

# Semantiken für Statecharts

Benjamin Braatz, Markus Klein

GraGra-/PN-AG, 22.5.2003

## Inhalt

- Klassische Statecharts:
  - Zustände und Konfigurationen
  - Konfigurations-Transformationen
  - Ereignisse und Aktionen
  - Schritte und Zeitmodelle
- Objekt-Orientierte Statecharts:
  - Syntax für Objekt-Orientierte Statecharts
  - Semantik mit Abstract State Machines
  - Semantik mit Graph-Transformation

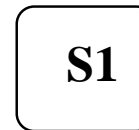
## Zustände und Konfigurationen

- Einfache Zustände:

**Aktiv:**

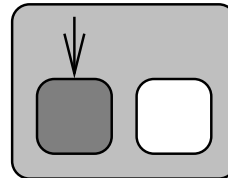


**Inaktiv:**

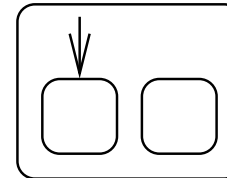


- OR-Zustände:

**Aktiv:**

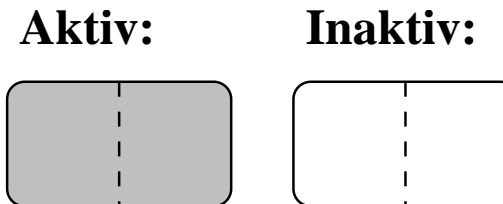


**Inaktiv:**



Genau ein Unterzustand eines aktiven OR-Zustands ist aktiv.  
Wenn ein OR-Zustand neu aktiv wird, wird der initiale Unterzustand aktiv.

- **AND-Zustände:**

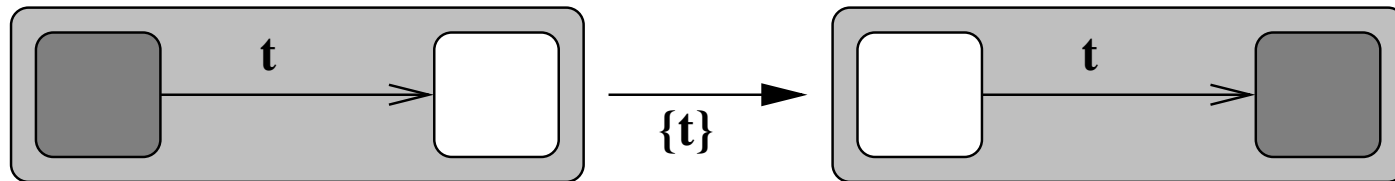


Alle Unterzustände eines aktiven AND-Zustands sind aktiv.

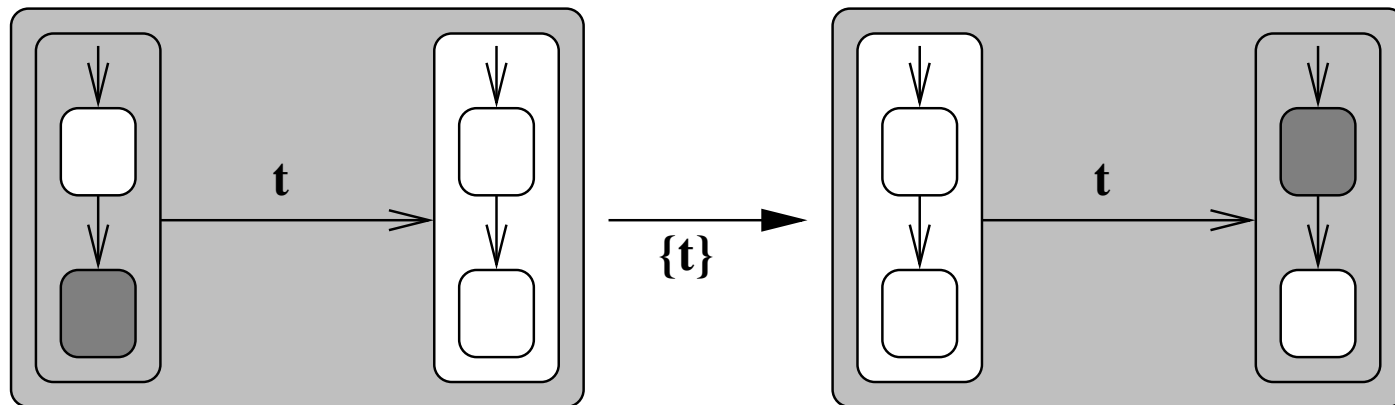
- **Konfigurationen:** Alle legalen Mengen aktiver Zustände inklusive des **Root-Zustandes**
- **History-Konnektoren:** Zuletzt aktiver statt initialer Zustand  
Shallow: Zuletzt aktiver Zustand wird initialisiert  
Deep: Der zuletzt aktive Zustand wird rekursiv aktiviert

## Konfigurations-Transformationen

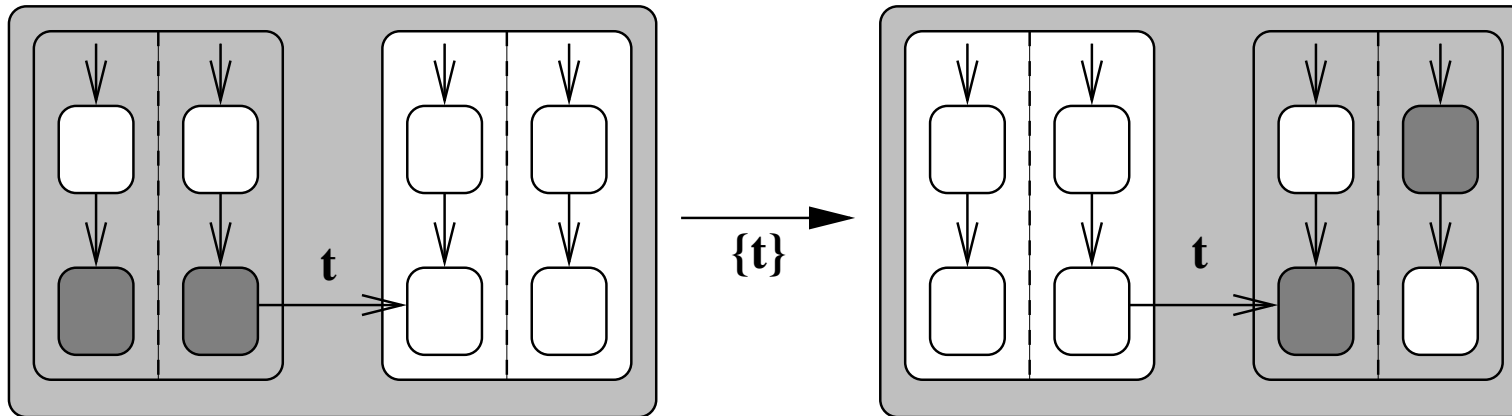
- Einzelne Transitionen zwischen einfachen Zuständen:



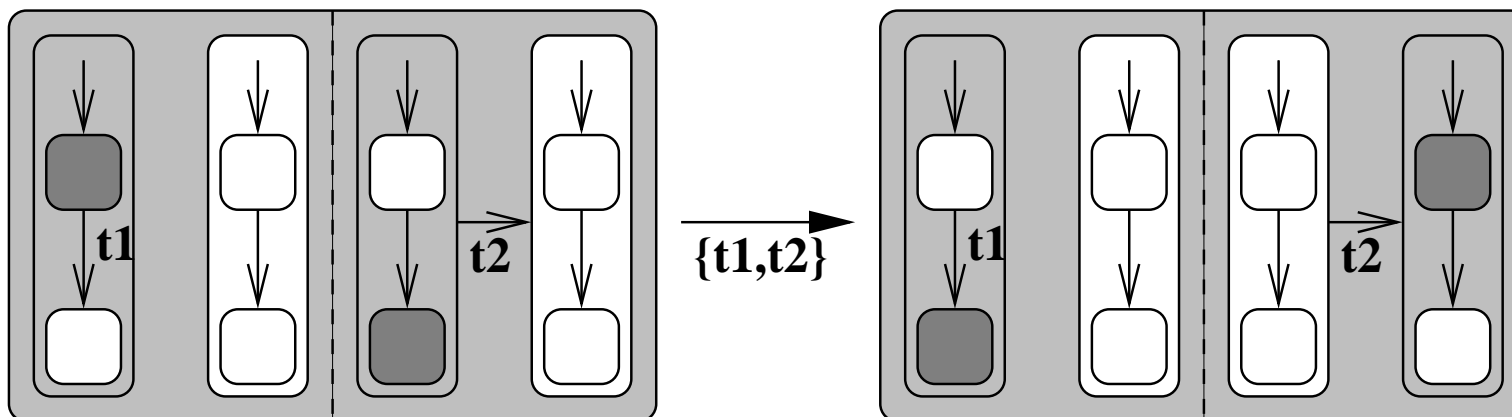
- Einzelne Transitionen zwischen OR-Zuständen:



- Transitionen über Zustandsgrenzen hinweg:



- Mehrere Transitionen:



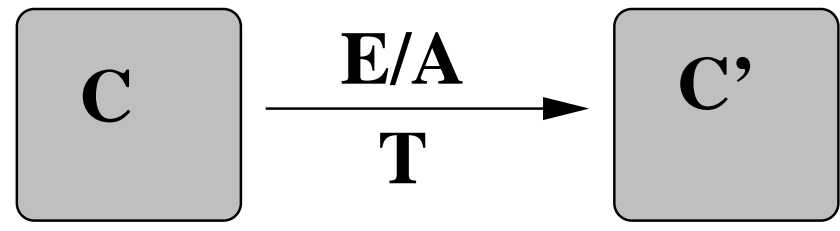
- **Relevanz einer Transitionsmenge für eine Konfiguration:**  
Alle Quell-Zustände in der Konfiguration enthalten
- **Konsistenz einer Transitionsmenge:**  
Keine überlappenden Transitionen
- **Prioritäten bei zueinander inkonsistenten Transitionen:**
  - Klassisch: Transitionen mit höherem Scope haben Priorität
  - UML: Transitionen mit niedrigerem Quell-Zustand haben Priorität

## Ereignisse und Aktionen

- Gegeben: **Ereignisse**  $\Sigma$  und **negierte Ereignisse**  $\neg\Sigma$
- **Trigger und Aktionen für Transitionen:**  
 $Trigger(t) \subset \Sigma \cup \neg\Sigma$ ,  $Actions(t) \subset \Sigma$
- **Aktivierung von Transitionsmengen für Ereignismengen:**  
Selbst-Triggerung im nächsten Schritt [Harel/Naamad]:
  - Positive Trigger aller Transitionen in der Ereignismenge
  - Negative Trigger aller Transitionen nicht in der Ereignismenge
  - Aktionen erst im nächsten Schritt sichtbarSelbst-Triggerung im gleichen Schritt [Pnueli/Shalev]:
  - Ereignismenge plus Aktionen anderer Transitionen
  - Daher: Keine Negation einer Aktion im Trigger
  - Kausalitätsprinzip zwischen Transitionen

## Schritte und Zeit-Modelle

- Bestehen aus Konfigurations-Transformationen



wobei **T**

- für **C** relevant und konsistent ist,
- zu **C'** schaltet,
- für **E** aktiviert und maximal priorisiert ist,
- eine maximale Anzahl Transitionen enthält und
- die Aktionen **A** auslöst.

- **Schritte** schalten zwischen Konfigurationen mit **Eingabe** und **Ausgabe**

$$in \in (\mathcal{P}(\Sigma))^* \text{ and } out \in (\mathcal{P}(\Sigma))^*$$

- **Selbst-Triggerung im nächsten Schritt [Harel/Naamad]:**
  - **Asynchrone Zeit:**
  - Mikro-Schritte:**

$$(C, E.in, out) \longrightarrow_{\mu} (C', A.in, A.out)$$



**Super-Schritte:**

$$(C, E.in, out) \longrightarrow_{sup} (C', in, A.out) \text{ für}$$

$$(C, E.in, out) \longrightarrow_{\mu}^{+} (C', in, A_n \dots A_1.out) \text{ und } A = \bigcup_{i=1, \dots, n} A_i$$

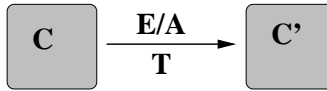
– **Synchrone Zeit:**

$$(C, E.F.in, out) \longrightarrow_{sync} (C', (F \cup A).in, A.out)$$

für 

• **Selbst-Triggerung im gleichen Schritt [Pnueli/Shalev]:**

$$(C, E.in, out) \longrightarrow_{PS} (C', in, A.out)$$

für 

• **Kompositionale Semantik nach [Lüttgen/von der Beeck/Cleveland]:**

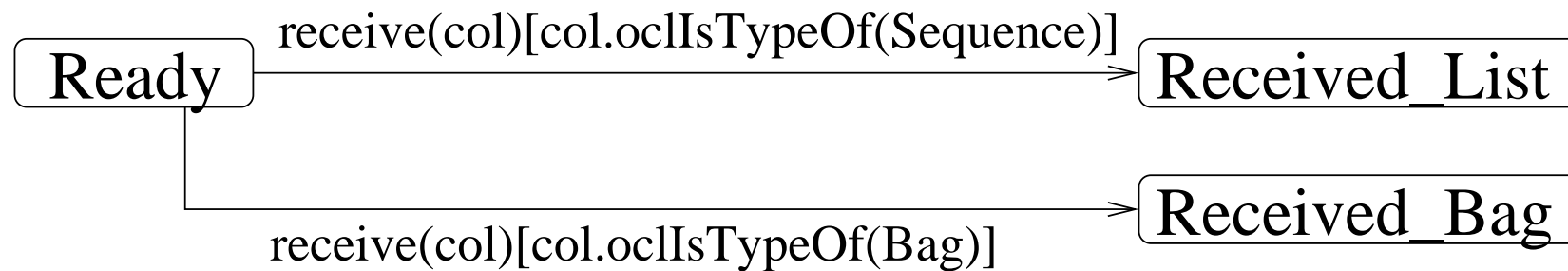
Information über Zusammensetzung von Schritten aus einzelnen Transitionen in **Mikro-Konfigurationen**

## Syntax für OO-Statecharts

- Guards und Finale Zustände



- Beispiel



## Abstract State Machines

<b>Update</b>	$f(\bar{s}) := t$
<b>Conditional</b>	if $g$ then $R$ else $S$
<b>Blocks</b>	<b>do-in-parallel</b> $R_1, \dots, R_k$ <b>enddo</b> (Regeln müssen konsistent sein)
<b>Sequential Composition</b>	<b>seq</b> $R, S$ <b>endseq</b>
<b>Quantor</b>	<b>forall</b> $v$ <b>with</b> $g(v)$ <b>do</b> $R(v)$ (Konsistenz)
<b>Loop List</b>	<b>loop</b> $v$ <b>through list</b> $X$ $R(v)$

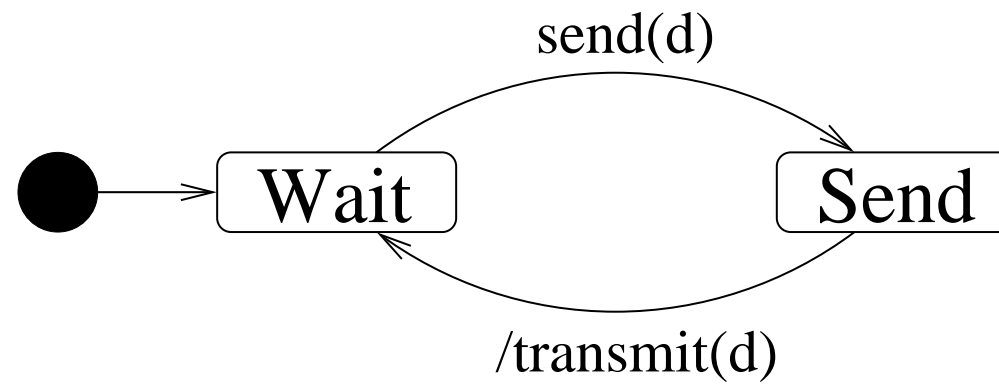
## Definition: Interaktive ASM

1. Die Menge der Nachrichtennamen,  $\mathbf{MsgNm}$ , enthält endliche Sequenzen  $n_1.n_2.\dots.n_k$ , wobei  $n_1$  bis  $n_{k-2}$  Namen von Subsystemen sind.  $n_{k-1}$  ist der Name eines Objekts und  $n_k$  ist der lokale Name der Nachricht. (Operationen, Signale, Return-Nachrichten)
2. Eine *interaktive ASM*  $iA = (A, in, out)$  besteht aus einer ASM  $A$ , und zwei Mengen von Multimengennamen  $in, out$ , wobei die Regeln in  $A$  nur Elemente zu  $out$  hinzufügen dürfen – es sei denn,  $elem \in out$  und  $elem \in in$ .

## Actions

<b>Call Action</b>	<b>ActionRule</b> ( $call_{op[args]}$ ) $outQueue(O) := outQueue(O) \uplus \{op_O[args]\}$
<b>Send Action</b>	<b>ActionRule</b> ( $send_e$ ) $outQueue(O) := outQueue(O) \uplus \{e\}$
<b>ReturnAction</b>	<b>ActionRule</b> ( $send_{return_{op(a)}}$ ) $outQueue(O) := outQueue(O) \uplus$ $\{trigsusy(op).return_{op}(a)\}$
<b>Assignment</b>	$att := exp$
<b>Void Action</b>	ASM Regel <b>skip</b>

## Beispiel: Sender

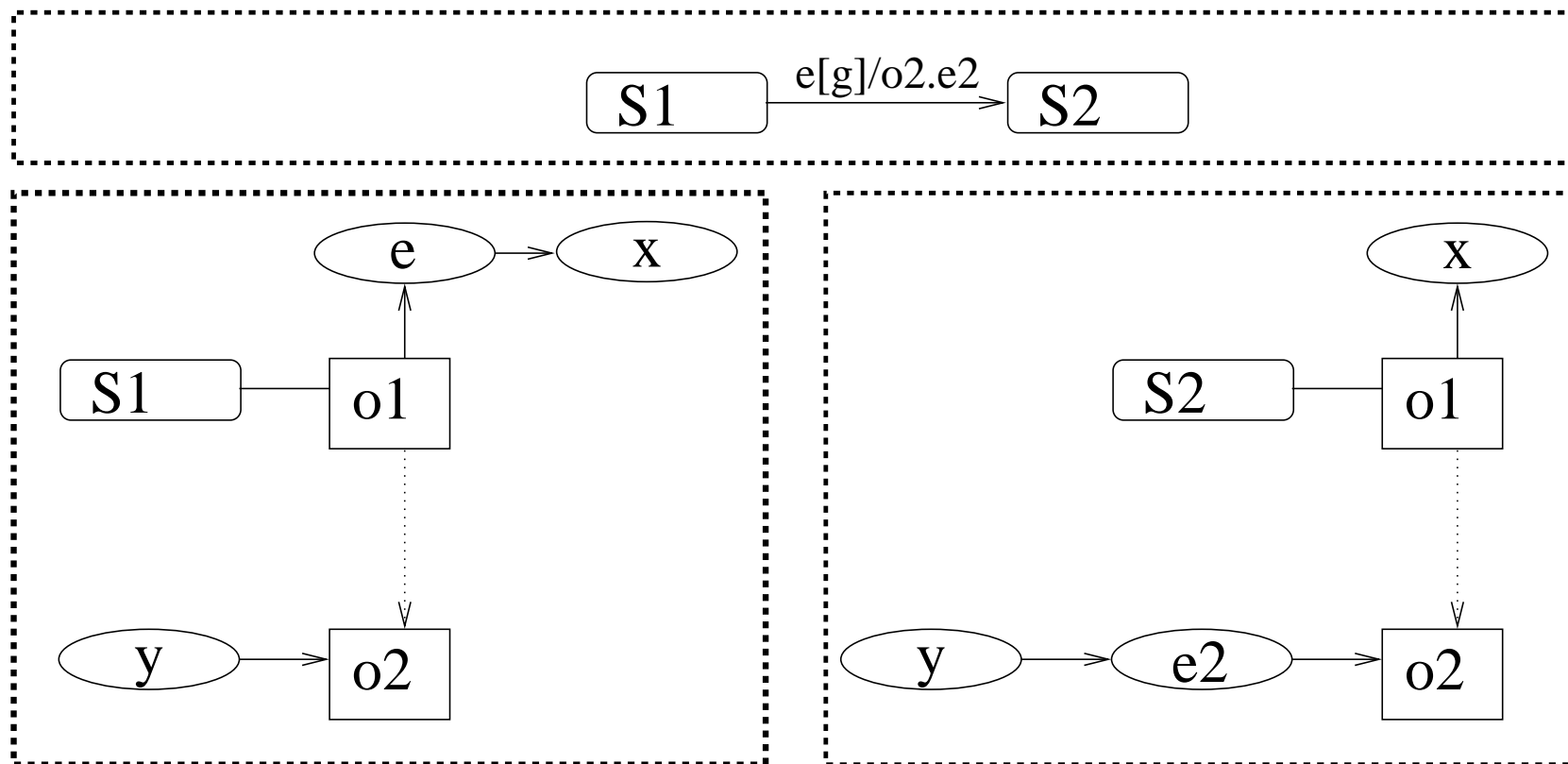


```

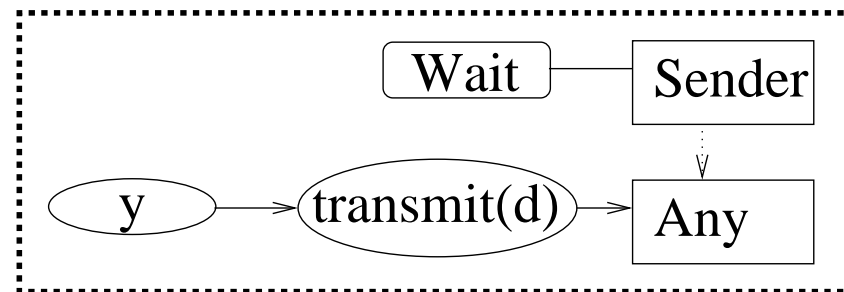
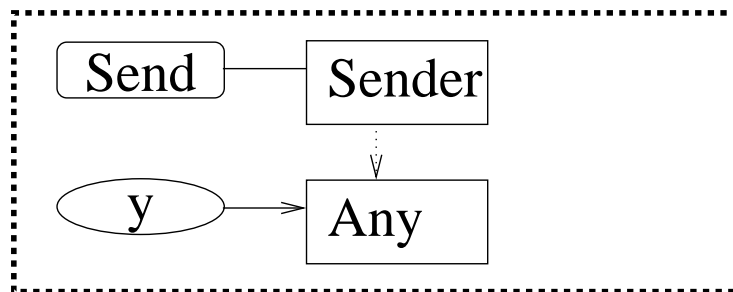
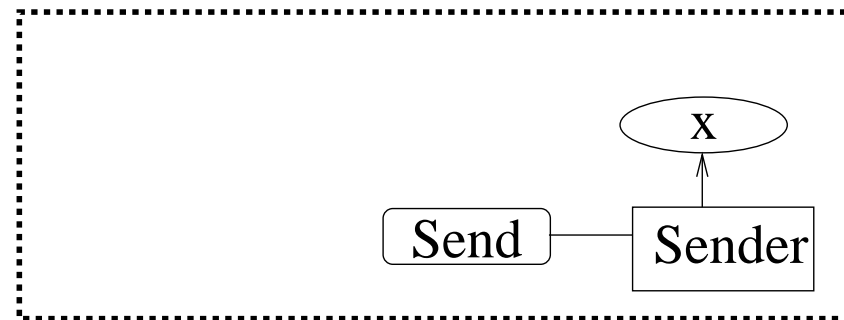
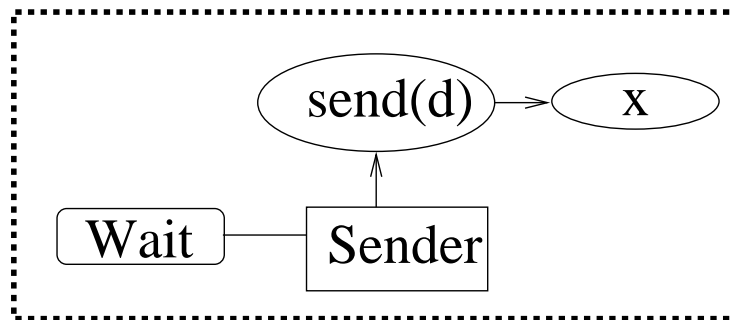
case currState of
  {Initialsender}: do currState:={Wait}
  {Wait}: do choose  $e : e \in inQueue(sender)$ 
    do-in-parallel
       $inQueue(sender) = inQueue(sender) \setminus \{e\}$ 
      if  $mname(e) = send$  then
        do-in-parallel
          currState:={Send}
           $d := \mathbf{Args}(e)$ 
        enddo
      enddo
  {Send}: do-in-parallel
    currState:={Wait}
     $outQueue(sender) := outQueue(sender) \uplus \{transmit(d)\}$ 
  enddo

```

# Graph-Transformations basierte Semantik



## Regeln für Beispiel Sender



## Übersicht

ASMs

- keine Transitionen über Grenzen von zusammengesetzten Zuständen
- synchrone und asynchrone Kommunikation

---

GraTra

- Zustände sind nur Namen
- asynchrone Kommunikation

## Literatur

- [**HN96**] David Harel, Amnon Naamad: The STATEMATE Semantics of Statecharts, ACM Transactions on Software Engineering and Methodology, Vol. 5, No. 4, 1996
- [**Joh02**] Sebastian John: Steps for Statecharts, TU Berlin – Fak. IV Bericht Nr. 2003-04
- [**LBC00**] Gerald Lüttgen, Michael von der Beeck, Rance Cleaveland: A Compositional Approach to Statecharts Semantics, ICASE Report No. 2000-12

- [**KGK02**] Sabine Kuske, Martin Gogolla, Hans-Jörg Kreowski, Ralf Kollmann: An Integrated Semantics for UML Class, Object and State Diagrams Based on Graph Transformation, 3rd International Conference on Integrated Formal Methods (IFM 2002)
- [**Jür02**] Jan Jürjens: A UML Statecharts Semantics with Message-Passing, Symposium of Applied Computing (SAC 2002)