Kapitel 13: Prozessmodellierung und Workflow-Management

13.1 Prozessmodellierung

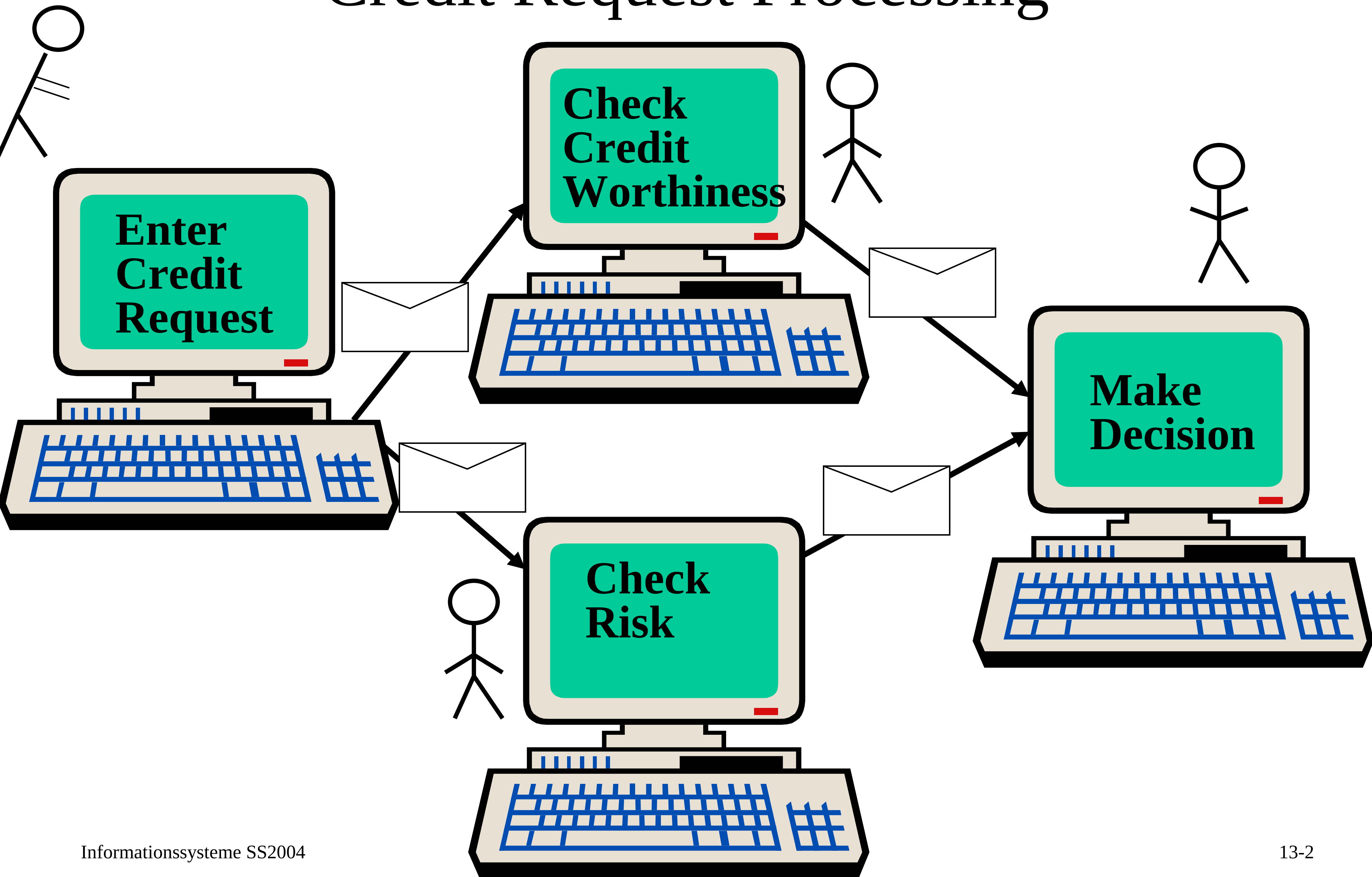
- Statecharts
- Ereignis-Prozess-Ketten

13.2 Workflow-Management für Geschäftsprozesse

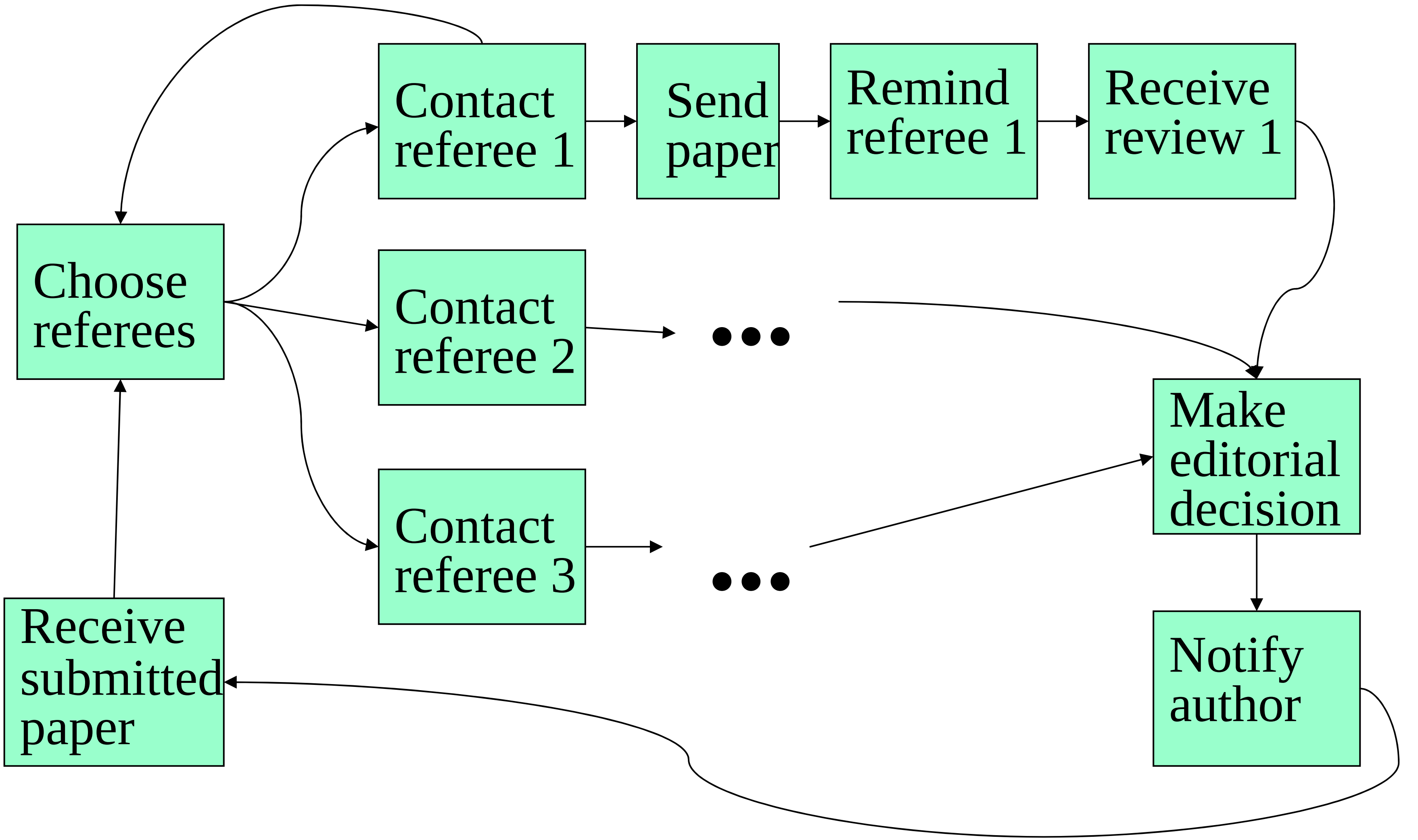
13.3 Semantik von Statecharts

13.4 Eigenschaften von Statecharts und deren Verifikation

Workflow Application Example 1: Credit Request Processing



Workflow Application Example 2: Journal Refereeing Process



What is Workflow Management?

Computer-supported business processes: coordination of control and data flow between distributed - automated or intellectual - activities

Application examples:

- Credit requests, insurance claims, etc.
- Tax declaration, real estate purchase, etc.
- Student exams, journal refereeing, etc.
- Electronic commerce, virtual enterprises, etc.

Business Benefits of Workflow Technology

- Business process automation
  (to the extent possible and reasonable)
  - ⇨ shorter turnaround time, less errors, higher customer satisfaction
  - ⇨ better use of intellectual resources for exceptional cases
- Transparency
  - ⇨ understanding & analyzing the enterprise
- Fast & easy adaptation
  - ⇨ Business Process Reengineering (BPR)

13.1 Specification Method and Environment

Requirements:

- Visualization
- Refinement & Composability
- Rigorous Semantics
- Interoperability with other methods & tools
- Wide acceptance & standard compliance

Solutions:

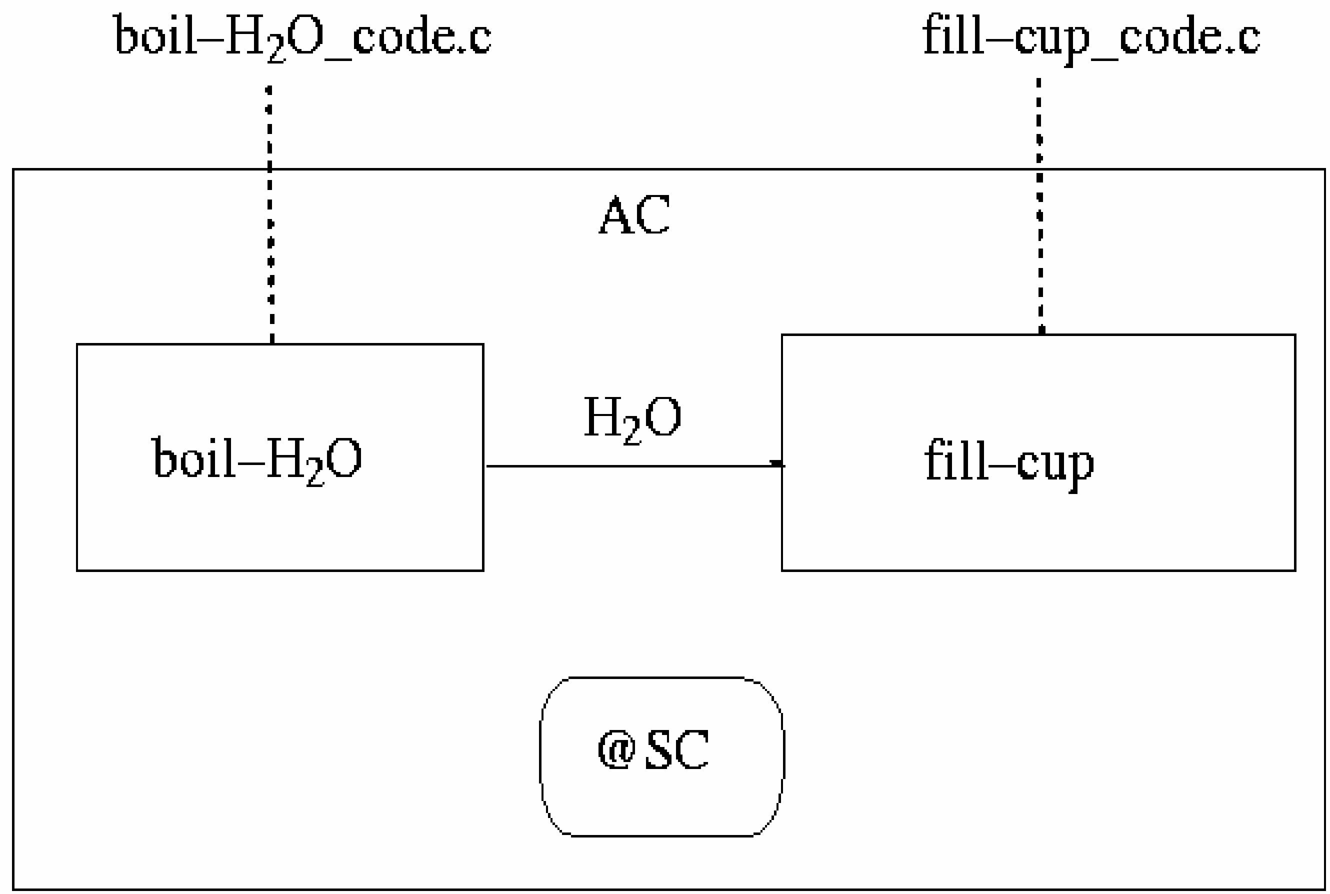
- ***Statecharts*** (Harel et al. 1987)

  (alt.: Petri Net variants, temporal logic, process algebra, script language)

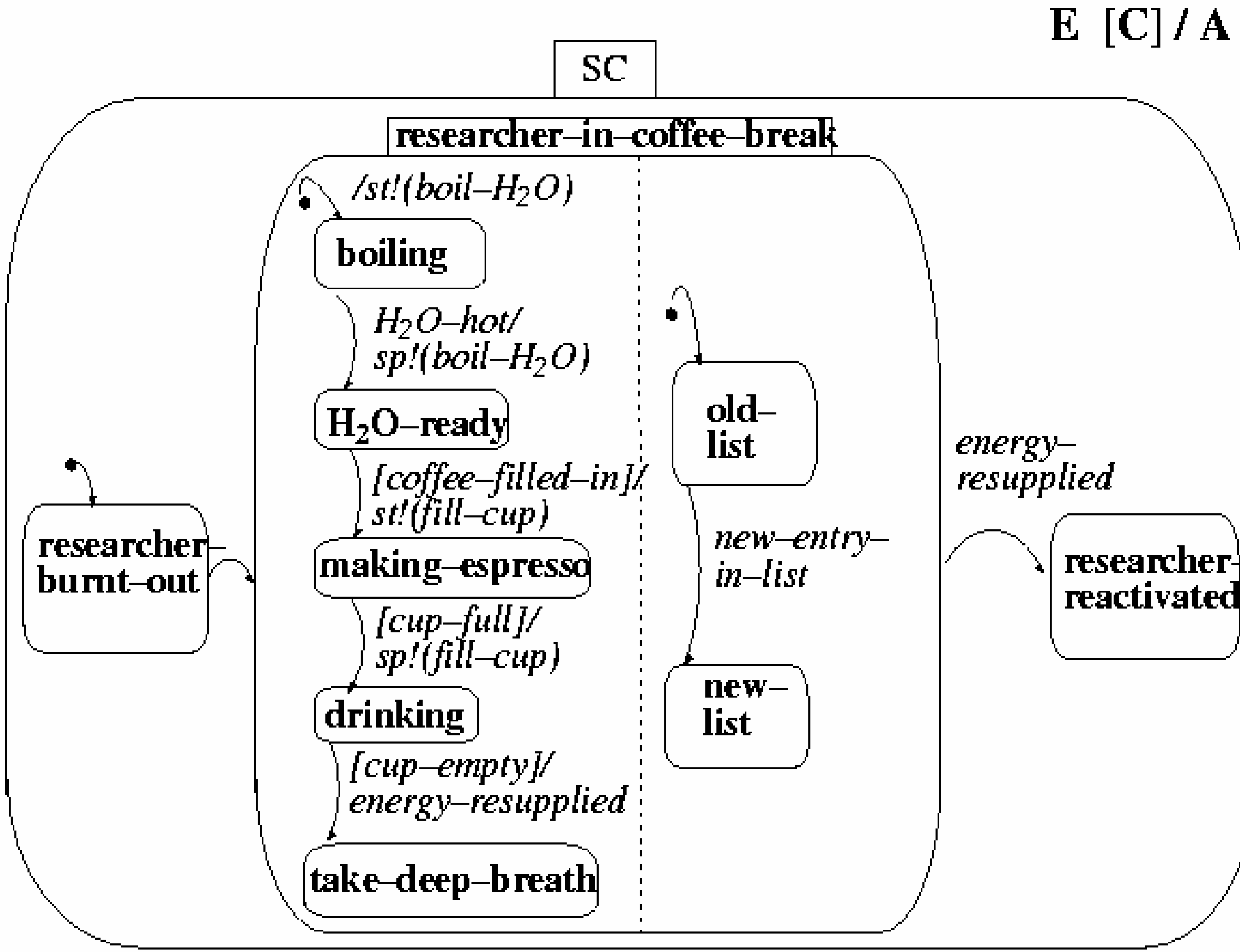
- Import / export
  BPR tools → WFMS ↔ WFMS

- Statecharts included in UML industry standard (Unified Modeling Language, OMG 1997))

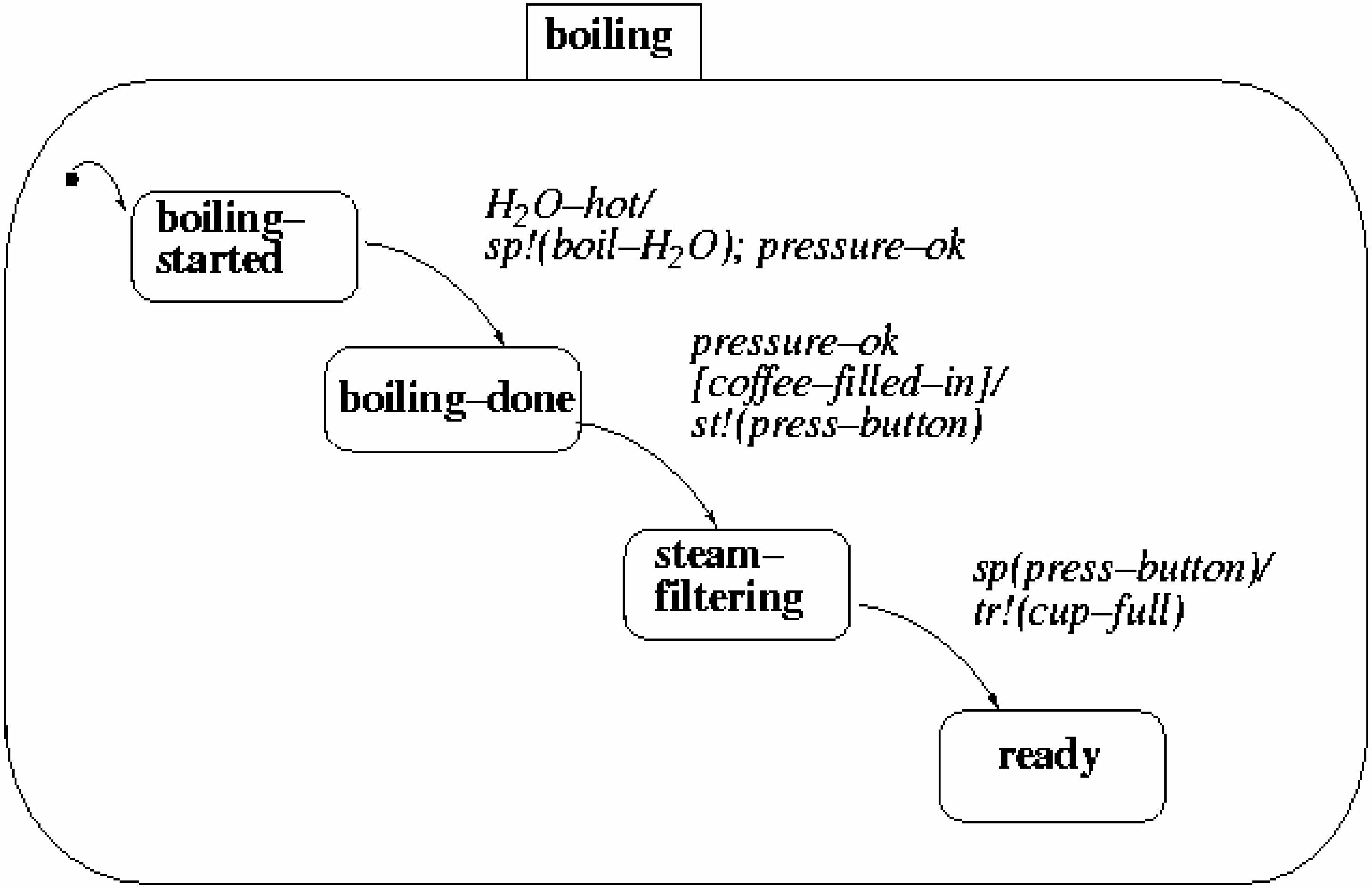
Example of Activitychart



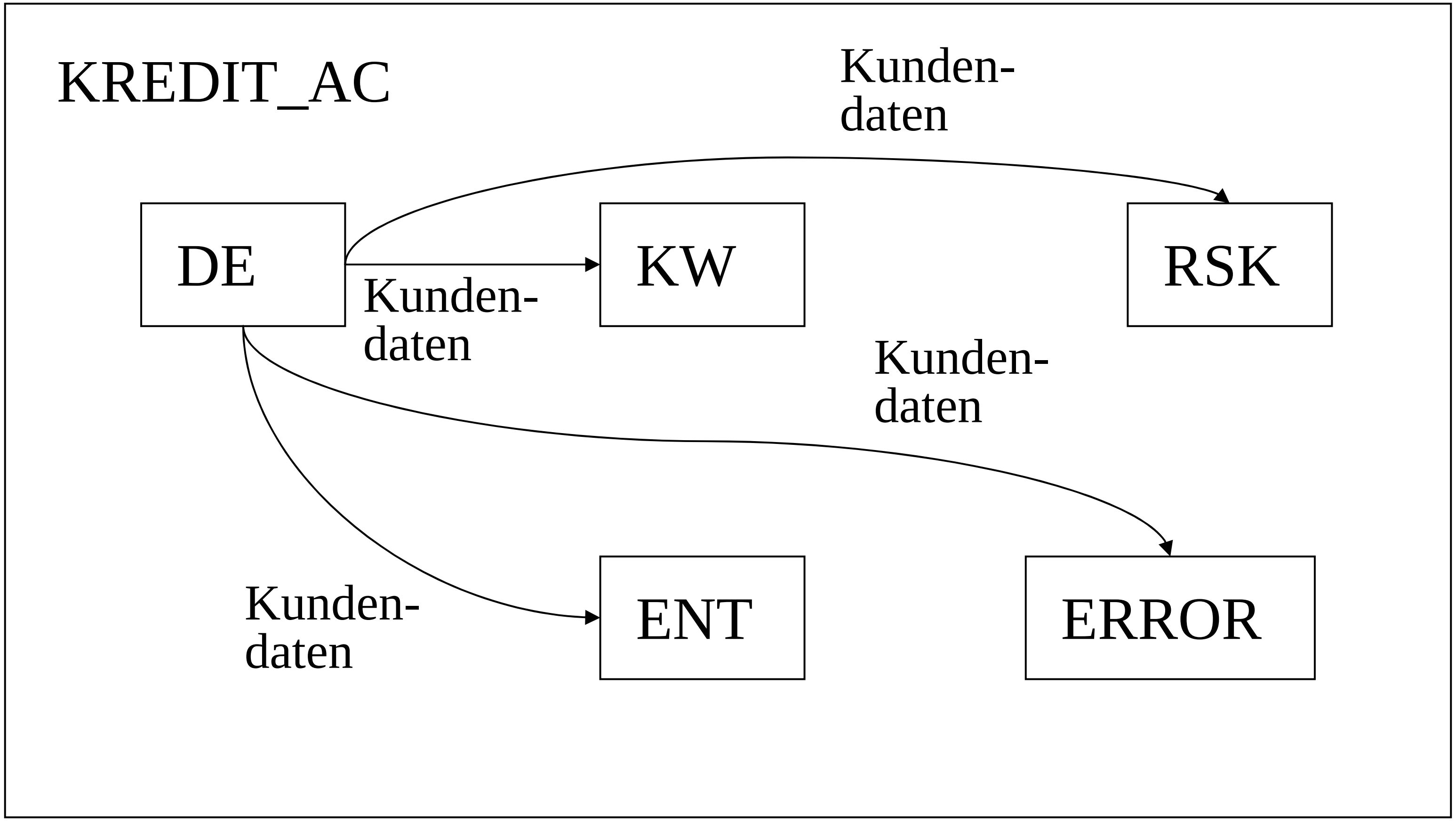
Example of Statechart



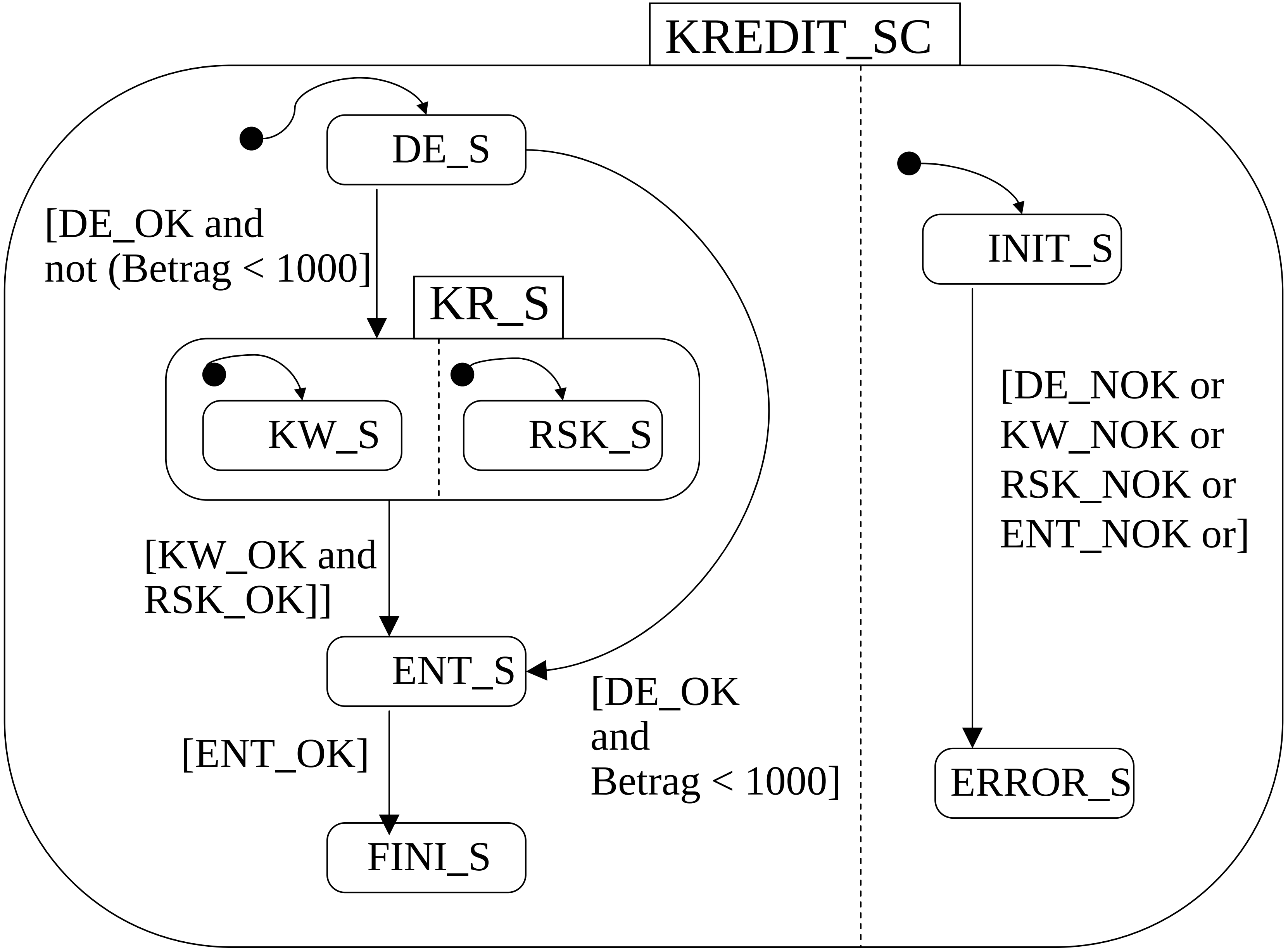
Refinement of States



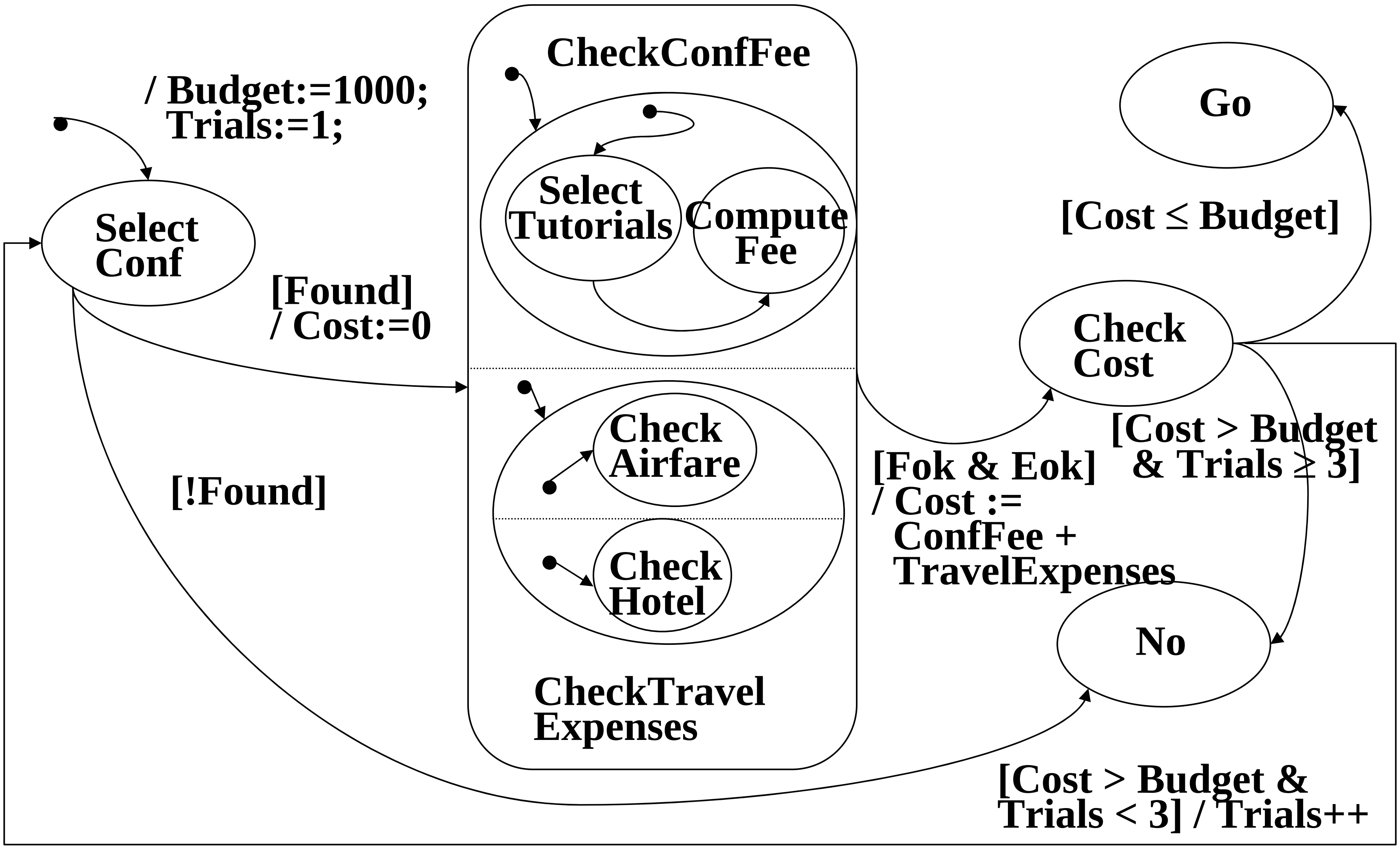
Activitychart Example 1



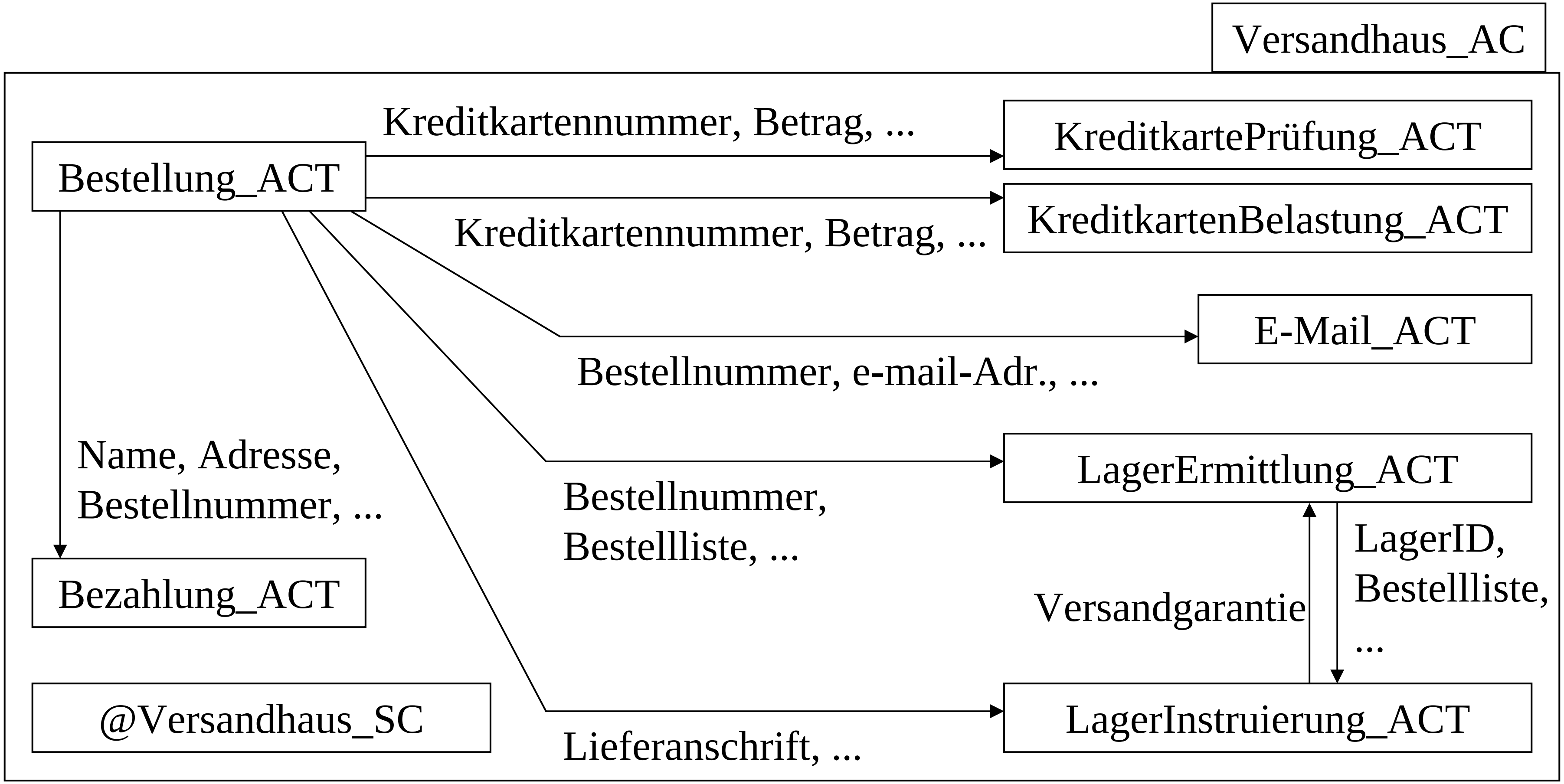
Statechart Example 1



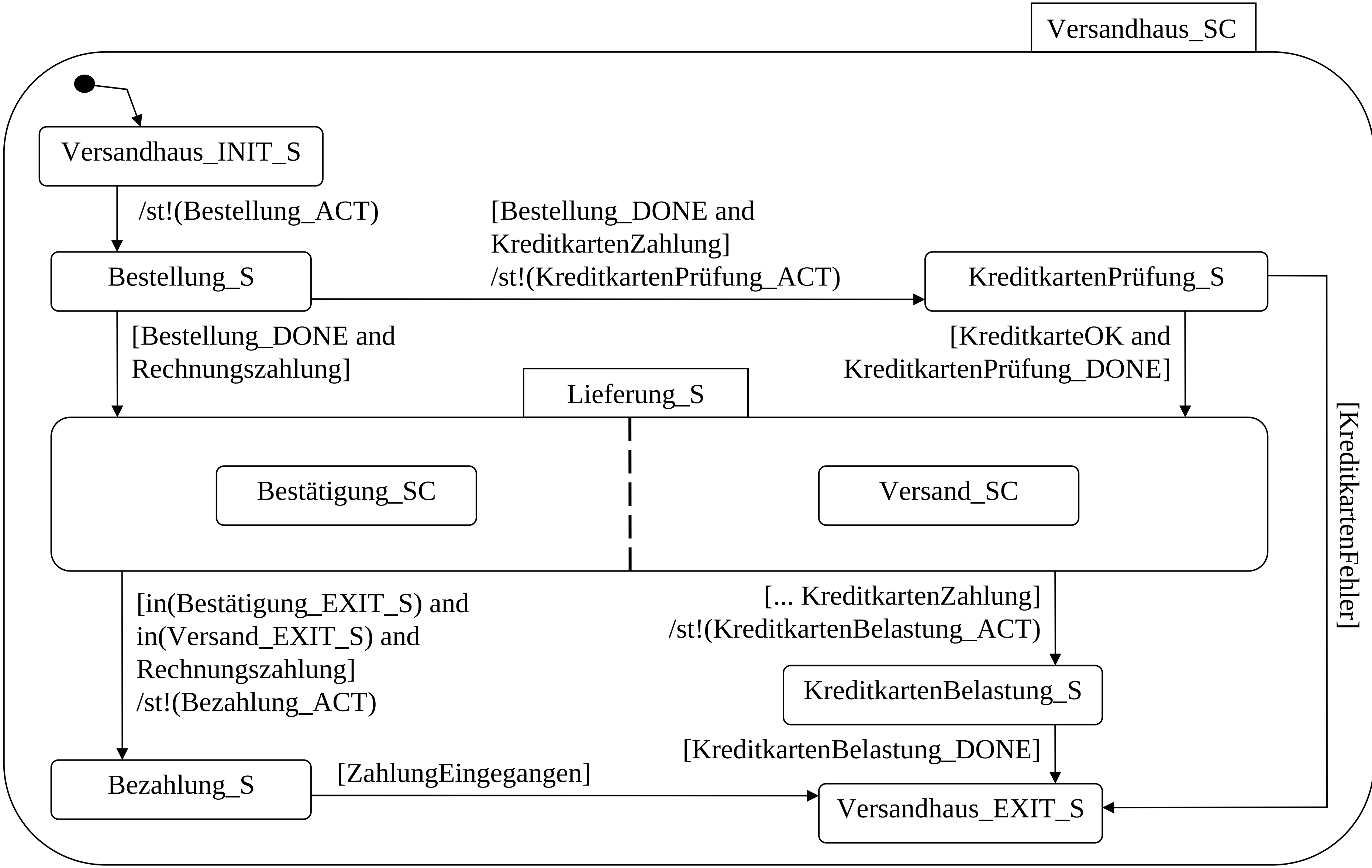
Statechart Example 2



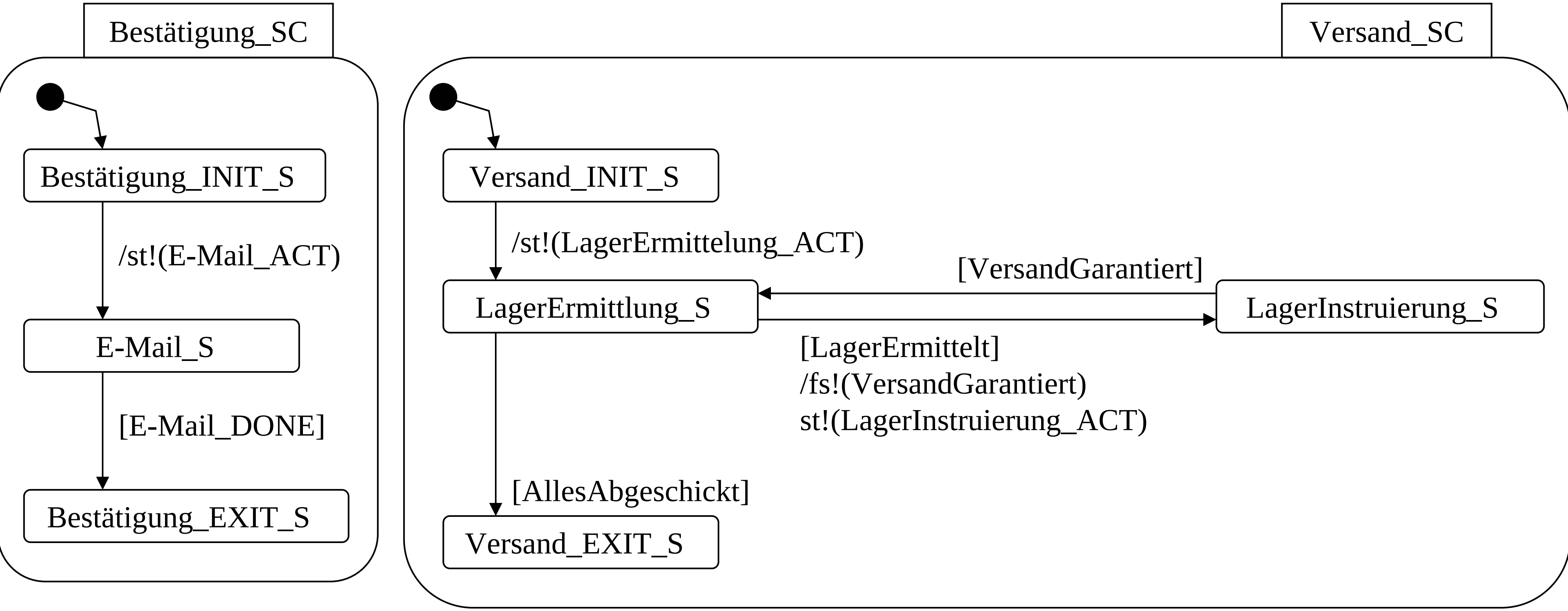
E-Commerce Workflow: Activitychart



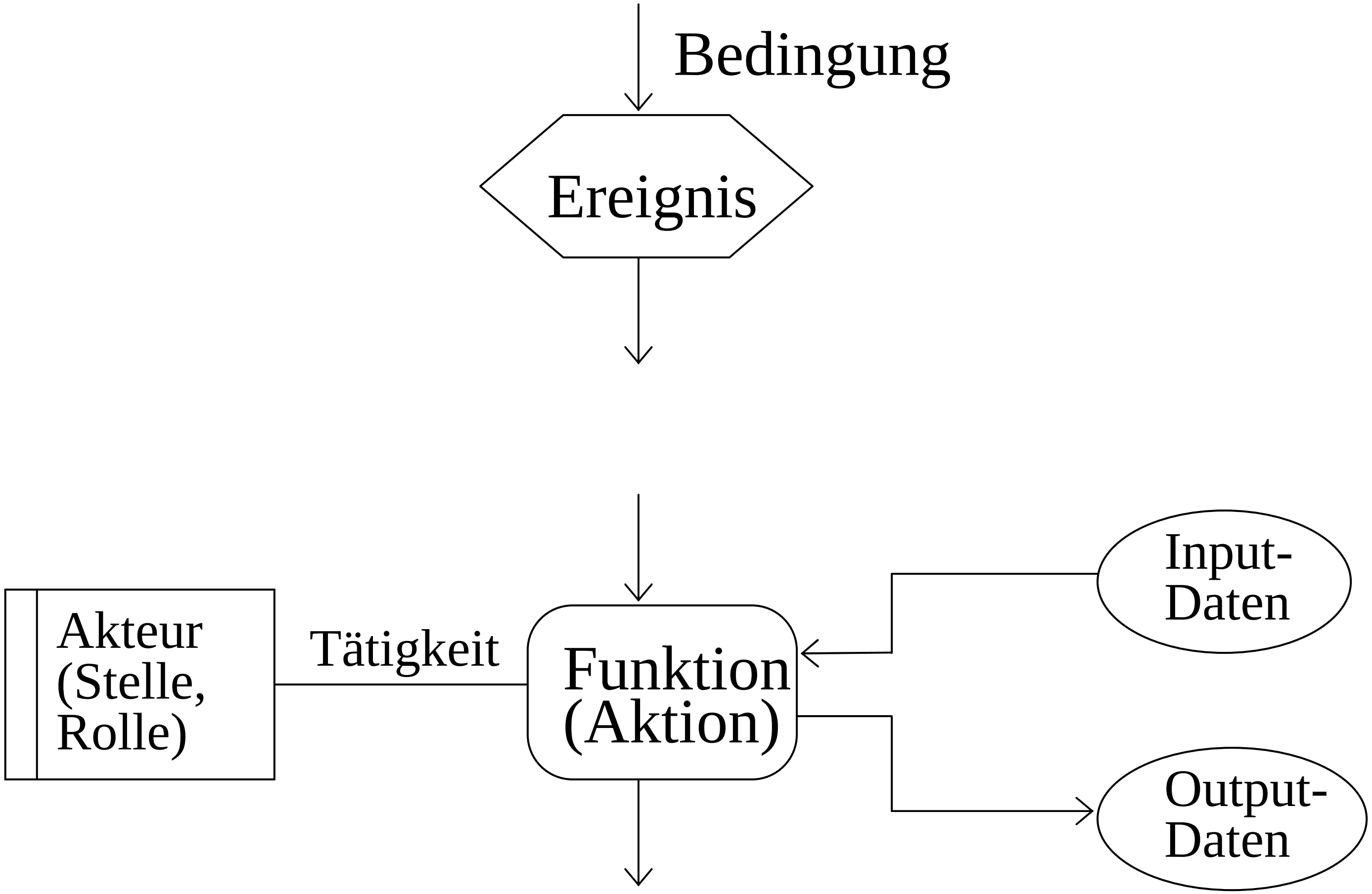
E-Commerce Workflow: Statechart



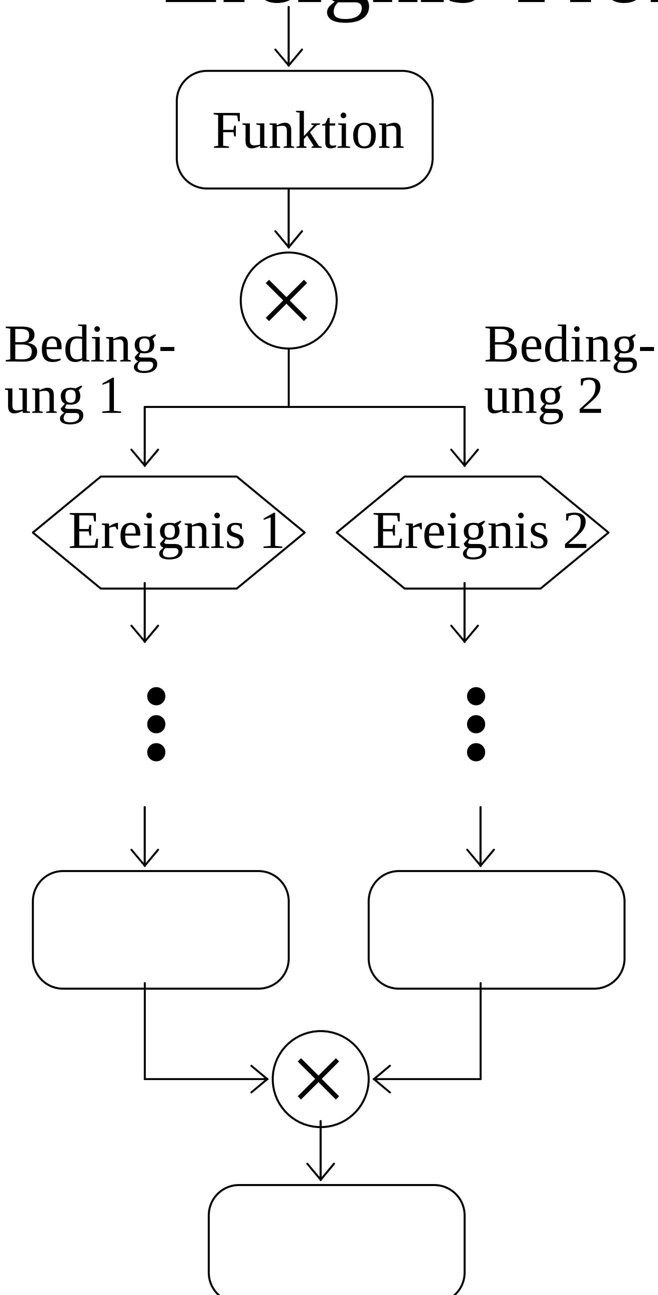
E-Commerce Sub-Workflows



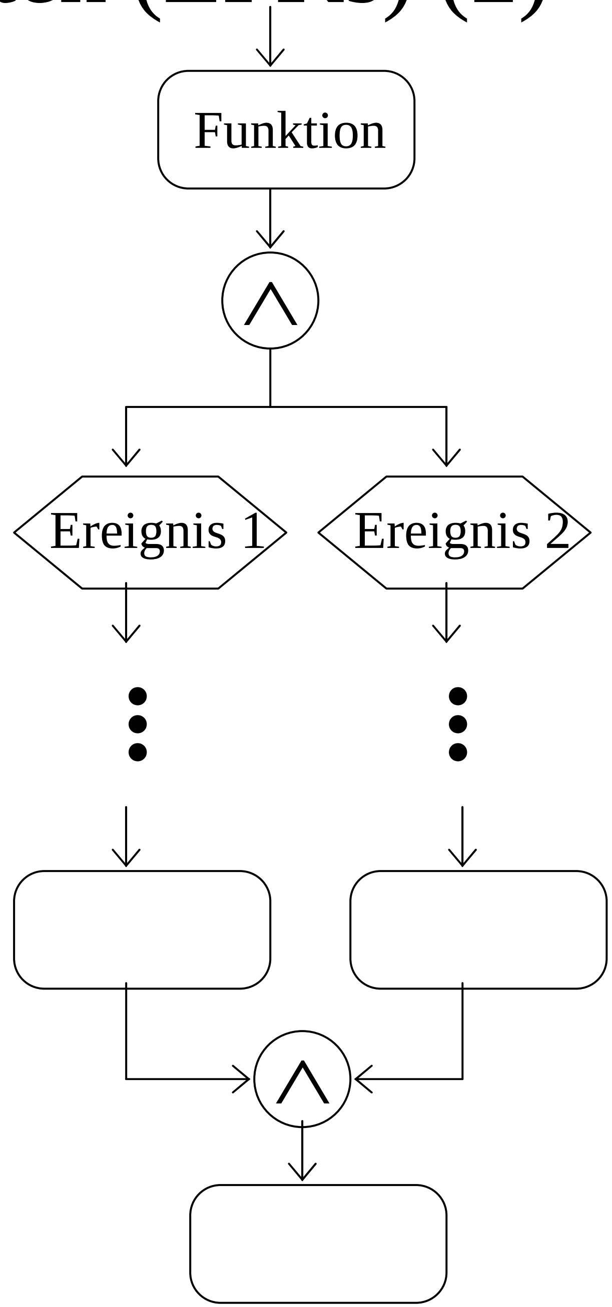
Ereignis-Prozeß-Ketten (EPKs) (1)



Ereignis-Prozeß-Ketten (EPKs) (2)

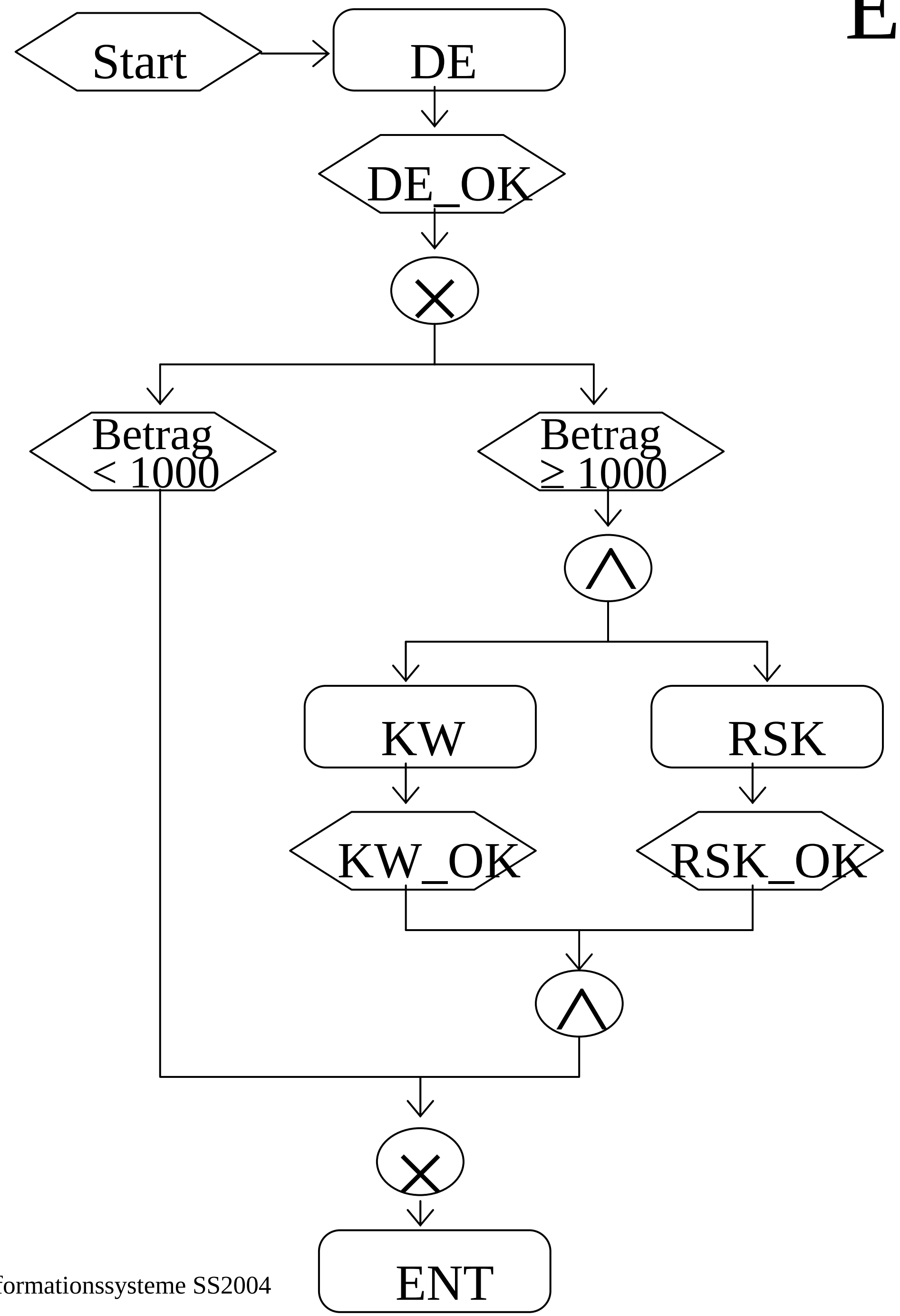


Verzweigung



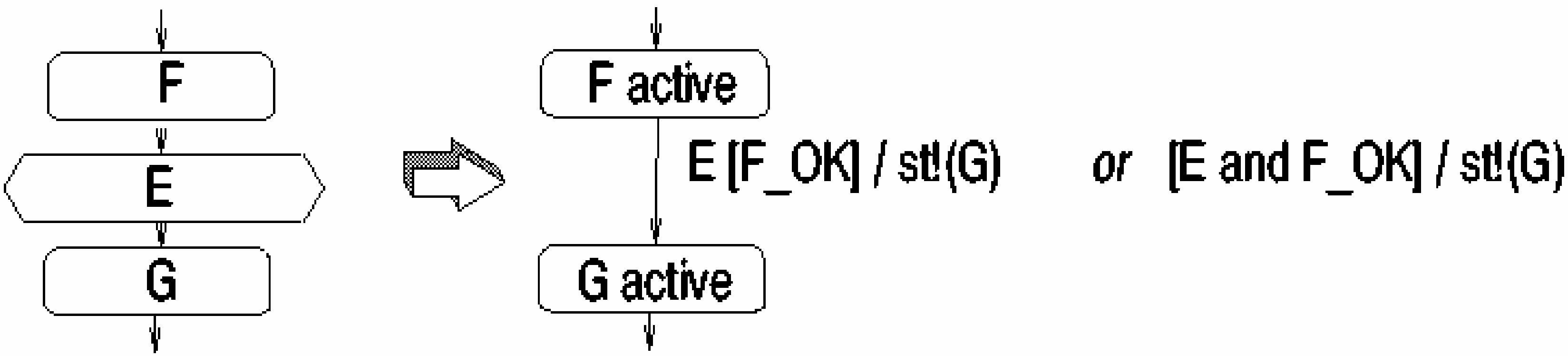
(Fork-Join-) Split

EPK-Beispiel



Import from BPR Tools

Principle:

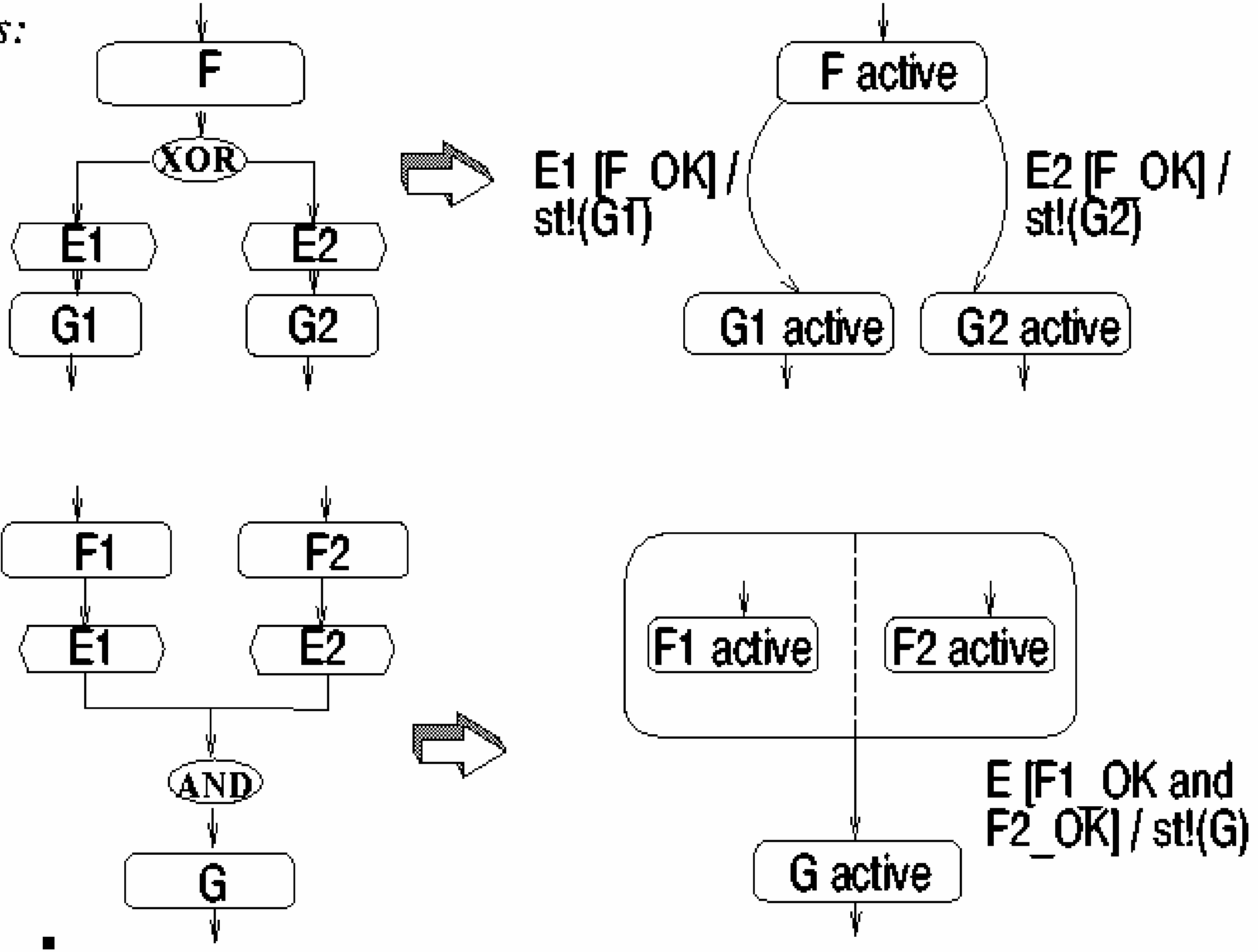


Event process chains
(EPCs à la Aris Toolset):

- process decomposed into functions
- completed functions raise events
  that trigger further functions
- control-flow connectors

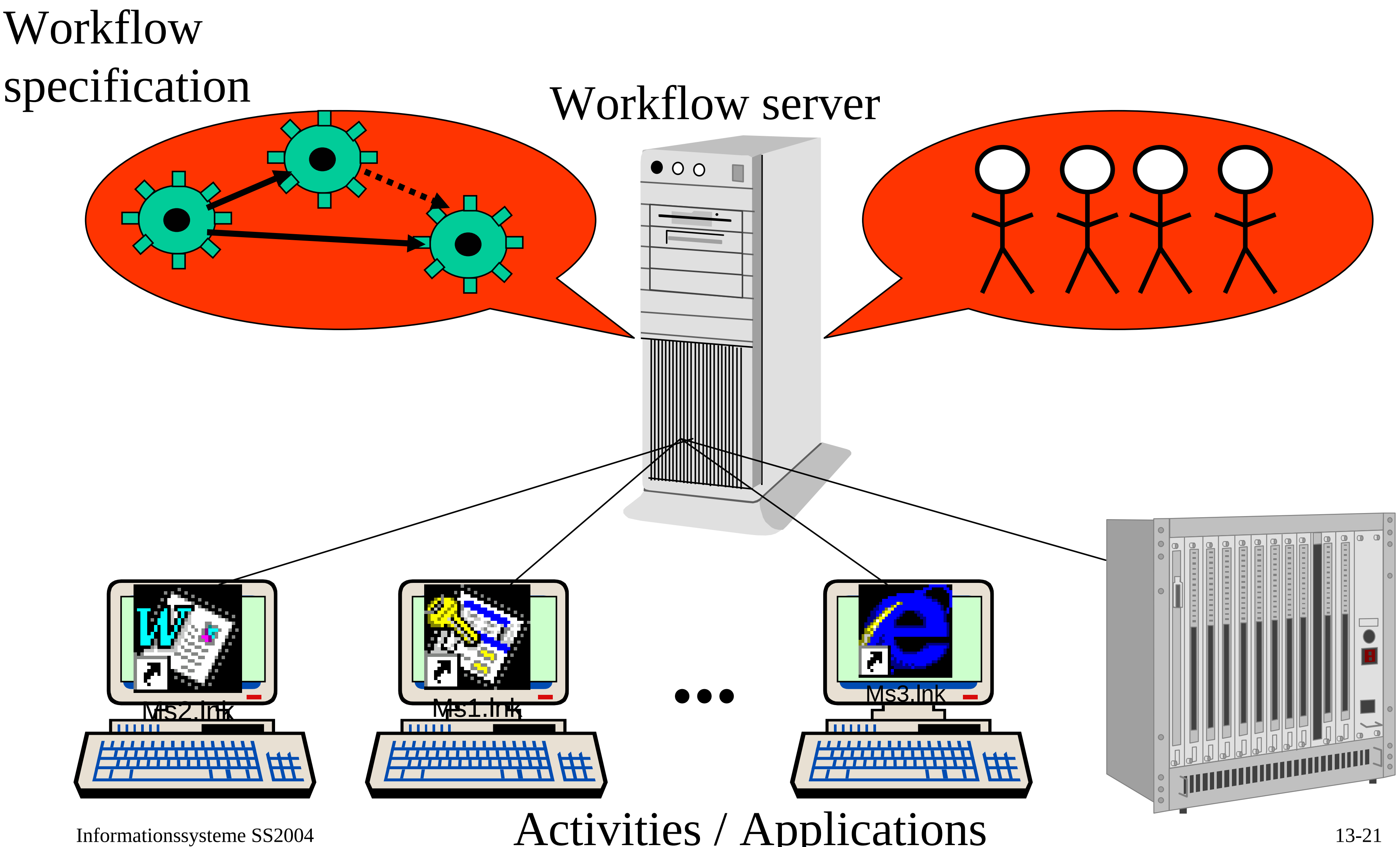
Import from BPR Tools (continued)

Some Subtleties:

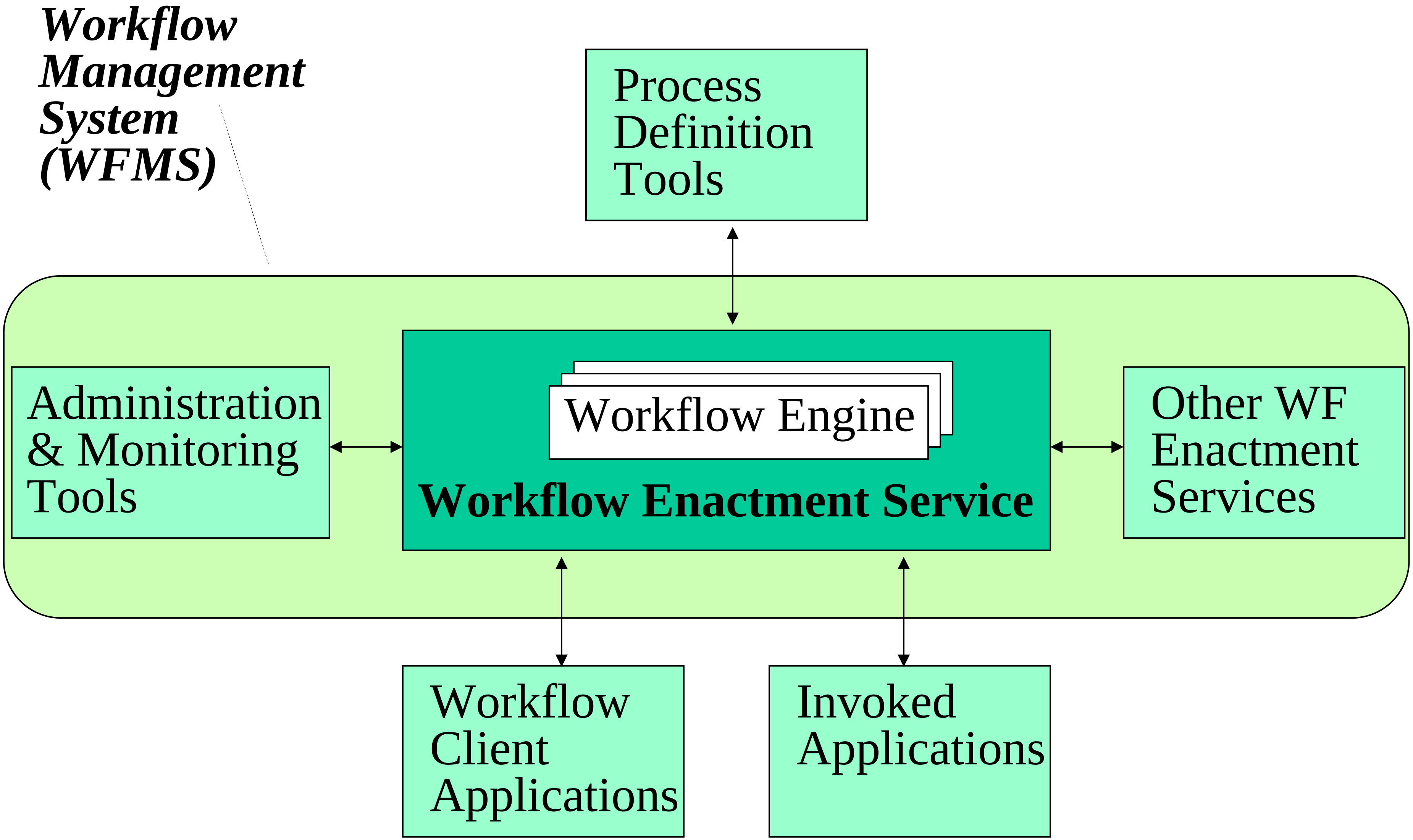


⋮

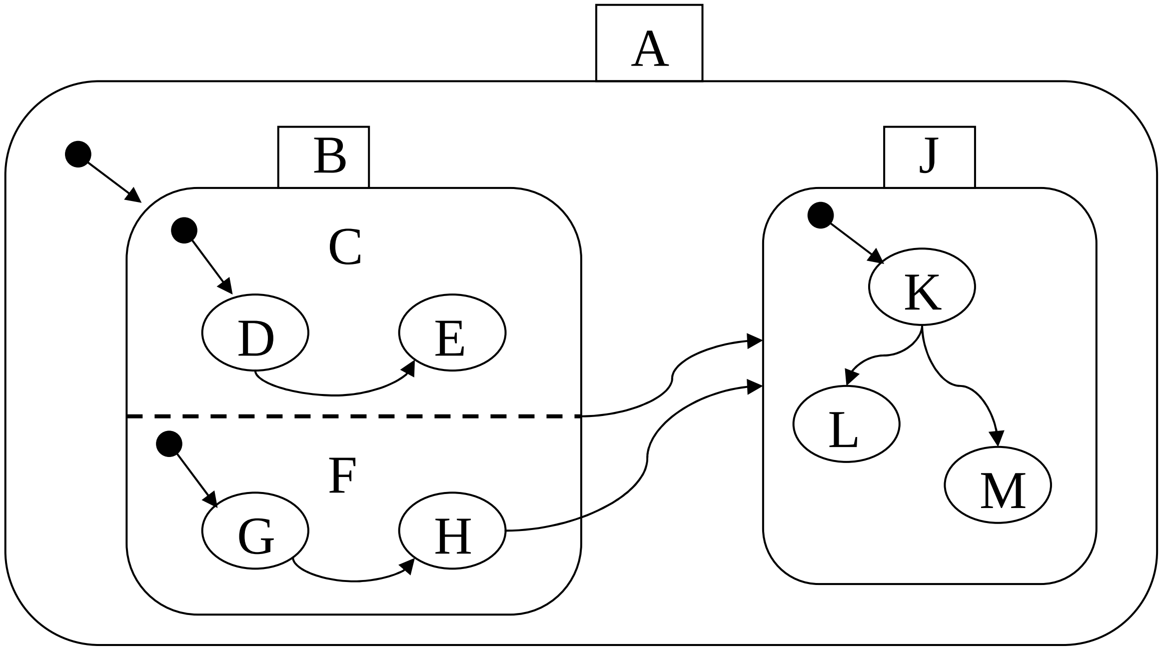
13.2 Workflow Management System Architecture



WfMC Reference Architecture



13.3 Abstract Syntax of Statecharts (1)



State set S

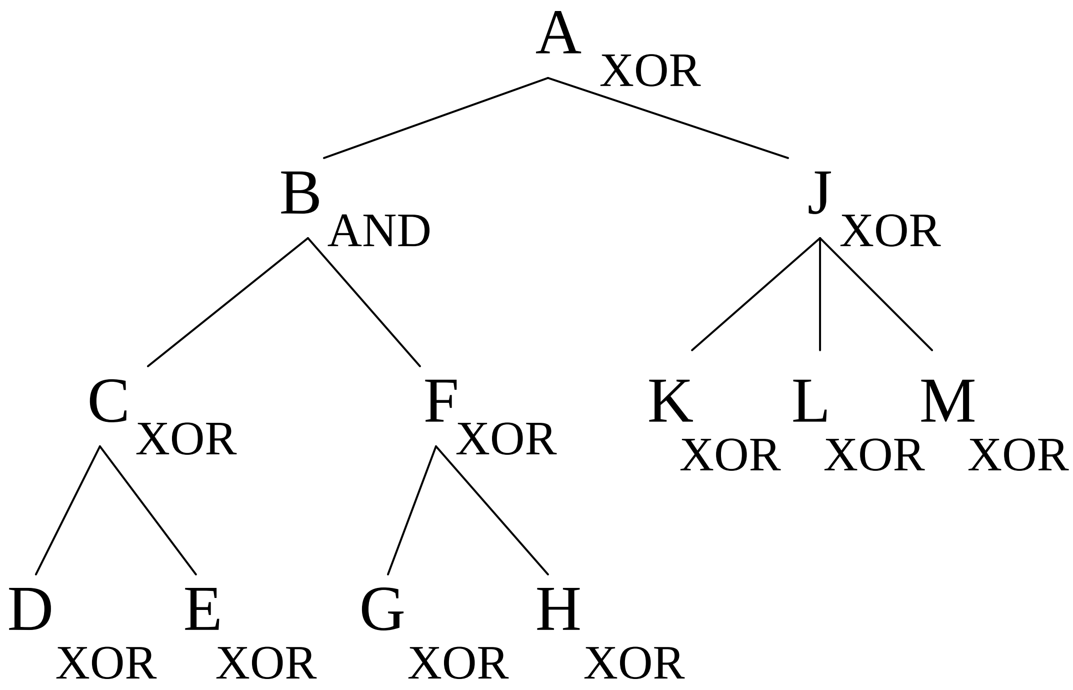
State tree (with node types AND or XOR)

Transition t: (source, target, [c]/a)

Transition set T

Variable set V

Abstract Syntax of Statecharts (2)



Operational Semantics of Statecharts (1)

Execution state of statechart (S,T,V):
subset *states* $\subseteq$ S of currently active states s.t.
- root of S is in states
- if s in states and type of s is AND then all children of s are in states
- if s in states and type of s is XOR
  then exactly one child of s is in states

Execution context of statechart (S,T,V):
current values of variables defined by *val*: V $\rightarrow$ Dom

Configuration of statechart (S,T,V): *(states, val)*
Initial configuration

Operational Semantics of Statecharts (2)

Evaluation of expression in configuration:
eval (expr, conf) defined inductively

Effect of action on context:
modification of variable values in val

fire(conf) = set of transitions
t = (source, target, [cond]/action)
with source(t) in states for which eval(cond, conf) = true

Operational Semantics of Statecharts (3)

for transition t:

- a = lca (source(t), target(t))
- src(t) = child of a in subtree of source(t)
- tgt(t) = child of a in subtree of target(t)

when t fires:

- set of left states **source*(t):**
  - src(t) is in source*(t)
  - if s in source*(t) then all children of s are in source*(t)
- set of entered states **target*(t):**
  - tgt(t) and target(t) are in target*(t)
  - if s in target*(t) and type of s is AND then all children of s are in target*(t)
  - if s in target*(t) and type of s is XOR then exactly one child of s with initial transition is in target*(t)

Operational Semantics of Statecharts (4)

For a given configuration conf = (states, val) a **successor configuration** conf‘ = (states‘, val‘) is derived by selecting one transition t from fire(conf) with the effect:

- states‘ = states – source*(t) $\cup$ target*(t)
- val‘ captures the effect of action(t) and equals val otherwise

The operational semantics of a statechart (S,V,T) is the set of all possible executions along configurations $conf_0$, $conf_1$, $conf_2$, ... with

- initial configuration conf0 and
- $conf_{i+1}$ being a successor configuration of $conf_i$

Digression: Finite State Automata

Definition:

Ein endlicher Automat (finite state automaton) ist ein 5-Tupel

$M = (Z, \Sigma, \delta, z0, E)$ mit

- einer endlichen Zustandsmenge Z
- einem Alphabet (d.h. einer endlichen Menge von Zeichen) Σ
- einer Transitionsfunktion $\delta: Z \times \Sigma \rightarrow Z$
- einem Startzustand $z0$
- einer Menge von Endzuständen $E \subseteq Z$

M geht in $z \in Z$ mit Eingabe $x \in \Sigma$ in $\delta(z,x) \in Z$ über.

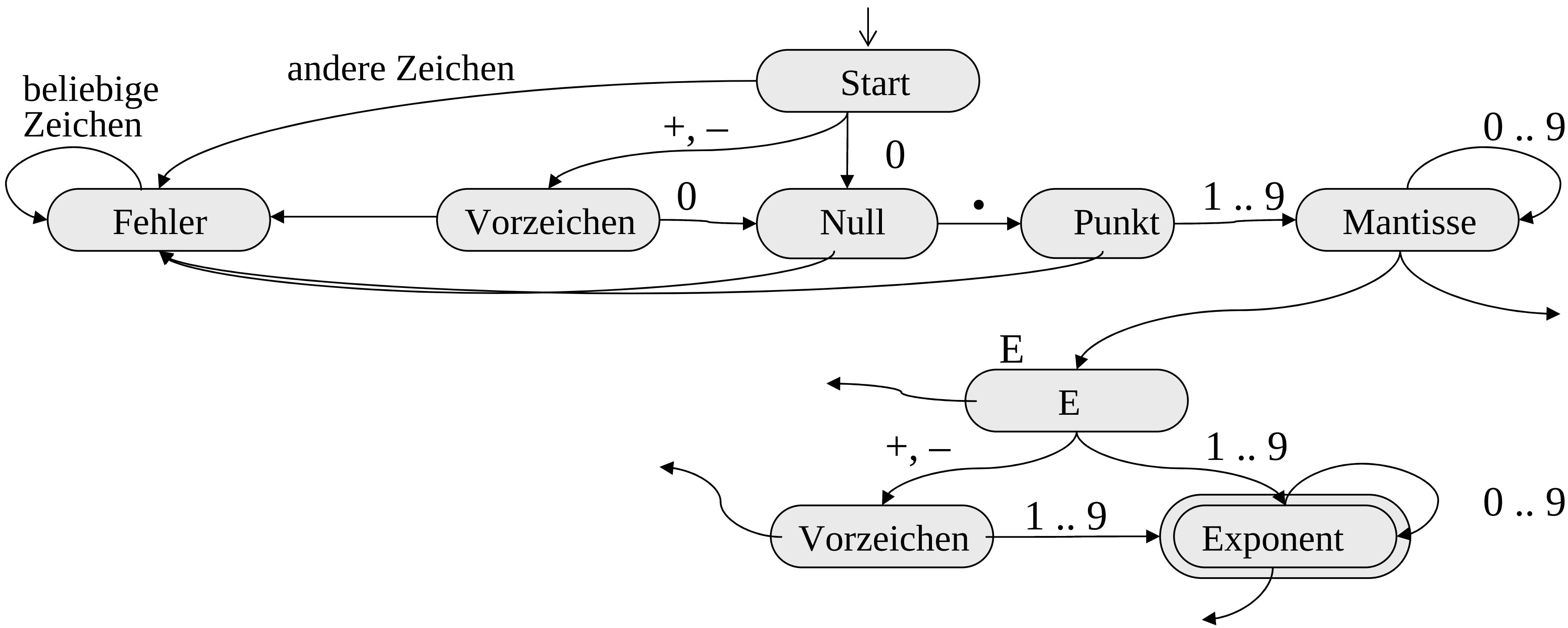
δ wird homomorph zur Funktion $\delta^*: Z \times \Sigma^* \rightarrow Z$ erweitert:

$\delta^*(z, au) = \delta^*(\delta(z,a),u)$ mit $z \in Z$, $a \in \Sigma$, $u \in \Sigma^*$.

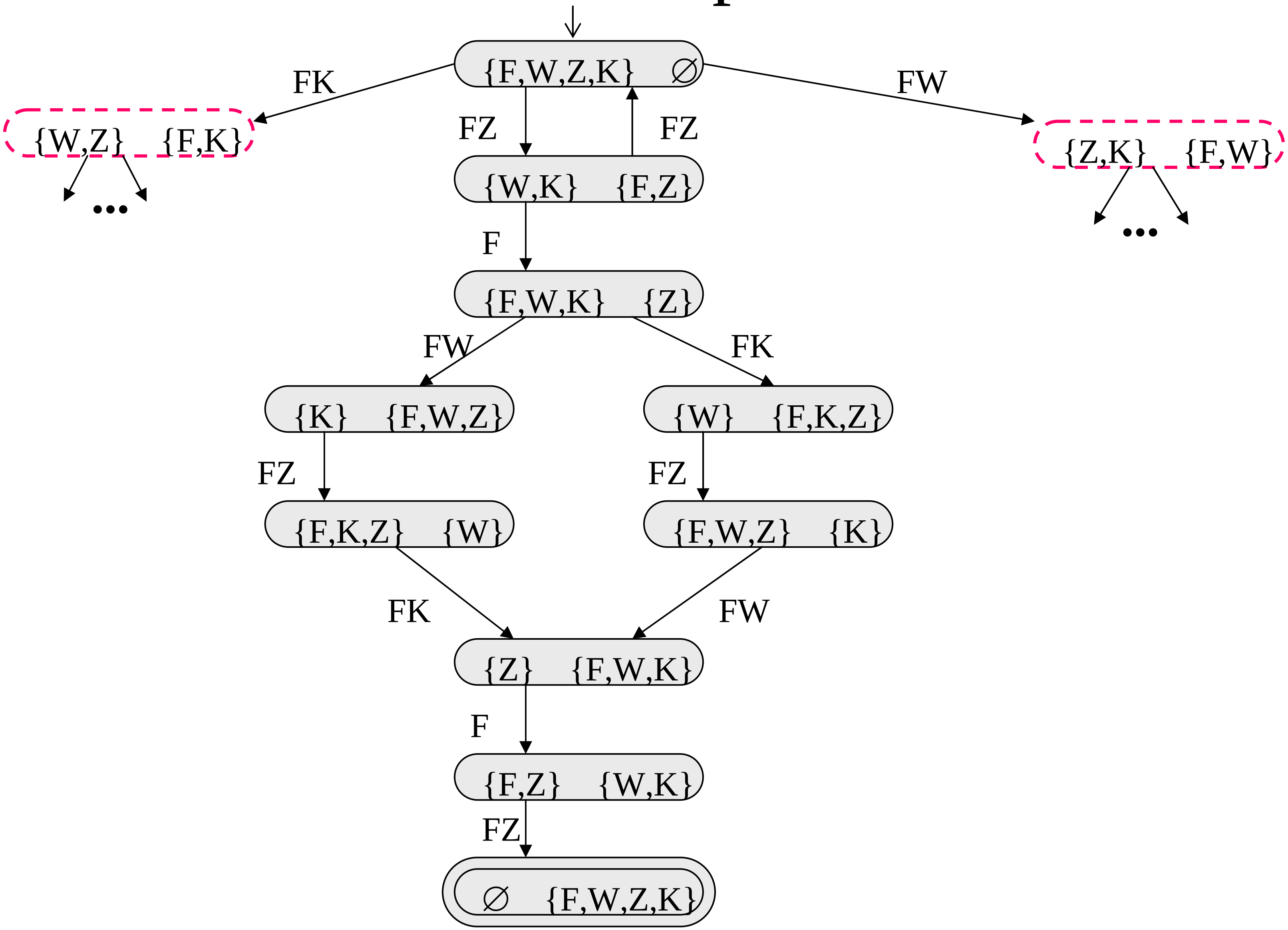
Die Menge $L(M) = \{w \in \Sigma^* \mid \delta^*(z0,w) \in E\} \subseteq \Sigma^*$

ist die vom Automat M akzeptierte Sprache.

FSA Example 1



FSA Example 2



Mapping Statecharts into FSAs

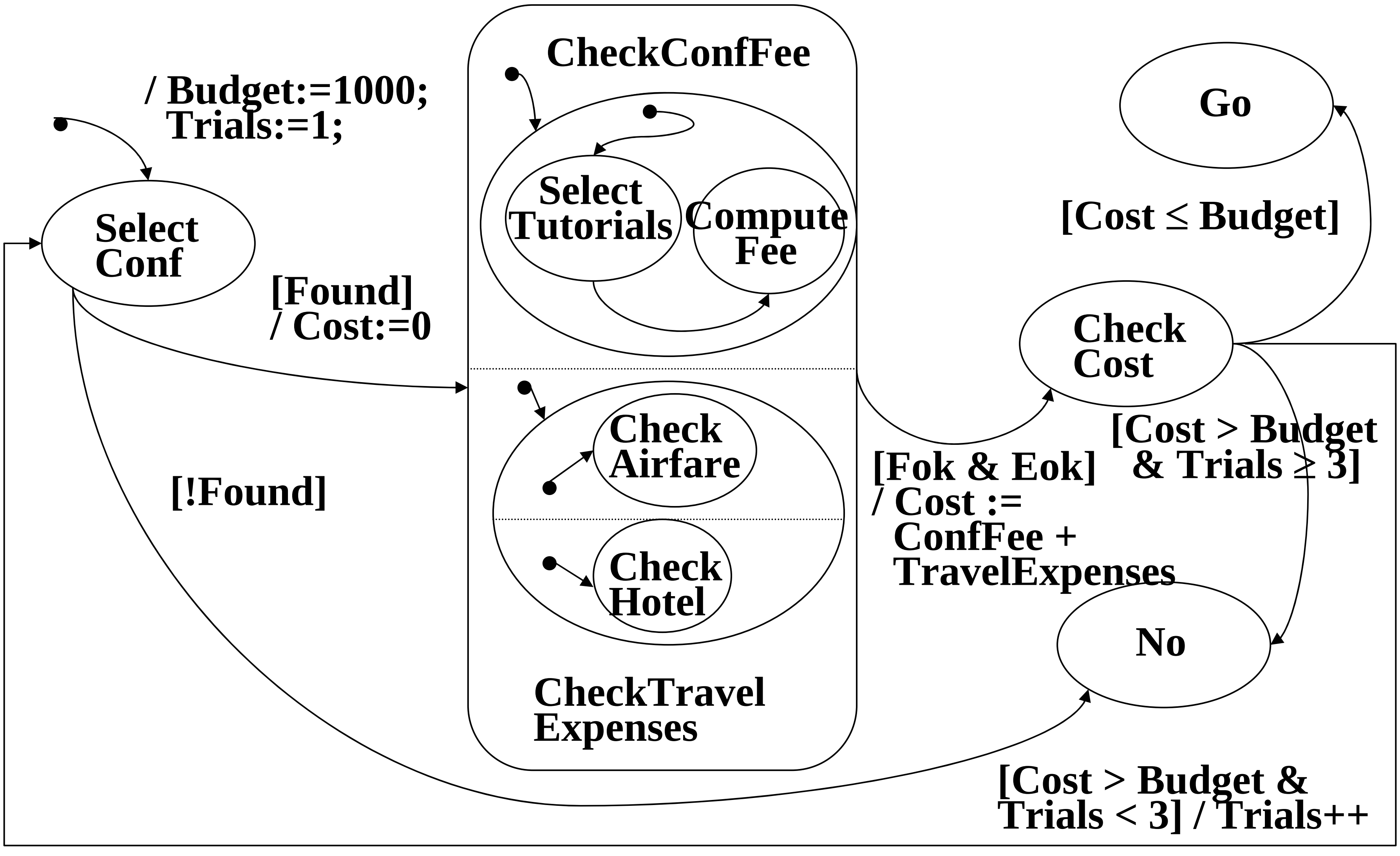
Represent SC configurations as states of a FSA:

Step 1:
abstract conditions on infinite-domain variables into Boolean vars
formal mapping: $\psi 1$: val $\rightarrow$ B1 $\times$ B2 $\times$ ... $\times$ Bm

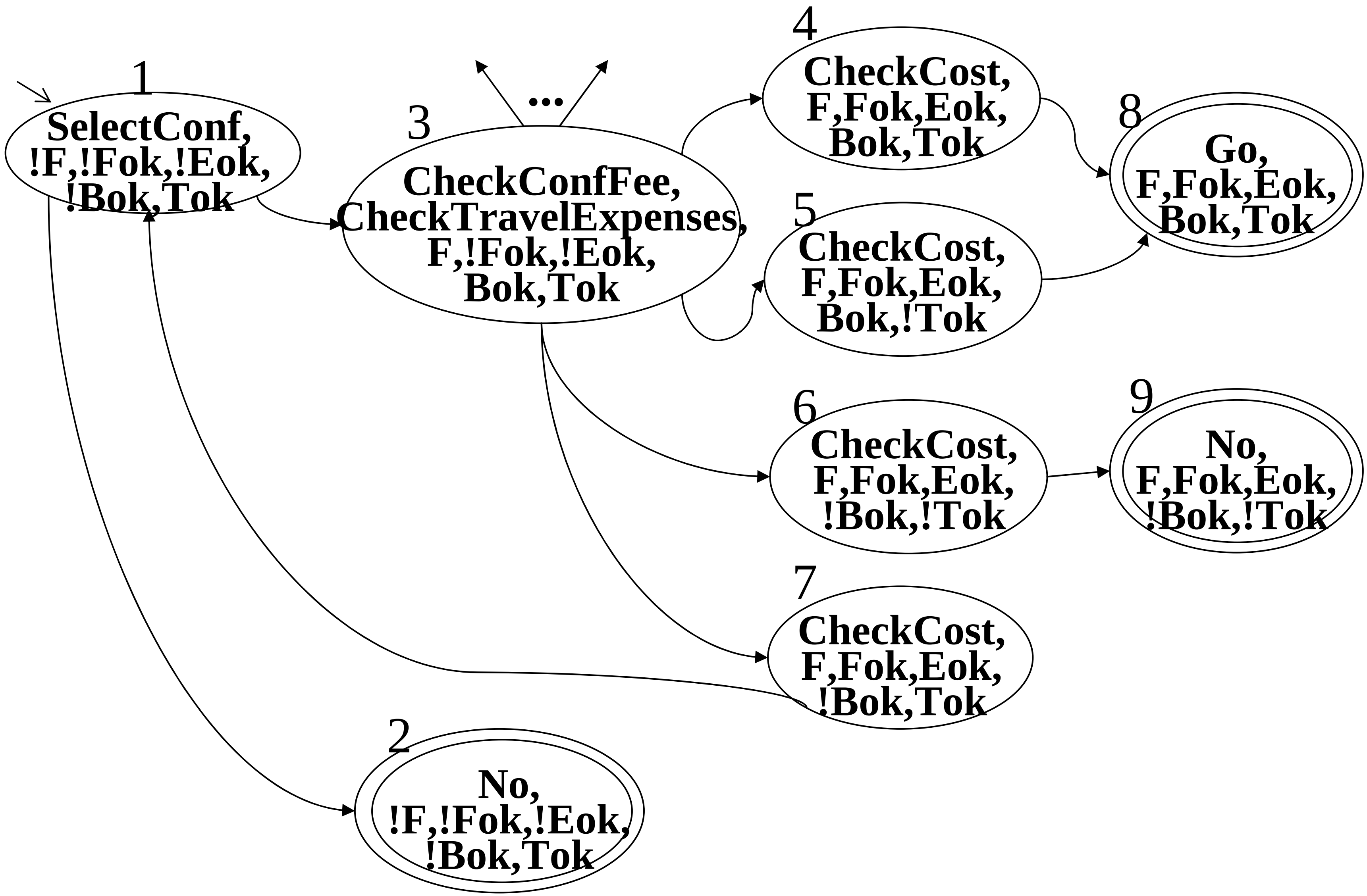
Step 2:
capture set of active SC states (in SC hierarchy and in components)
by powerset automaton $\psi 2$: states $\rightarrow 2^S$ =: Z

Step 3:
encode SC context into extended state space of FSA
by an injective mapping $\psi 3$: Z $\times$ B1 $\times$ B2 $\times$ ... $\times$ Bm $\rightarrow$ Z'
such that there is a transition from z1 to z2 in the FSA
iff $\psi 3^{-1}$(z2) is a possible successor configuration of $\psi 3^{-1}$(z1) in the SC

Example: From SC To FSA (1)



Example: From SC To FSA (2)



13.4 Guaranteed Behavior and Outcome of Mission-critical Workflows

Crucial for workflows in
banking, medical applications, electronic commerce, etc.

- **Safety** properties (invariants):
  nothing bad ever happens
- **Liveness** properties (termination, fairness, etc.):
  something good eventually happens

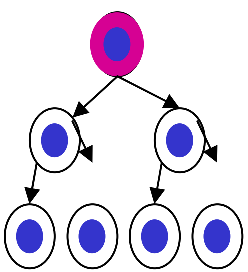
• Mathematical model	→	Finite-state automaton
• Formalization of properties	→	Temporal logic
• Verification method	→	Model checking

CTL: Computation Tree Logic

- propositional logic formulas
- quantifiers ranging over execution paths
- modal operators referring to future states

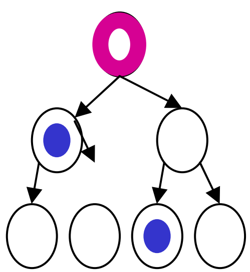
all next:

AX p



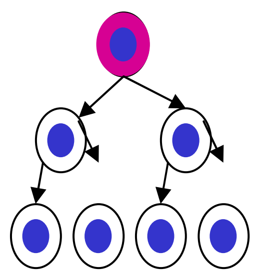
exists next:

EX p



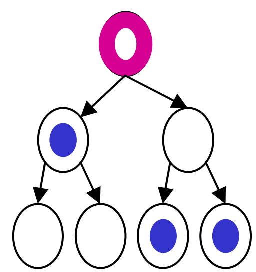
all globally:

AG p



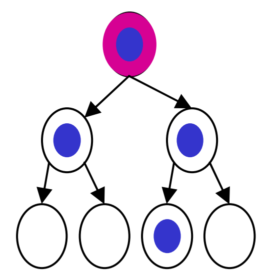
all finally (inevitably):

AF p



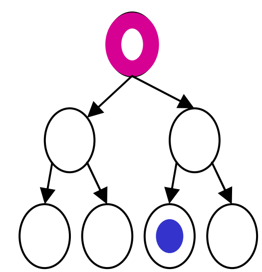
exists globally:

EG p



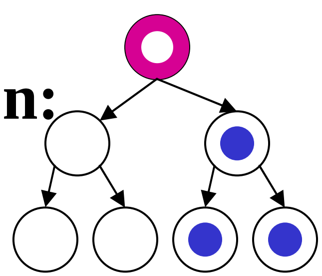
exists finally (possibly):

EF p



combination:

EF AG p



Critical Properties of the Example Workflow

formalized in CTL (Computation Tree Logic)

- ***Can we ever exceed the budget ?***

  not EF (in(Go) and !Bok)

  ≡ AG (not in(Go) or Bok)

- ***Do we always eventually reach a decision ?***

  AF (in(Go) or in(No))

- ***Can the trip still be approved after a proposal that would have exceeded the budget ?***

  EF ((in(CheckCost) and !Bok) => (EF (in(Go))))

CTL Syntax

<u>Definition:</u>

Eine atomare *CTL-Formel* ist eine aussagenlogische Formel über elementaren Aussagen (bzw. Booleschen Variablen).

Die Menge der in CTL erlaubten Formeln ist induktiv wie folgt definiert:

- **Jede atomare CTL-Formel ist eine Formel.**
- **Wenn P und Q Formeln sind, dann sind auch EX (P), AX (P), EG (P), AG (P), EF (P), AF (P), (P), $\neg$P, P$\wedge$Q, P$\vee$Q, P$\Rightarrow$Q und P$\Leftrightarrow$Q Formeln.**

CTL Semantik (1)

Definition:

Gegeben sei eine Menge P atomarer aussagenlogischer Formeln.

Eine *Kripke-Struktur* M über P ist ein 4-Tupel (S, s0, R, L) mit

- **einer endlichen Zustandsmenge S,**
- **einem Startzustand s0 $\in$ S,**
- **einer Transitionsrelation R $\subseteq$ S $\times$ S,**
- **einer Funktion L: S $\rightarrow 2^P$, die einem Zustand wahre Aussagen zuordnet.**

Definition:

Eine Kripke-Struktur M = (S, s0, R, L) ist ein *Modell* einer Formel F, wenn M,s0 $\models$ F.

Eine Formel heißt *erfüllbar*, wenn sie mindestens ein Modell hat, ansonsten *unerfüllbar*.

Eine Formel F heißt *allgemeingültig* (oder Tautologie), wenn jede Kripke-Struktur über den atomaren Aussagen von F ein Modell von F ist.

CTL Semantik (2)

Definition:

Die *Interpretation* ψ einer Formel F mit atomaren Aussagen P ist eine Abbildung auf eine Kripke-Struktur $M=(S, s_0, R, L)$ über Aussagen P, so dass die Wahrheitswerte von Teilformeln p bzw. p1, p2 von F in den Zuständen s von M, in Zeichen: M,s |= p, wie folgt sind:

(i) M,s |= p mit einer aussagenlogischen Formel p gilt g.d.w. $p \in L(s)$;

(ii) M,s |= $\neg$p g.d.w. nicht M,s |= p gilt;

(iii) M,s |= p1 $\wedge$ p2 g.d.w. M,s |= p1 und M,s |= p2;

(iv) M,s |= p1 $\vee$ p2 g.d.w. M,s |= p1 oder M,s |= p2;

(v) M,s |= EX p g.d.w. es $t \in S$ gibt mit $(s,t) \in R$ und M,t |= p;

(vi) M,s |= AX p g.d.w. für alle $t \in S$ mit $(s,t) \in R$ gilt: M,t |= p;

(vii) M,s |= EG p g.d.w. es $t_1, ..., t_k \in S$ gibt mit $t_1=s$, $(t_i, t_{i+1}) \in R$ für alle i und $t_k=t_j$ für ein $j: 1 \le j < k$ oder t_k ohne Nachfolger, so dass M,t_i |= p für alle i;

(viii) M,s |= AG p g.d.w. für alle $t \in S$ mit $(s,t) \in R^*$ gilt: M,t |= p;

(ix) M,s |= EF p g.d.w. es $t \in S$ gibt mit $(s,t) \in R^*$ und M,t |= p;

(x) M,s |= AF p g.d.w. es für alle $t \in S$ mit $(s,t) \in R^*$ einen Zustand $t' \in S$ gibt mit a) $(t,t') \in R^*$ oder b) $(s,t') \in R^*$ und $(t',t) \in R^*$, so dass M,t' |= p gilt.

Model Checking

Für CTL-Formel F und Transitionssystem (Kripke-Struktur) M teste, ob M ein Modell von F ist, indem man
induktiv alle Zustände von M mit q markiert, in denen die Teilformel q von F wahr ist.

Sei q eine Teilformel von F, seien p, p1, p2 direkte Teilformeln von q und seien P, P1, P2 die mit p, p1, p2 markierten Zustände von M.

(i) q ist eine atomare Aussage (Boolesche Variable):
Markiere alle Zustände s mit $q \in L(s)$ mit q

(ii) q hat die Form $\neg$p: Markiere S – P mit q

(iii) q hat die Form p1 $\wedge$ p2: Markiere P1 $\cap$ P2 mit q

(iv) q hat die Form p1 $\vee$ p2: Markiere P1 $\cup$ P2 mit q

(v) q hat die Form EX p:
Markiere alle Vorgänger von P mit q, also alle $s \in S$, für die es ein $x \in P$ gibt mit R(s,x)

(vi) q hat die Form AX p:
Markiere s mit q, wenn alle Nachfolger von s mit p markiert sind

Model Checking: Fall EF

(vii) q hat die Form EF p:

Löse Rekursion EF p $\Leftrightarrow$ p $\vee$ EX (EF p).

(Fixpunktgleichung Q = P $\cup$ pred(Q))

```
Q := P;
Qnew := Q ∪ pred(Q);
while not (Q = Qnew) do
   Q := Qnew;
   Qnew := Q ∪ pred(Q);
od;
```

Model Checking: Fall EG

(viii) q hat die Form EG p:

Löse Rekursion EG p $\Leftrightarrow$ p $\wedge$ EX (EG p) :

```
Q := P;
Qnew := Q ;
repeat
for each s in Q do
      if s has successors and
            no successor of s is in Q
    then Qnew := Q - {s}; fi;
od;
until (Q = Qnew);
```

Model Checking: Fall AG

(ix) q hat die Form AG p:

Löse Rekursion AG p $\Leftrightarrow$ p $\wedge$ AX (AG p)

```
Q := P;
repeat
     Qnew := Q;
     for each s in Q do
          if s has successors and
             one successor of s is not in Q
          then Q := Q - {s} fi;
     od;
until (Q = Qnew);
```

Alternativ wegen AG p $\Leftrightarrow \neg$ EF ($\neg$p):
Berechne Zustandsmenge Q' zur Formel EF ($\neg$p)
und markiere dann die Zustandsmenge S – Q' mit q.

Model Checking: Fall AF

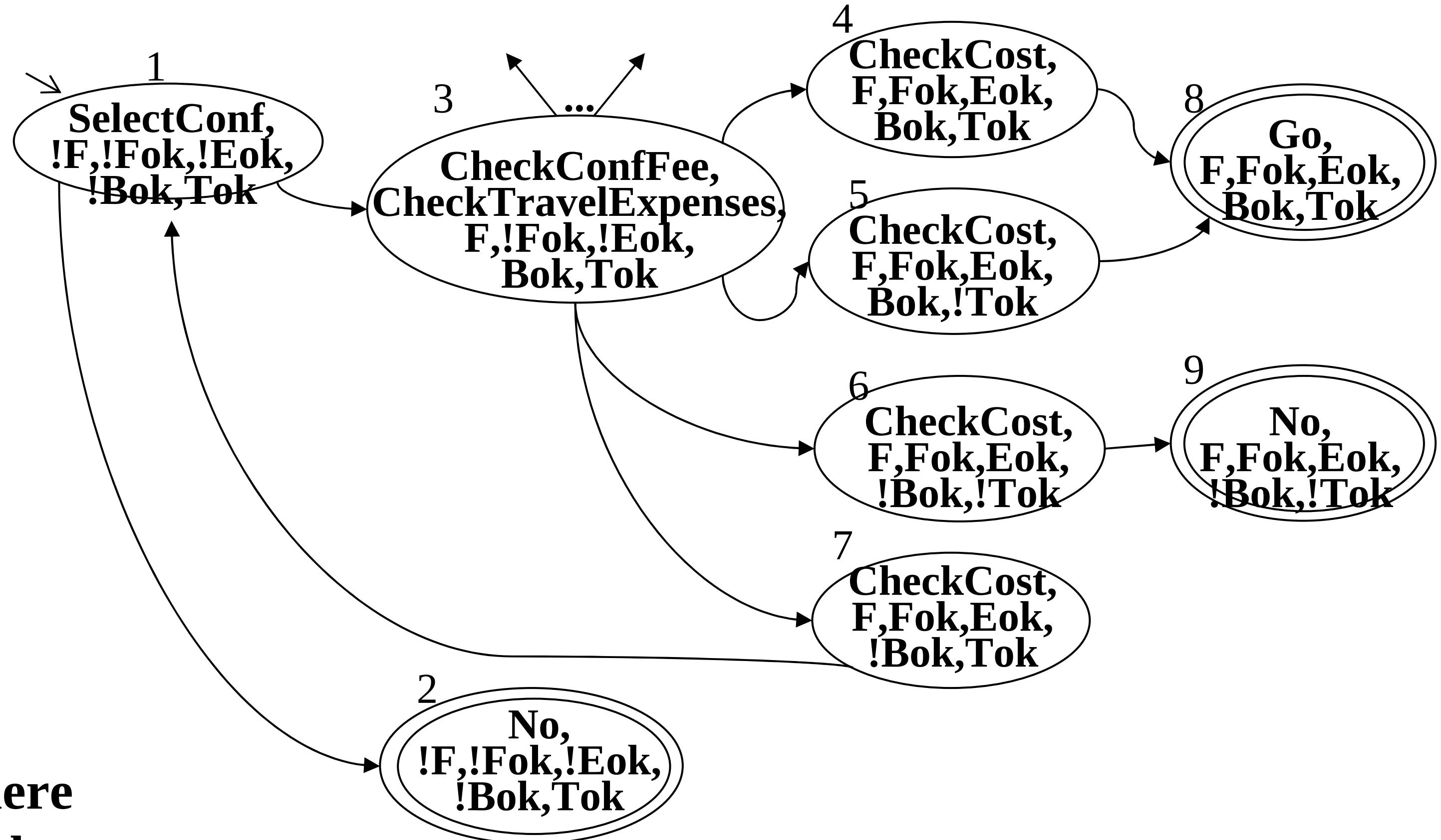
(x) q hat die Form AF p:
Löse Rekursion AF p $\Leftrightarrow$ p $\vee$ AX (AF p)

```
Q := P;
repeat
     Qnew := Q;
for each s in pred(Q) do
          if all successors of s are in Q
     then Q := Q ∪ {s}; fi;
od;
until (Q = Qnew);
```

Alternativ wegen AF p $\Leftrightarrow$ $\neg$ EG ($\neg$p):
Berechne Zustandsmenge Q' zur Formel EG ($\neg$p)
und markiere dann die Zustandsmenge S – Q' mit q.

Model Checking: Beispiel 1

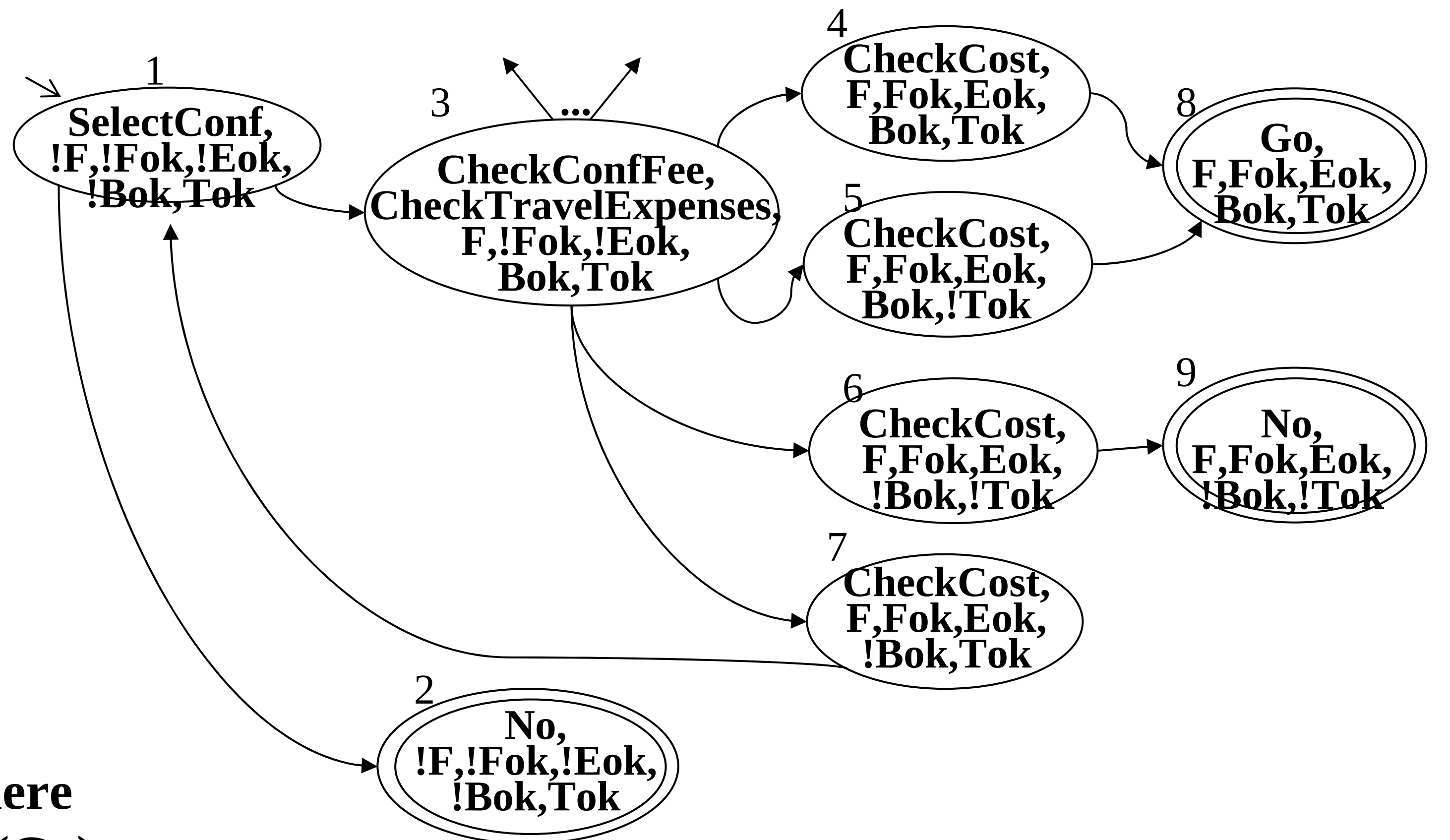
AG (not in(Go) or Bok)



Markiere
mit Bok:
mit in(Go):
mit ¬in(Go):
mit (¬Bok ⇒ ¬in(Go)):
mit AG (¬Bok ⇒ ¬in(Go)):

Model Checking: Beispiel 2

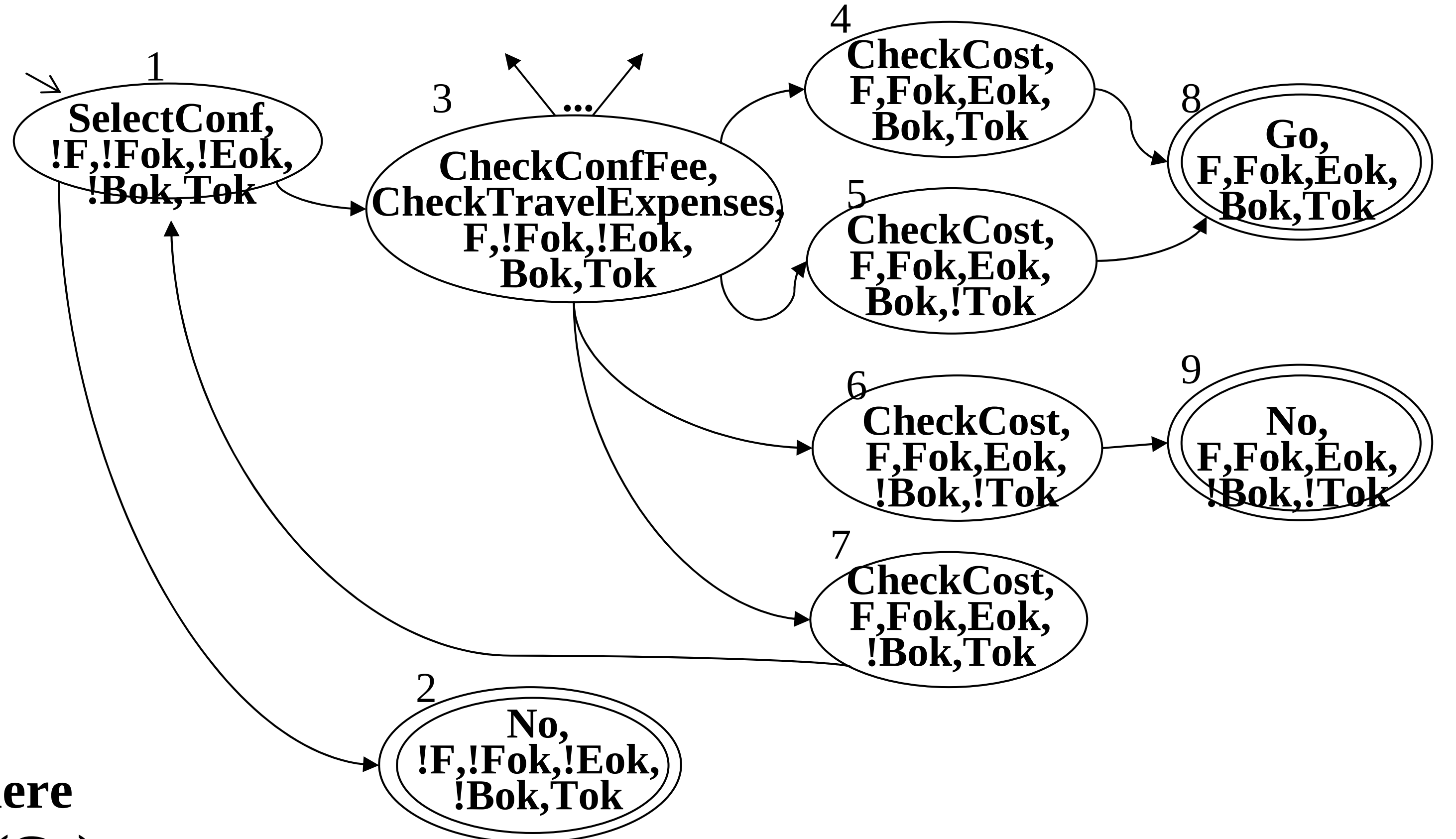
AF (in(Go) ∨ in(No))



Markiere
mit in(Go):
mit in(No):
mit in(Go) ∨ in(No):
mit AF (in(Go) ∨ in(No)):

Model Checking: Beispiel 3

EF ((in(CheckCost) and !Bok) => (EF (in(Go))))



Markiere

mit in(Go):

mit EF (in(Go)):

mit not in(CheckCost) or Bok:

mit (in(CheckCost) and !Bok) => (EF (in(Go)):

mit EF ((in(CheckCost) and !Bok) => (EF (in(Go)))):

Guaranteed Behavior of Workflows

- **Leverage computer-aided verification techniques for finite-state concurrent systems**

- **Efficiency gain with encoding of FSM as OBDD**

- **Further requirements:**
  - **User-friendly macros for CTL**
  - **More expressive logic**
  - **Adding assertions on behavior of invoked apps**
  - **Adding real-time (clock variables)**

- **Preserving guaranteed behavior in distributed, failure-prone system environment**
  → System guarantees