelnen Threads und deren Interaktion erfordert Notation, die den Zustand jedes Threads, seine möglichen Aktionen und Interaktionen mit anderen Threads erfaßt. Ziel einer Modellierung ist die Absicherung eines funktional korrekten Verhaltens.

(Ergänzung)

Möglicher Nutzen: Emulation des Verhaltens zur Visualisierung und zum Testen



***LTSA* Animator erlaubt Auswahl von Aktionen, Aufbau einer Sequenz.**



LTS: Beobachter für Geschwindkeit

Ziel:

Spezifikation, Animation, Verifikation von Modellen mit FSP und LTS.

Notationen für dynamische Systeme

Aus Grundstudium bekannte Notation:
endliche Automaten (Mealy, Moore)

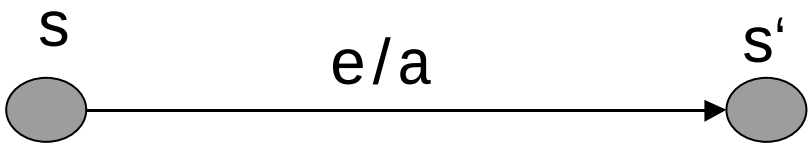
hier:
ähnlich, aber etwas weniger auf I/O fokussiert
Labelled Transition System (LTS)

und als formale Sprache, Prozeßalgebra
Finite State Processes (FSP)

Mealy Automat (zur Erinnerung)

Definition: (endlicher) Mealy-Automat $(S, s_0, E, A, \mathrm{f}, \mathrm{g})$
S endliche Menge der Zustände, $s_0 \in S$ Anfangszustand,
E/A endliche Mengen der Ein/Ausgaben
$\mathrm{f} : \mathrm{S} \times \mathrm{E} \to \mathrm{S}$ Überführungsfunktion
$\mathrm{g} : \mathrm{S} \times \mathrm{E} \to \mathrm{A}$ Ausgabeabbildung, surjektiv

Darstellung tabellarisch oder als
gerichteter Graph mit Kantenbeschriftung



Klassisches Beispiel: Getränkeautomat

(Ergänzung)

Kennzeichen, Querbezüge (formale Sprachen, Compilerbau)

- Kennzeichen: Zustandsbeschreibung global, explizit aufgezählt, Fokus I/O, taugt zur Modellierung eines einzelnen Prozesses und Interaktion mit Umwelt
- Theoretische Informatik: formale Sprachen
  - Automaten als Akzeptor für eine Eingabesprache, die aus einer Grammatik erzeugt wird, (Stichwort: Chomsky-Hierarchie) daher Fokussierung auf Ein/Ausgabeverhalten in der Notation
- Compilerbau:
  - praktischer Einsatz zur Erzeugung von Parsern
  - Sprachen mit LALR-1 Grammatiken durch Automaten erkennbar
  - Kennzeichen: Folgezustand mit 1 Lookahead feststellbar
  - Algorithmen zur Erzeugung eines Automaten für LALR-1 Grammatik bekannt und in Parsergeneratoren verwendet: flex/bison, java-cup

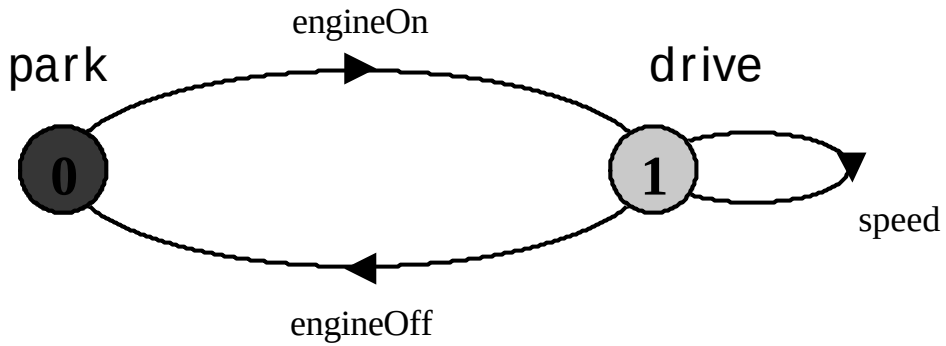
Labelled Transition System (LTS)

Definition: LTS $P = (S, S_0, A, \Delta, X, \mathrm{p})$

S	Zustandsmenge,
S_0	Menge der Anfangszustände,
$A = aP \cup \{t\}$	Menge der Transitionsbezeicher, Labels
$\Delta \subseteq S \times A \times S$	Transitions-, Zustandsübergangsrelation
X	Menge von Zustandsparametern
$\mathrm{p} : \mathrm{S} \rightarrow 2^{\mathrm{X}}$	Zuordnung von Zustandsparametern zu jedem Zustand

Allg. Labels $aP \subseteq L$ $A \subseteq Act = L \cup \{t\}$

Beispiel LTS



Zustandsmenge $S = \{0\}$

Anfangszustände $S_0 = \{0\}$

Transitionsbezeichner $A = \{engineOn, engineOff, speed, t\}$

Transitionsrelation

$$\Delta = \{(0, engineOn, 1), (1, engineOff, 0), (1, speed, 1)\}$$

Zustandsparameter $X = \{park, drive\}$

Zuordnung $p(0) = \{park\}, p(1) = \{drive\}$

Abgrenzung LTS zu Mealy-Automat

- LTS: keine Unterscheidung bzgl Ein/Ausgabe
- LTS: Transitionsbezeichner stellen auf Beobachtbarkeit ab,
  - (interne) Tau-Transitionen, beobachtbare andere Transitionen
  - Transitionsbezeichner mit Definition über Alphabet umständlich
    - Ursache: Komposition von Prozessen und Verbindung zu FSP
  - erlauben Aussagen über Folgen von Zustandsübergängen
- LTS: Zustandsparameter erlauben Aussagen über Zustandsmengen
  - untypisch im Kontext von Prozeßalgebren, wie FSP
  - typisch für Petri Netze und Modelchecking
- LTS&Mealy
  - Zustand s beschreibt Gesamtzustand (global),
  - explizite Aufzählung der Zustandsmenge
  - keine Möglichkeit explizit Nebenläufigkeit auszudrücken
  - keine Datenstrukturen
  - kein Zeitbegriff

Prozeßalgebren

- Zahlreiche Varianten:
  - Milners Calculus of Communicating Systems (CCS)
  - Hoares Communicating Sequential Processes (CSP)
  - Magee&Kramer Finite State Processes (FSP)
- Idee: Grammatik mit Ableitungsregeln beschreibt Dynamik des Models
  - sonst bei formalen Sprachen: Herleitung von Terminalen ist das Ziel
  - hier: mögliche Ableitungen (auch unendlich) sind als Weg interessant
- typische Operationen
  - Prefix
  - Choice
  - Recursion

  } für sequentielle Teilprozesse

  - Parallel Composition
  - Relabeling
  - Hiding

  } zur Komposition & Synchronisation

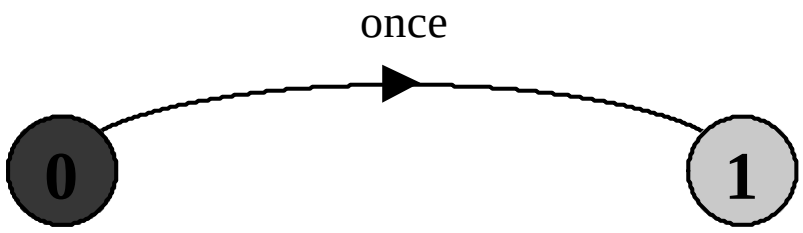
Finite State Processes (FSP)

- Notation zur Beschreibung einzelner Prozesse mittels
  - Action Prefix: erlaubt Transitionssequenzen (Befehlssequenzen)
  - Choice, Guarded Action: erlauben nicht-deterministische Auswahl und Fallunterscheidungen (If-then-else)
  - Rekursion: erlaubt Schleifen
  - Alphabeterweiterung: Erweiterung des Alphabets wg Komposition
- Notation zur Komposition von Prozessen
  - Parallel Composition: Prozesse nebeneinander ausführbar, Synchronisation über gleiche Transitionsbezeichner
  - Relabeling
  - Hiding, Interface
  - Hilfen zur Beschreibung gleichartiger Prozesse nebst Kommunikation
    - forall Replicator: Vervielfachung von gleichartigen Prozessen
    - Process Labeling: Prefix Identifier für einzelnen Prozeß
    - Process Sharing: Auswahl Identifier für potentielle Interaktion
    - Priority high/low: Bevorzugen/Benachteiligen von Labels für Interaktion

FSP - Action Prefix

Falls **a** eine Aktion und **P** ein Prozeß sind, dann beschreibt **(x-> P)** einen Prozeß, der initial Aktion **x** durchführt und sich dann **P** verhält.

```
ONESHOT = (once -> STOP).
```



ONESHOT state machine

(terminierender Prozeß)

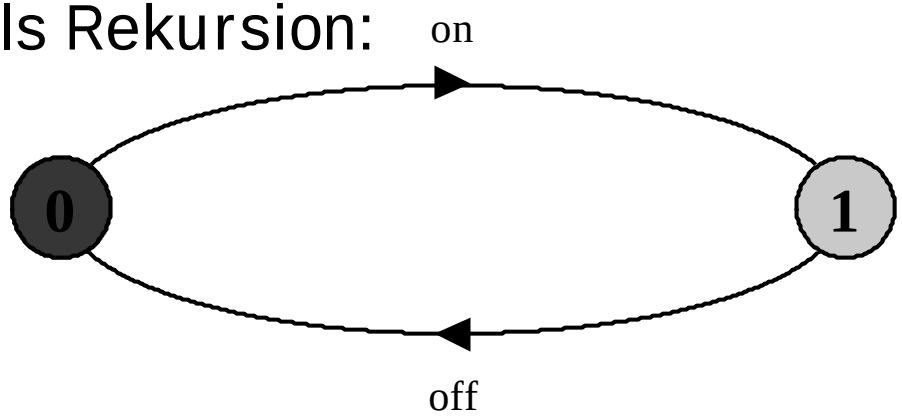
Konvention: **aktionen** beginnen mit Kleinbuchstaben

PROZESSE beginnen mit Großbuchstaben

FSP - Action Prefix & Rekursion

Wiederholung/Schleifen mittels Rekursion:

```
SWITCH = OFF,
OFF    = (on -> ON),
ON     = (off-> OFF).
```



Einsetzen liefert kürzere Beschreibung:

```
SWITCH = OFF,
OFF    = (on ->(off->OFF)).
```

Und nochmal:

```
SWITCH = (on->off->SWITCH).
```

FSP - Auswahl / Choice

Falls **x** und **y** Aktionen sind, dann ist **(x-> P | y-> Q)** ein Prozeß, der initial entweder **x** oder **y** durchführt. Falls **x** die erste Aktion ist, so definiert **P** das anschliessende Verhalten. Im Falle von **y** als erster Aktion, verhält sich der Prozeß infolge gemäß **Q.**

Wer oder was bewirkt die Entscheidung?

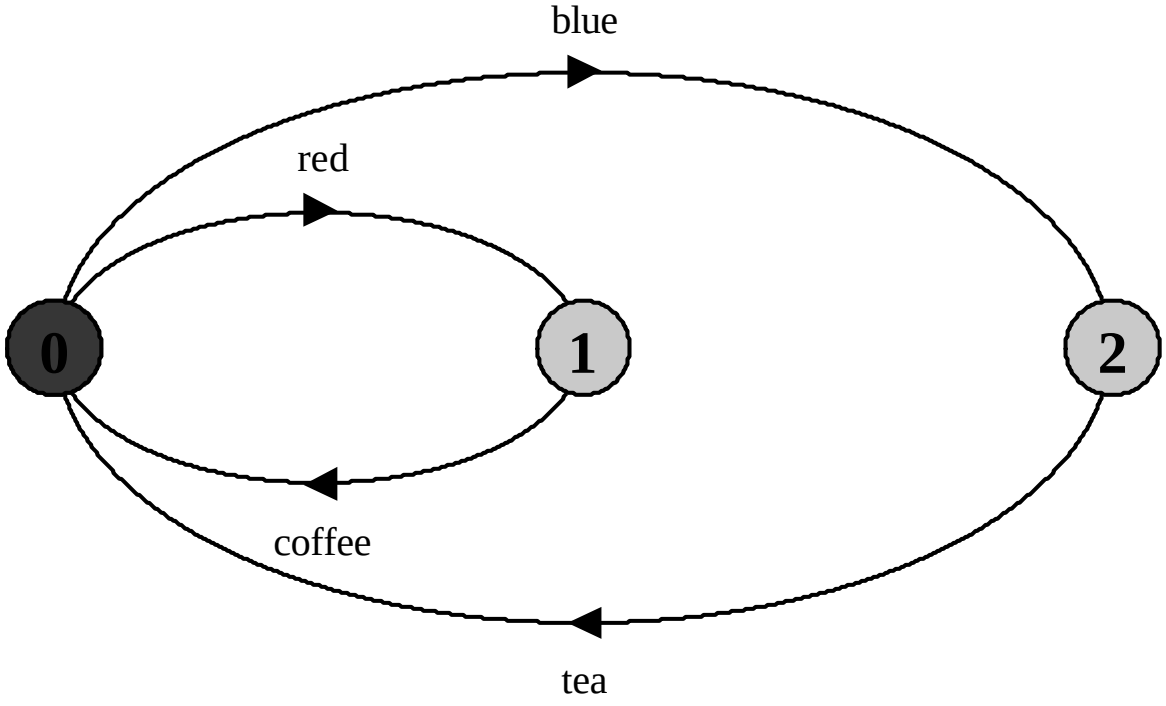
Gibt es einen Unterschied zwischen Ein- und Ausgabeaktionen?

FSP - Choice

FSP Modell eines Getränkeautomaten :

```
DRINKS = (red->coffee->DRINKS
         |blue->tea->DRINKS
         ).
```

LTSA erzeugt LTS:



Welche Sequenzen

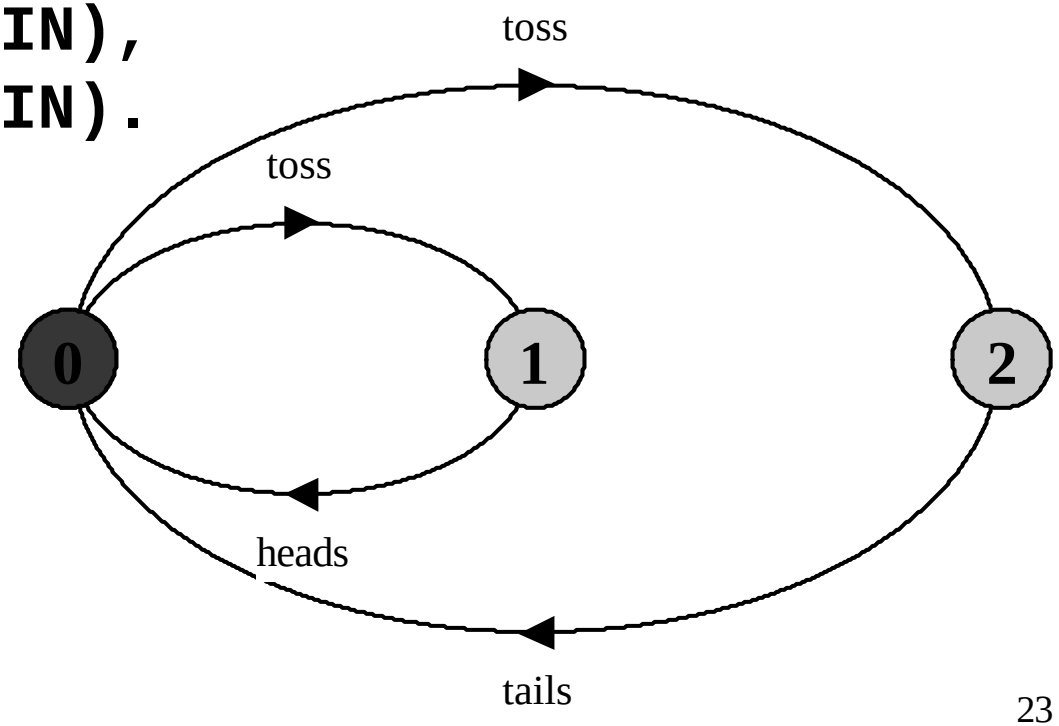
sind möglich?

Nicht-deterministische Auswahl

Prozeß (x-> P | x -> Q) beschreibt einen Prozeß, der nach Durchführung von **x** sich wie P oder Q verhält.

```
COIN = (toss->HEADS|toss->TAILS),
HEADS= (heads->COIN),
TAILS= (tails->COIN).
```

Münzwurf



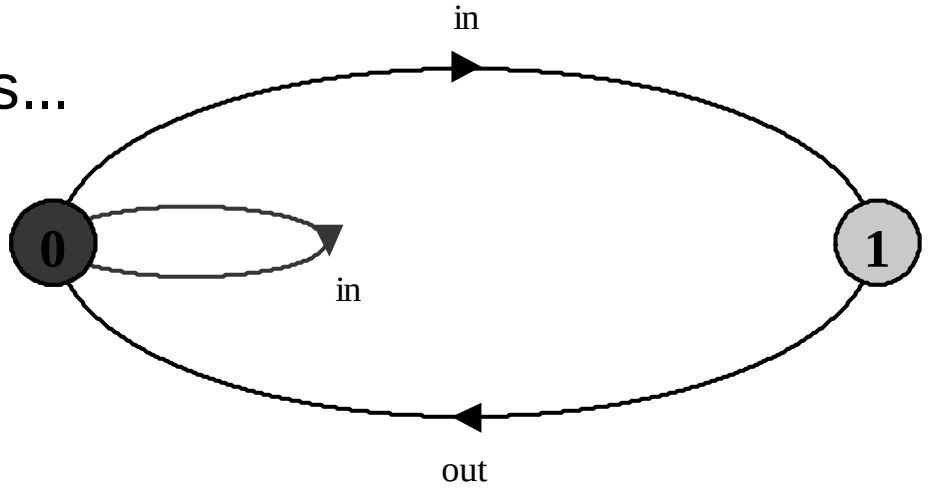
Welche Sequenzen

sind möglich?

Modellieren von unzuverlässigen Systemen

Wie läßt sich eine unzuverlässiger Kommunikationskanal modellieren, der eine Eingabeaktion **in** durchführen kann und dann im Falle eines Fehlers nichts tut und ansonsten eine Ausgabe mittels Aktion **out** signalisiert?

Mittels Non-Determinismus...




```
CHAN = (in->CHAN
       |in->out->CHAN
       ).
```

FSP - Prozesse und Aktionen mit Indizierung

Single Slot Puffer mit Eingabewertebereich 0 to 3 und identischer Ausgabe:

```
BUFF = (in[i:0..3]->out[i]-> BUFF).
```

äquivalent zu

```
BUFF = (in[0]->out[0]->BUFF
       |in[1]->out[1]->BUFF
       |in[2]->out[2]->BUFF
       |in[3]->out[3]->BUFF
       ).
```

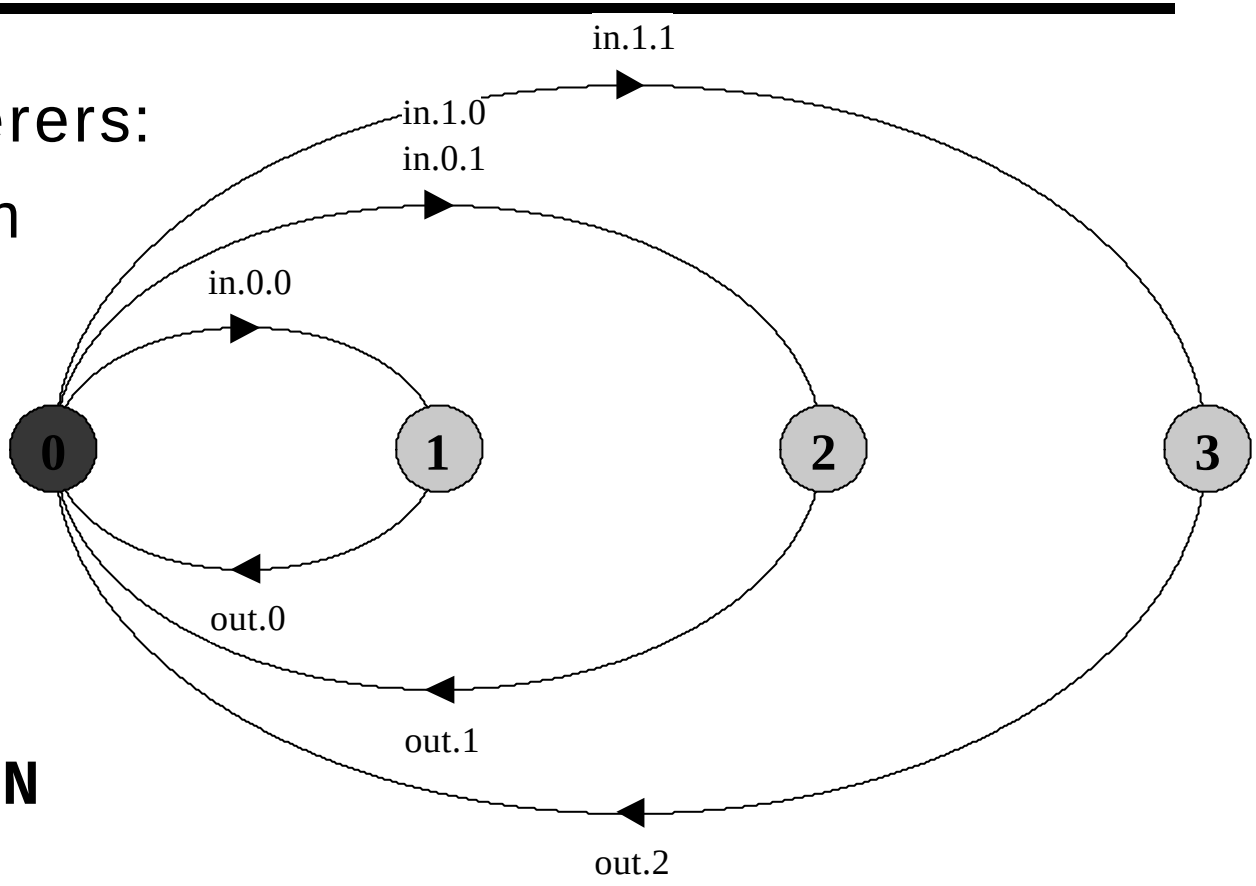
Oder zur Verwendung eines Prozeß Parameters mit Default-Wert:

```
BUFF(N=3) = (in[i:0..N]->out[i]-> BUFF).
```

FSP - Deklaration von Konstanten und Wertebereichen

Modell eines Addierers:

mittels Ausdrücken über Indizes

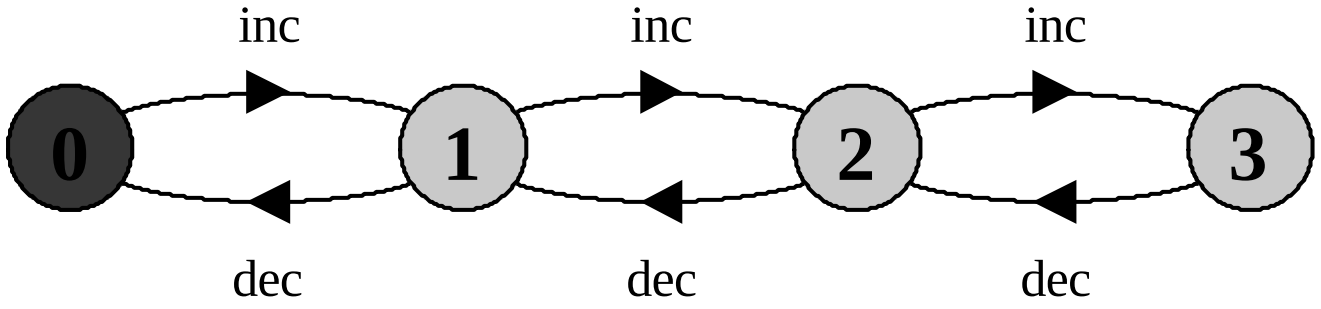



```
const N = 1
range T = 0..N
range R = 0..2*N

SUM         = (in[a:T][b:T]->TOTAL[a+b]),
TOTAL[s:R] = (out[s]->SUM).
```

FSP - Bedingte Aktionen (Guarded Actions)

(when B x -> P | y -> Q) bedeutet, daß falls Bedingung **B** erfüllt ist, Aktion **x** oder **y** stattfinden kann, anderenfalls kann lediglich **x** nicht ausgeführt werden.

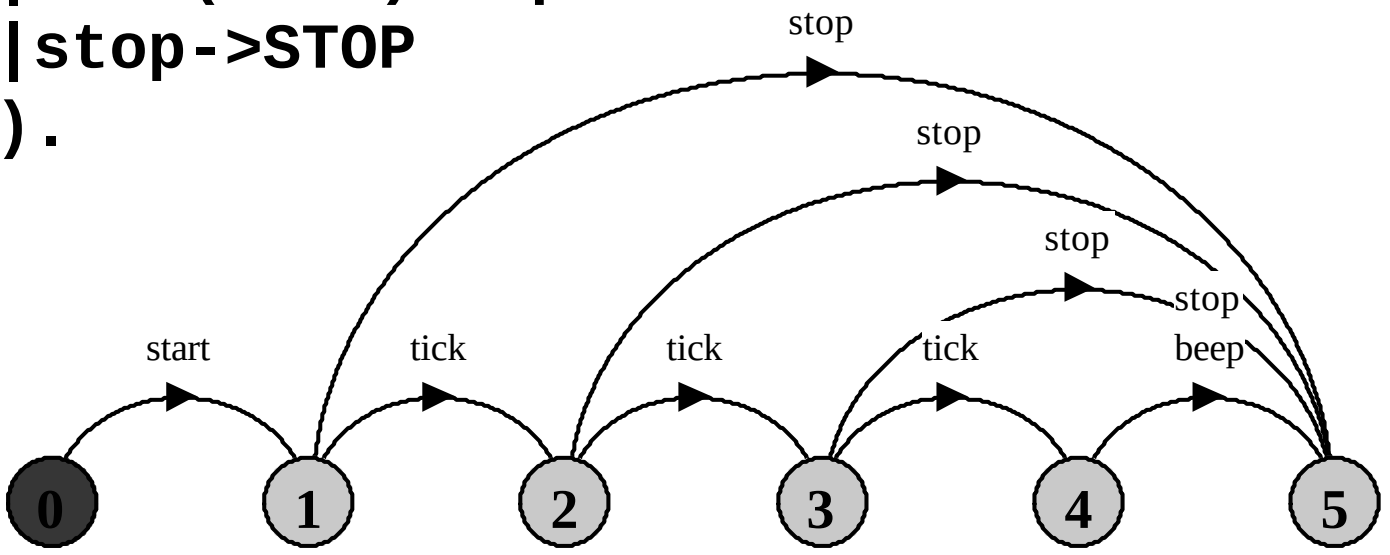
```
COUNT (N=3)      = COUNT[0],
COUNT[i:0..N] = (when(i<N) inc->COUNT[i+1]
                |when(i>0) dec->COUNT[i-1]
                ).
```



FSP - Bedingte Aktionen

Ein Countdown Zähler bewirkt “beep” nach N mal “tick”, falls er nicht vorzeitig gestoppt wird.

```
COUNTDOWN (N=3)   = (start->COUNTDOWN[N]),
COUNTDOWN[i:0..N] =
        (when(i>0) tick->COUNTDOWN[i-1]
        |when(i==0)beep->STOP
        |stop->STOP
        ).
```



FSP - Bedingte Aktionen

```
const False = 0
P = (when (False) tudochendlichwas->P).
```

Wie lautet ein einfacherer, äquivalenter Prozeß?

Antwort:

STOP

FSP - Das Alphabet eines Prozesses

Das Alphabet eines Prozesses ist die Menge der Labels/Transitionsbezeichner, die er verwendet.

Das **implizit** angegebene Alphabet eines Prozesses kann durch den Erweiterungsoperator erweitert werden :

```
WRITER = (write[1]->write[3]->WRITER)
        +{write[0..3]}.
```

Das Alphabet von **WRITER** ist **{write[0..3]}**

(Alphabeterweiterungen sinnvoll in Verbindung mit Komposition)

(Ergänzung)

Finite State Processes (FSP)

- Notation zur Beschreibung einzelner Prozesse mittels
  - Action Prefix: erlaubt Transitionssequenzen (Befehlssequenzen)
  - Choice, Guarded Action: erlauben nicht-deterministische Auswahl und Fallunterscheidungen (If-then-else)
  - Rekursion: erlaubt Schleifen
  - Alphabeterweiterung: Erweiterung des Alphabets wg Komposition
- Notation zur Komposition von Prozessen
  - Parallel Composition: Prozesse nebeneinander ausführbar, Synchronisation über gleiche Transitionsbezeichner
  - Relabeling
  - Hiding, Interface
  - Hilfen zur Beschreibung gleichartiger Prozesse nebst Kommunikation
    - forall Replicator: Vervielfachung von gleichartigen Prozessen
    - Process Labeling: Prefix Identifier für einzelnen Prozeß
    - Process Sharing: Auswahl Identifier für potentielle Interaktion
    - Priority high/low: Bevorzugen/Benachteiligen von Labels für Interaktion

Parallele Komposition - Interleaving von Aktionen

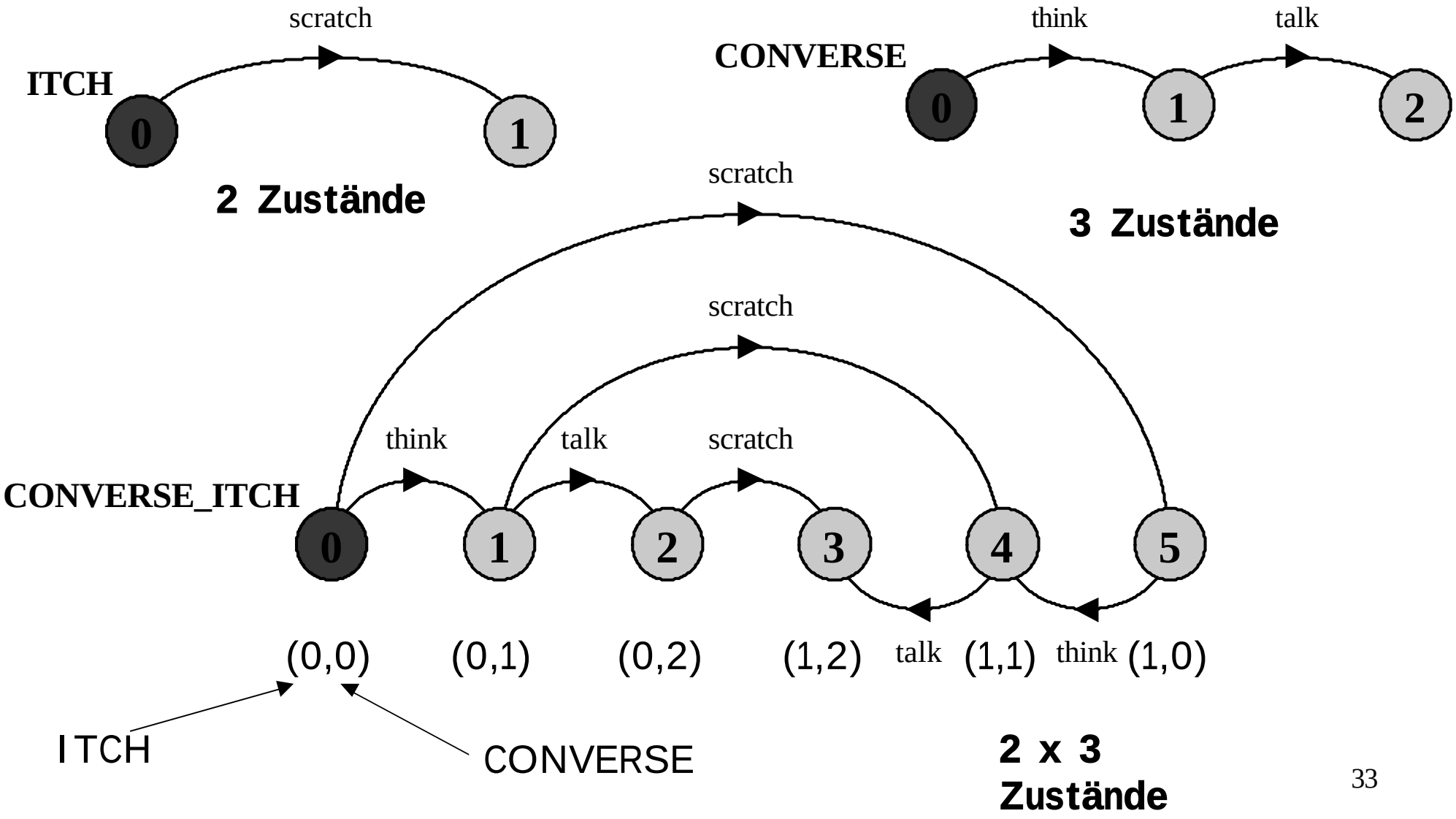
Falls P und Q Prozesse, dann beschreibt (P||Q) die nebenläufige Ausführung von P und Q. Operator || steht für parallele Komposition und Synchronisation über gleiche Transitionsbezeichner.

```
ITCH  = (scratch->STOP).
CONVERSE = (think->talk->STOP).

||CONVERSE_ITCH = (ITCH || CONVERSE).
```

think→talk→scratch
think→scratch→talk
scratch→think→talk

Mögliche Abläufe als Folge des Interleavings.

Parallele Komposition - Interleaving von Aktionen



Parallel Komposition - Algebraische Eigenschaften

Kommutativ: `(P||Q) = (Q||P)`

Assoziativ: `(P||(Q||R)) = ((P||Q)||R)`
`= (P||Q||R).`

Radioweckerbeispiel:

```
CLOCK = (tick->CLOCK).
RADIO = (on->off->RADIO).

||CLOCK_RADIO = (CLOCK || RADIO).
```

LTS? Abläufe? Anzahl Zustände?

Interaktion zwischen Prozessen - geteilte Aktionen

Bei der Komposition von Prozessen werden Aktionen mit gleichen Transitionsbezeichnern synchronisiert. Synchronisierte Transitionen können in allen beteiligten Prozessen nur gemeinsam auftreten (geteilte Aktionen).

Kommunikation durch Synchronisation.

```
MAKER = (make->ready->MAKER).
USER  = (ready->use->USER).

||MAKER_USER = (MAKER || USER).
```

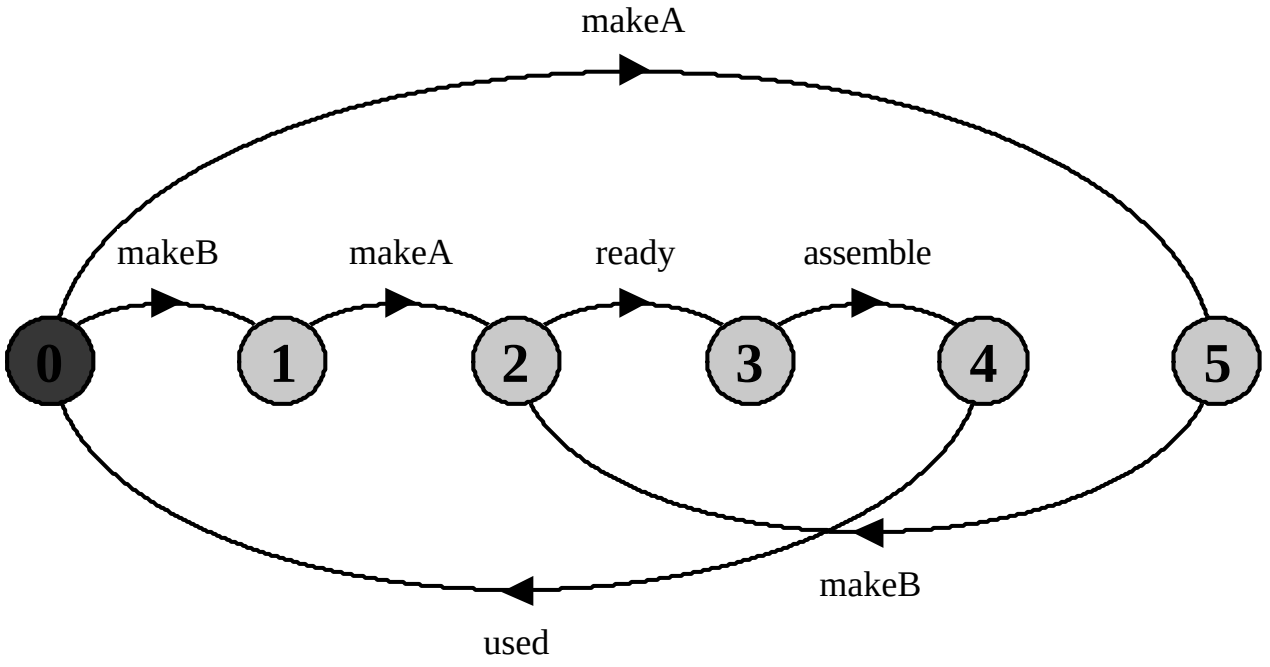
MAKER synchronisiert mit **USER** über Aktion **ready**.

LTS? Abläufe? Anzahl Zustände?

Interaktion zwischen mehreren Prozessen

Multi-party Synchronisation:

```
MAKE_A   = (makeA->ready->used->MAKE_A).
MAKE_B   = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).

||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).
```

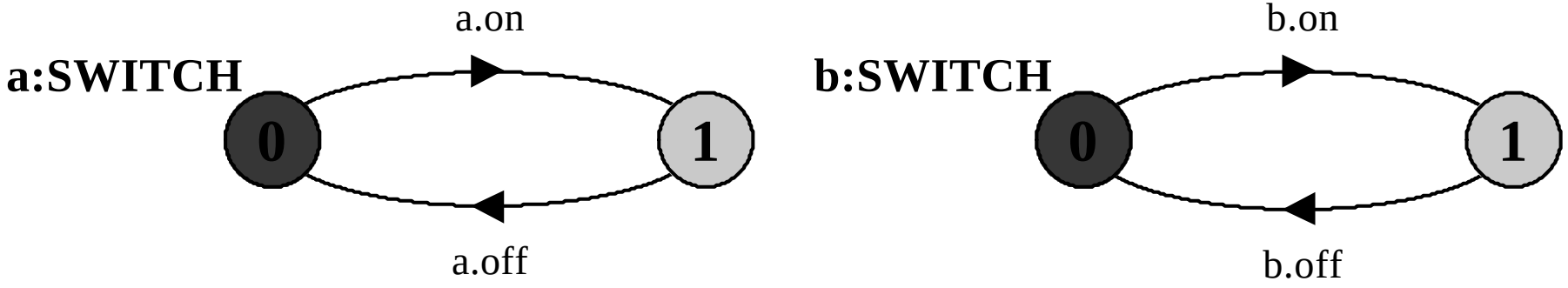


Beschriftung (Labeling) von Prozessen

a:P erweitert jedes Label aus Alphabet von P um Prefix a.

Zwei **Instanzen** eines Schalterprozesses:

```
SWITCH =  (on->off->SWITCH).
||TWO_SWITCH = (a:SWITCH || b:SWITCH).
```



Ein Feld von **Instanzen** aus Schalterprozessen:

```
||SWITCHES(N=3) = (forall[i:1..N] s[i]:SWITCH).
||SWITCHES(N=3) = (s[i:1..N]:SWITCH).
```

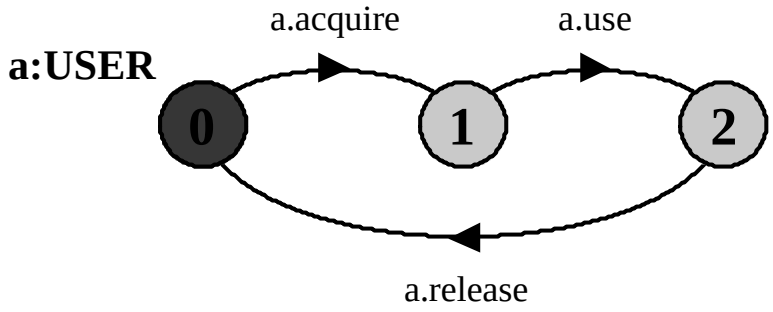
Erweiterung auf Bezeichnermengen

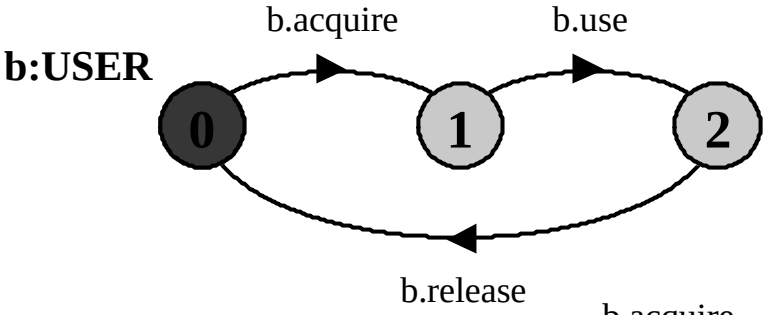
{a1,..,ax}::P ersetzen jeden Transitionsbezeichner n des Alphabets von P durch a1.n,…,ax.n. Ferner, jede Transition (n->X) aus P wird durch Transitionen ({a1.n,…,ax.n} ->X) ersetzt.

Prozeß Prefixing sinnvoll für **geteilte** Ressourcen:

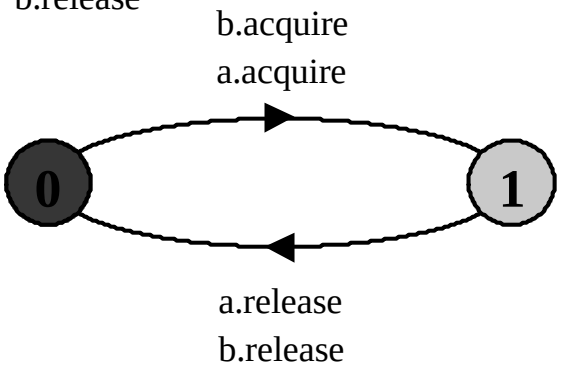
```
RESOURCE = (acquire->release->RESOURCE).
USER = (acquire->use->release->USER).

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

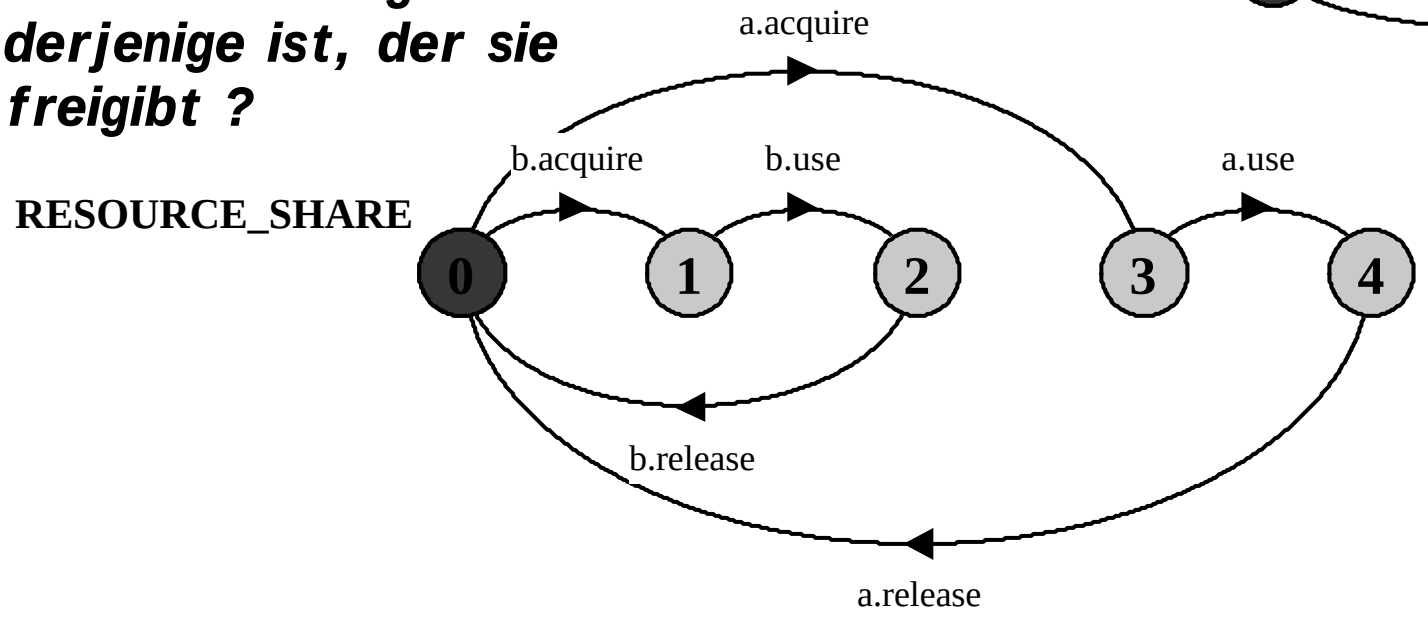
Beispiel für Prozeß Prefixing bei geteilten Ressourcen





Wodurch ist sichergestellt, daß der Benutzer, der die Ressource belegt auch derjenige ist, der sie freigibt ?





Umbenennung von Aktionen: Action Relabeling

Relabeling Funktionen beschreiben die Umbenennung von Transitionsbezeichnern durch eine 1-1 Ersetzung. In der Form:

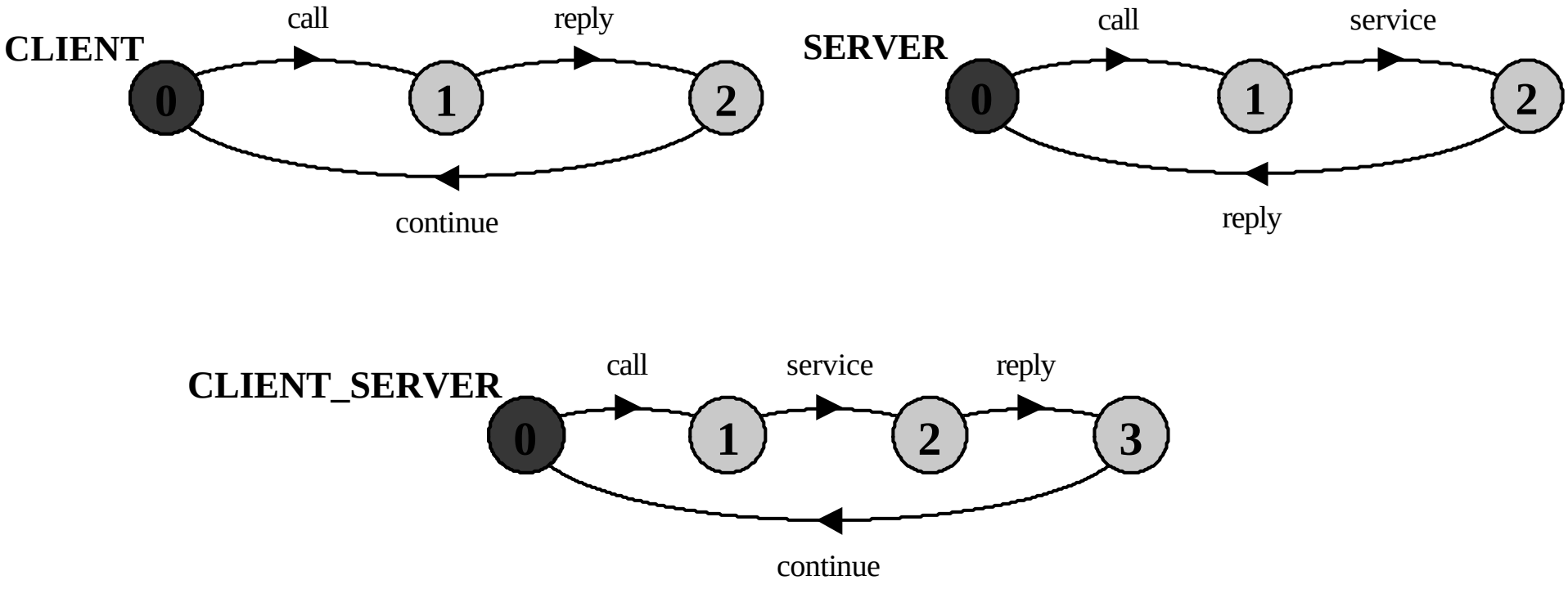
/{*newlabel*_1/*oldlabel*_1,... *newlabel*_n/*oldlabel*_n}.

Mit Relabeling kann man Synchronisation bei der Komposition von Prozessen herstellen.

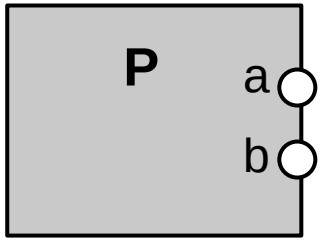
```
CLIENT = (call->wait->continue->CLIENT).
SERVER = (request->service->reply->SERVER).
```

Action Relabeling

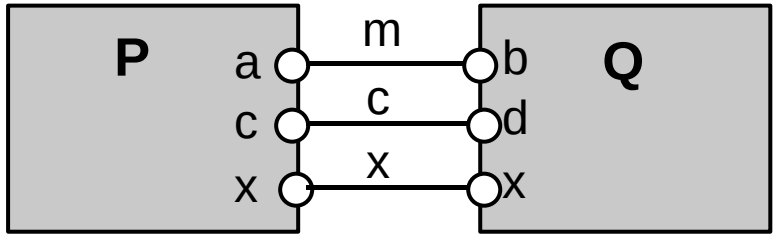
```
||CLIENT_SERVER = (CLIENT || SERVER)
                 /{call/request, reply/wait}.
```



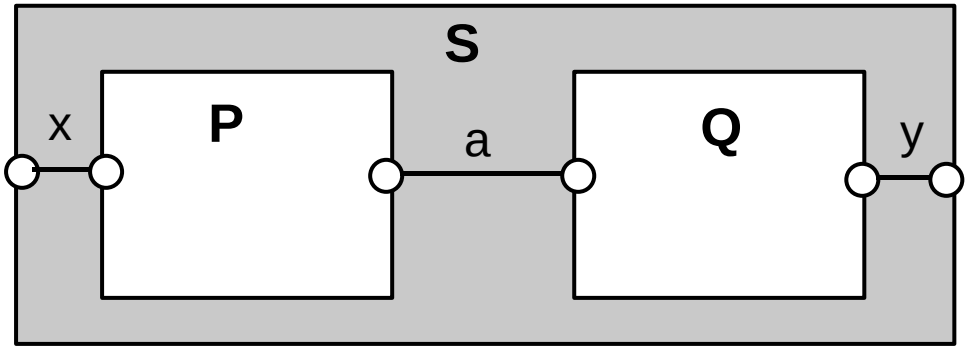
Visualisierung großer Modelle: Strukturdiagramm



Prozeß P mit
Alphabet {a,b}.

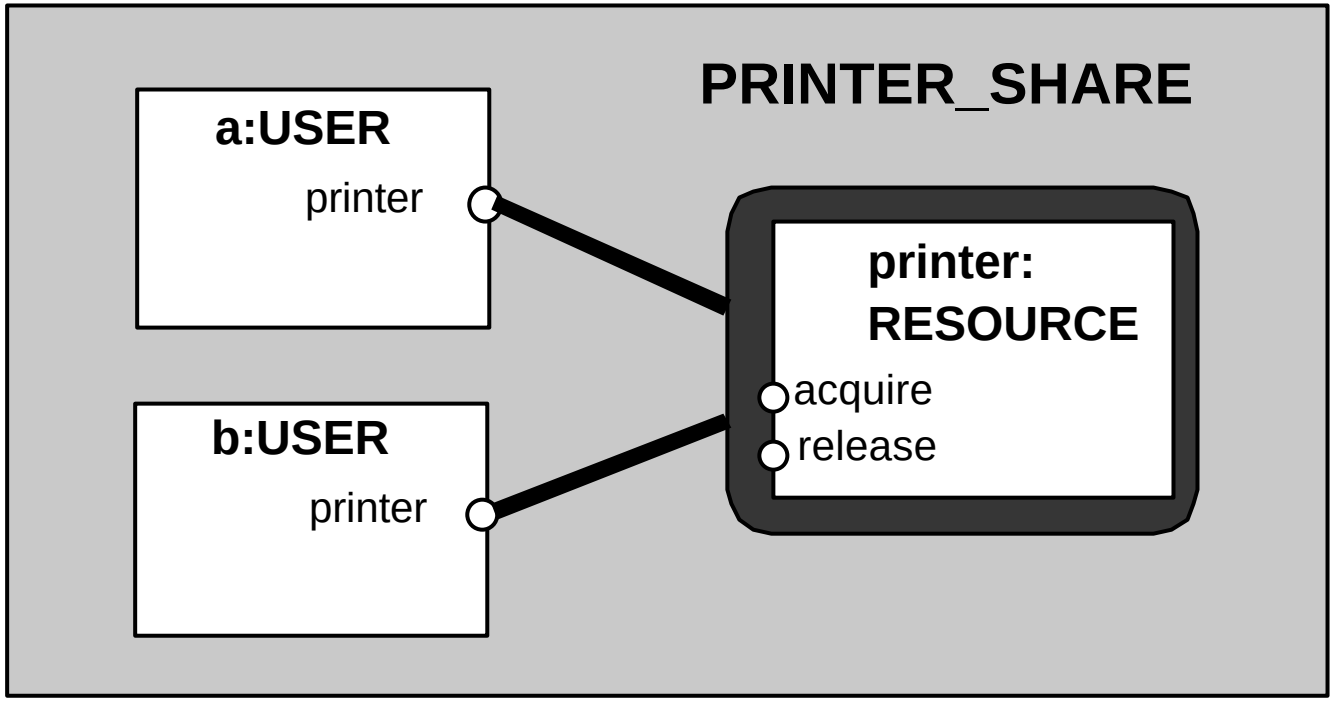


Parallele Komposition
(P||Q) / {m/a,m/b,c/d}



Prozeß aus Teilprozessen
||S = (P||Q) @ {x,y}

Strukturdiagramm - Beispiel geteilte Ressourcen




```
RESOURCE = (acquire->release->RESOURCE).
USER =     (printer.acquire->use
            ->printer.release->USER).

||PRINTER_SHARE
  = (a:USER||b:USER||{a,b}::printer:RESOURCE).
```

Zusammenfassung

- Konzepte in Prozeßalgebren
  - **Prozeß - Grundlage für Nebenläufigkeit (in Java: Thread)**
  - **Nebenläufigkeit - Asynchrone Abarbeitung & Interleaving**
  - **Kommunikation über geteilte Aktionen (gleiche Labels)**
- Modellwelten, Notationen
  - **Mealy-Automat Schwerpunkt auf I/O Relation**
  - **LTS zur Modellierung von Prozessen als Automaten - Sequenzen atomarer Aktionen**
  - **FSP zur Modellierung von Prozessen mit Prefix "->", Choice " | ", Rekursion, Komposition, Relabeling, Hiding.**
  - **Strukturdiagramme zur Darstellung von komplexer Modelle.**