The State Automata Formalism

- Untimed models of discrete event systems
- Languages
- Regular Expressions
- Automata
  - (Deterministic) Finite State Automata
  - Nondeterministic Finite State Automata
  - State Aggregation
  - Discrete Event Systems as State Automata

Untimed models

- Level of specification: I/O System (state based, deterministic)
- Time Base = $\mathbb{N}$ (time $=$ progression index)
- Dynamic but
  - only *sequence* (order) of states traversed matters
  - not *when* in state or *how long* in state
- Discrete Event: event set E

Languages – Regular Expressions – Automata

- *language* L, defined over alphabet E (events) $\equiv$ set of *strings* formed from E
- Example: all possible input behaviours:

$$L = \{\varepsilon, ARR, DEP, ARR\ ARR\ DEP, \ldots\}$$

- Regular expression: shorthand notation for a regular language

$$ARR\ DEP, ARR * DEP *, (DEP|ARR)*$$

  Concatenation, Alternatives ($|$), Kleene closure ($*$).

- Finite State Automaton (model): *generate*/*accept* a language

Finite State Automaton

$$(E, X, f, x_0, F)$$

- E is a finite alphabet
- X is a finite state set
- f is a state transition function,
  $f : X \times E \to X$
- x_0 is an initial state, $x_0 \in X$
- F is the set of final states

Dynamics (x' is next state):

$$x' = f(x, e)$$

FSA recognizes Language

- *extended* transition function:

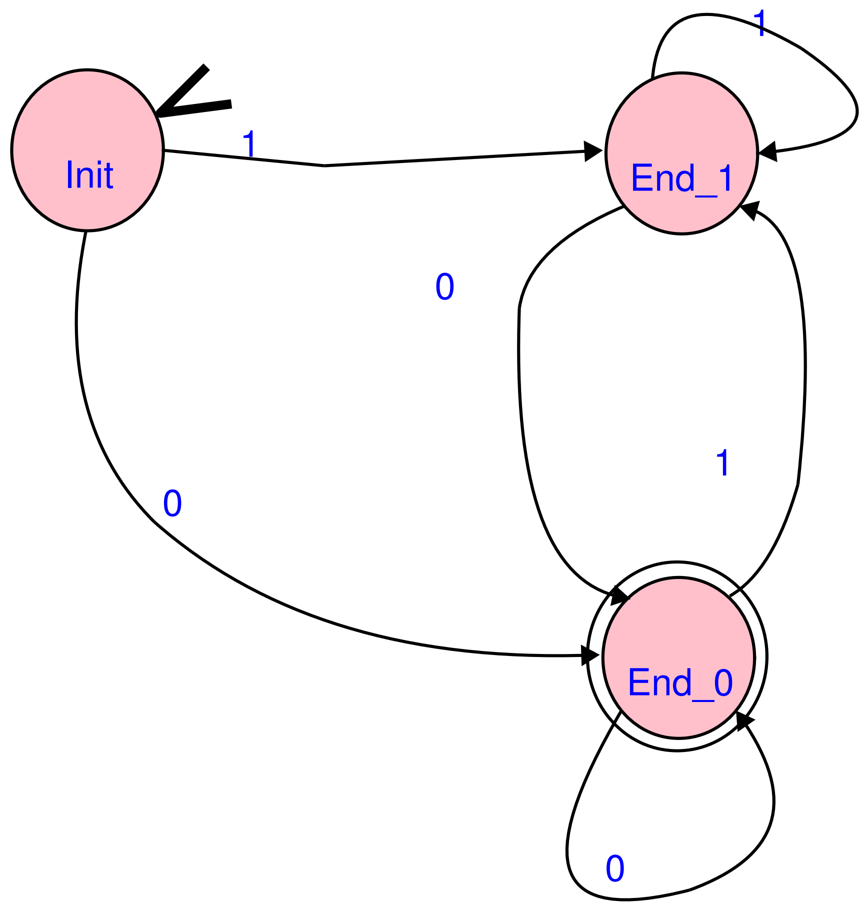
$$f : X \times E* \rightarrow X$$

$$f(x, ue) = f(f(x, u), e)$$

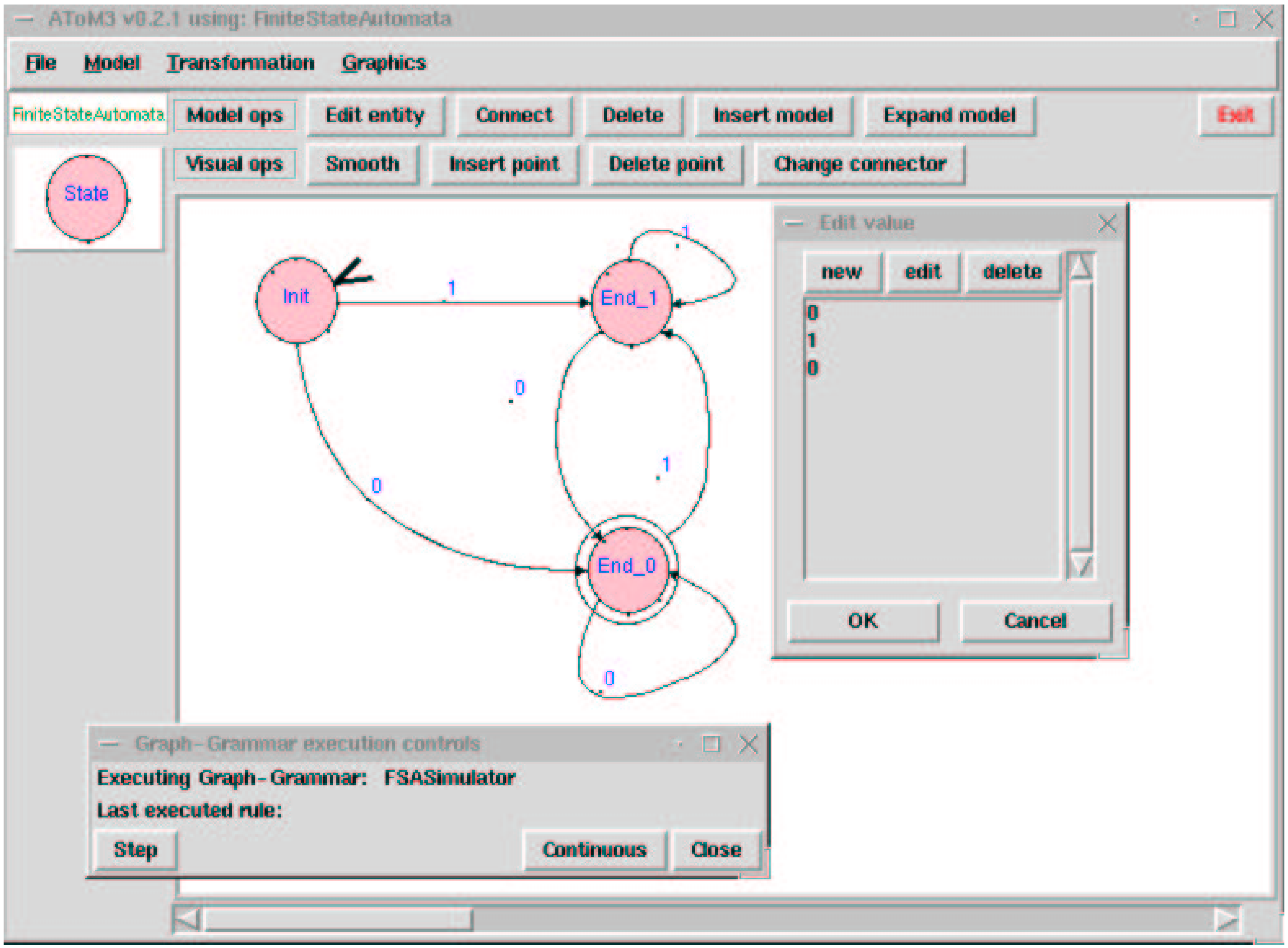
- A string u over the alphabet E is recognized by a FSA (E, X, f, x_0, F) if $f(x_0, u) = x$ where $x \in F$.

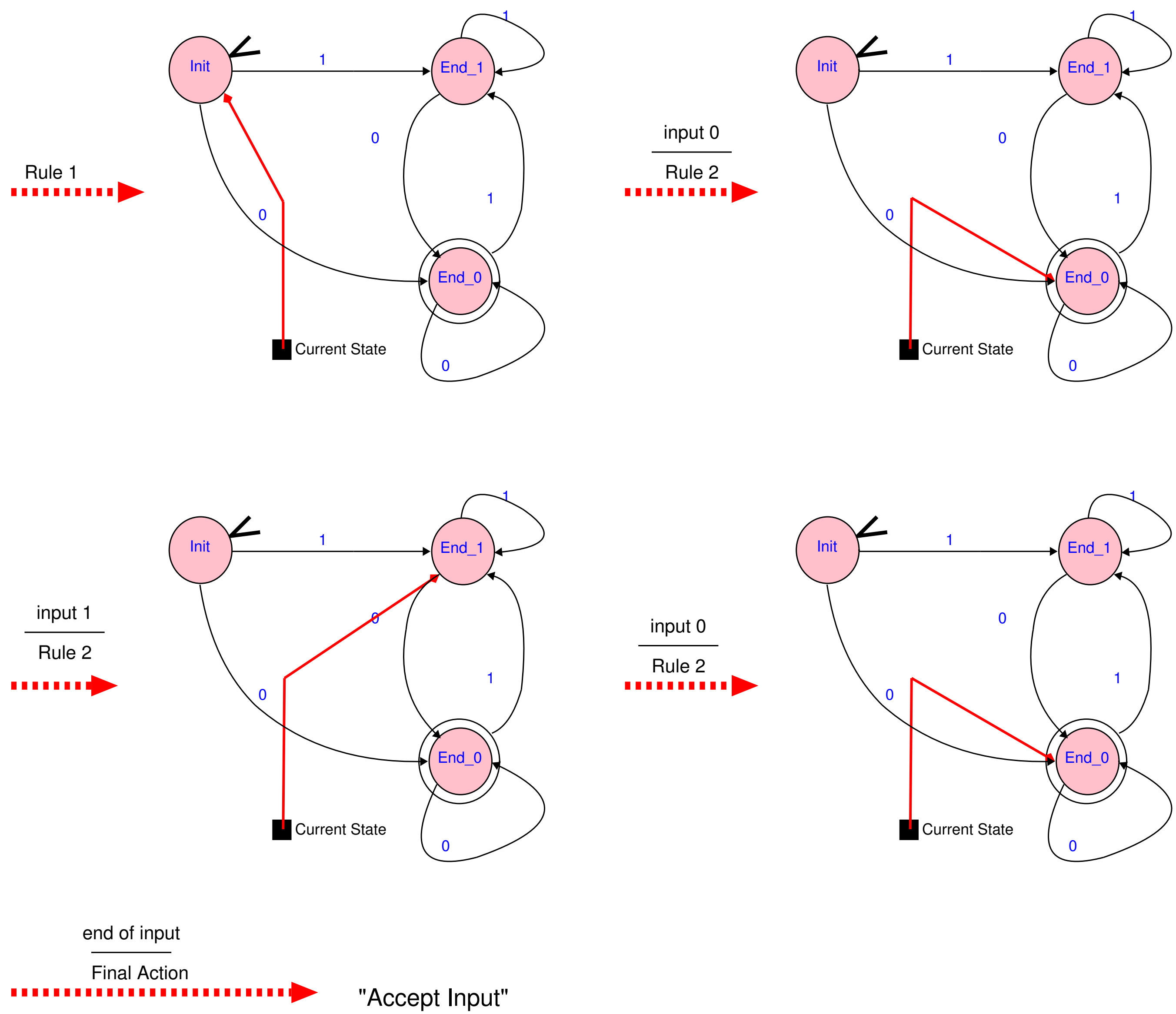
- The language $L(A)$ recognized by a FSA $A = (E, X, f, x_0, F)$ is the set of strings $\{u : f(x_0, u) \in F\}$.

FSA graphical notation: State Transition Diagram



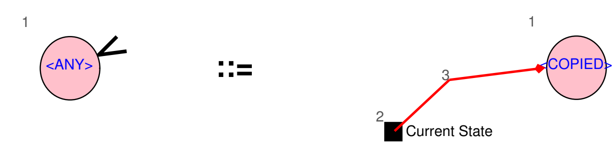
Simulation steps



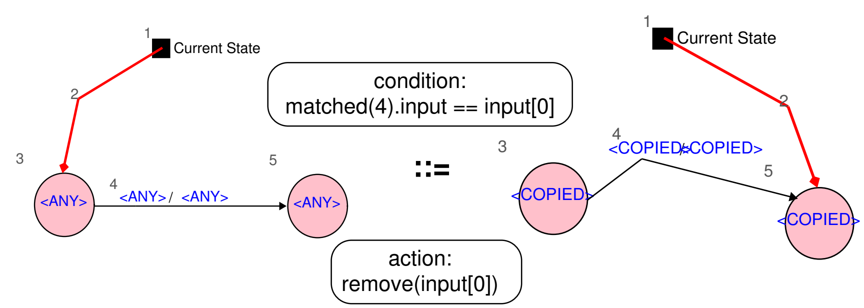

Rule 1
Init
End_1
End_0
Current State
input 0
Rule 2
input 1
Rule 2
input 0
Rule 2
end of input
Final Action
"Accept Input"

FSA Operational Semantics

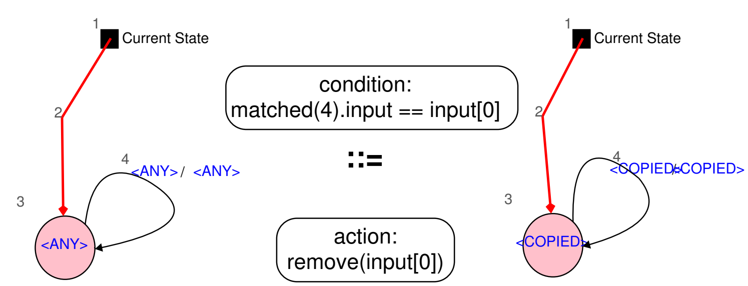
Rule 1 (priority 3) — **Locate Initial Current State**



Rule 2 (priority 1) — **State Transition**



Rule 3 (priority 2) — **Local State Transition**



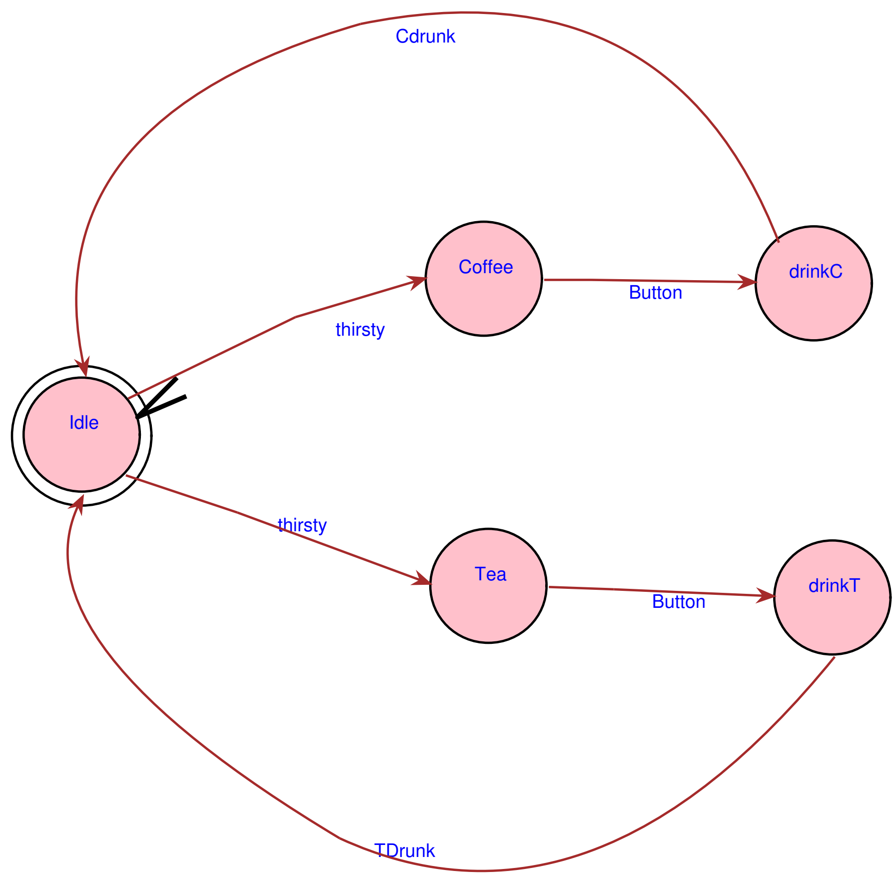
Nondeterministic Finite State Automaton

$$NFA = (E, X, f, x_0, F)$$

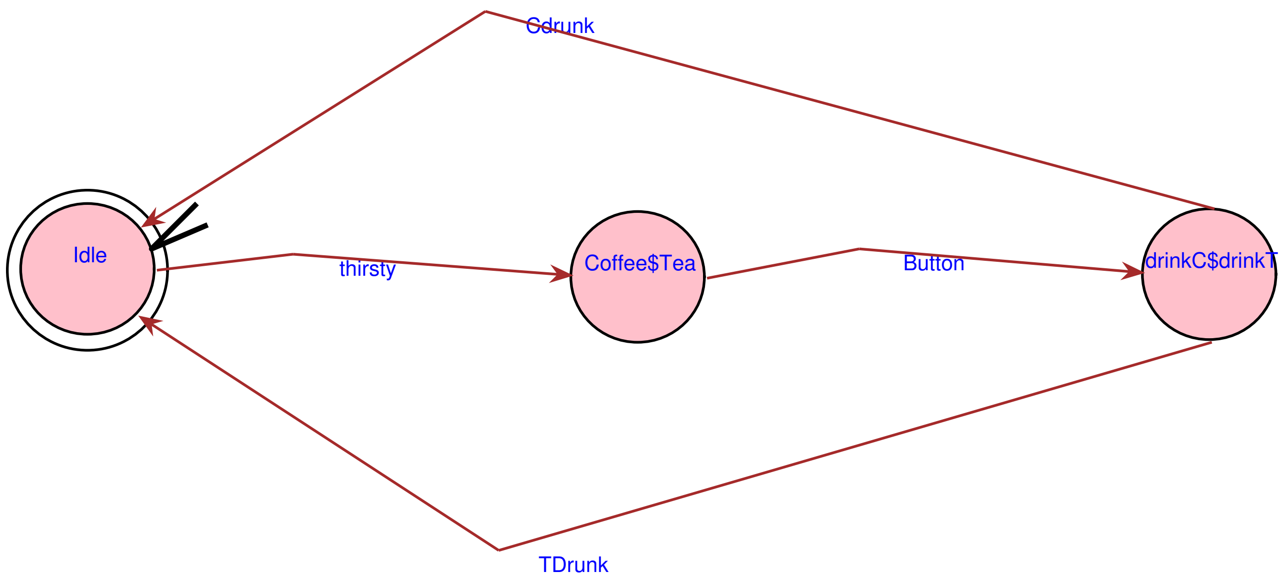
$$f : X \times E \to 2^X$$

- Monte Carlo simulation (if probabilities added)
- Transform to equivalent FSA (*aka* DFA)

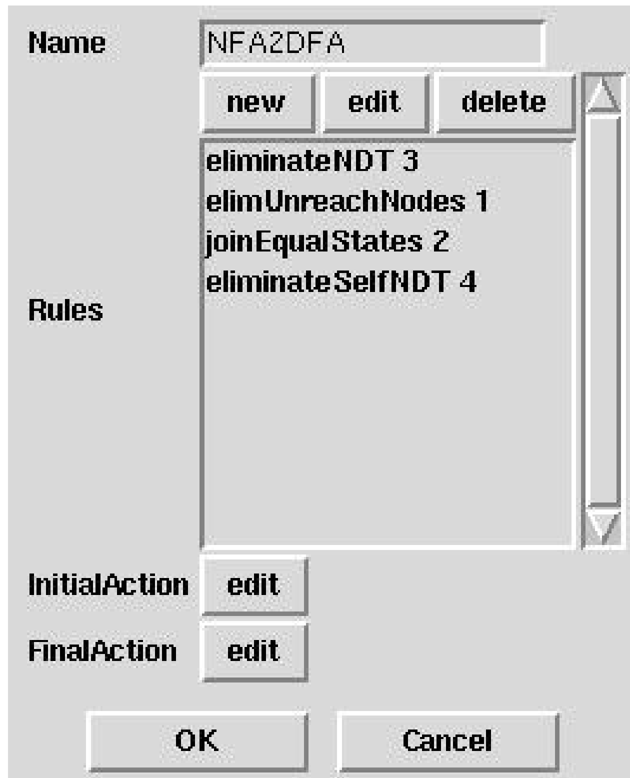
Nondeterministic Finite State Automaton



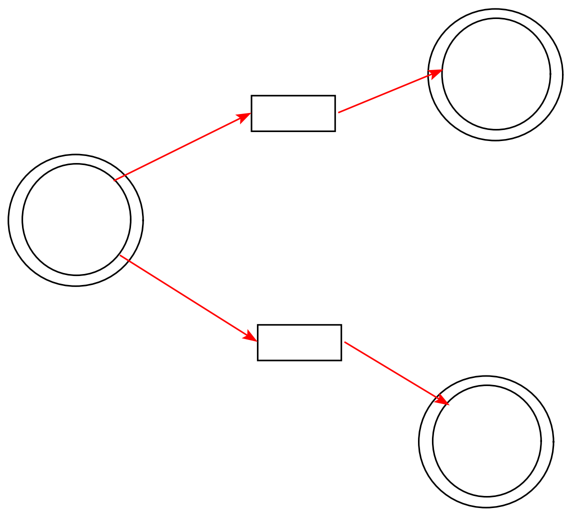
Constructed Deterministic Finite State Automaton



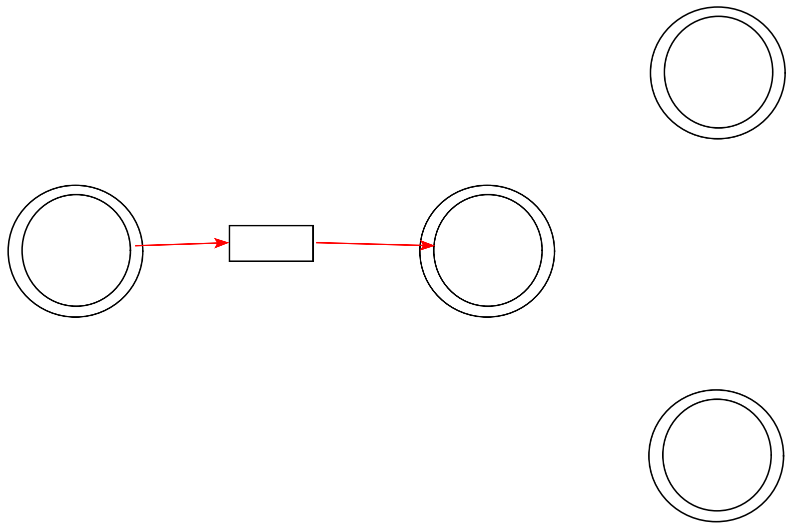
Transformation Rules



Rule LHS



Rule RHS



Managing Complexity: State Aggregation

$$(E, X, f, x_0, F)$$

$$R \subseteq X$$

R consists of *equivalent states with respect to F*
if for any $x, y \in R, x \neq y$ and any string u,

$$f(x,u) \in F \Leftrightarrow f(y,u) \in F$$

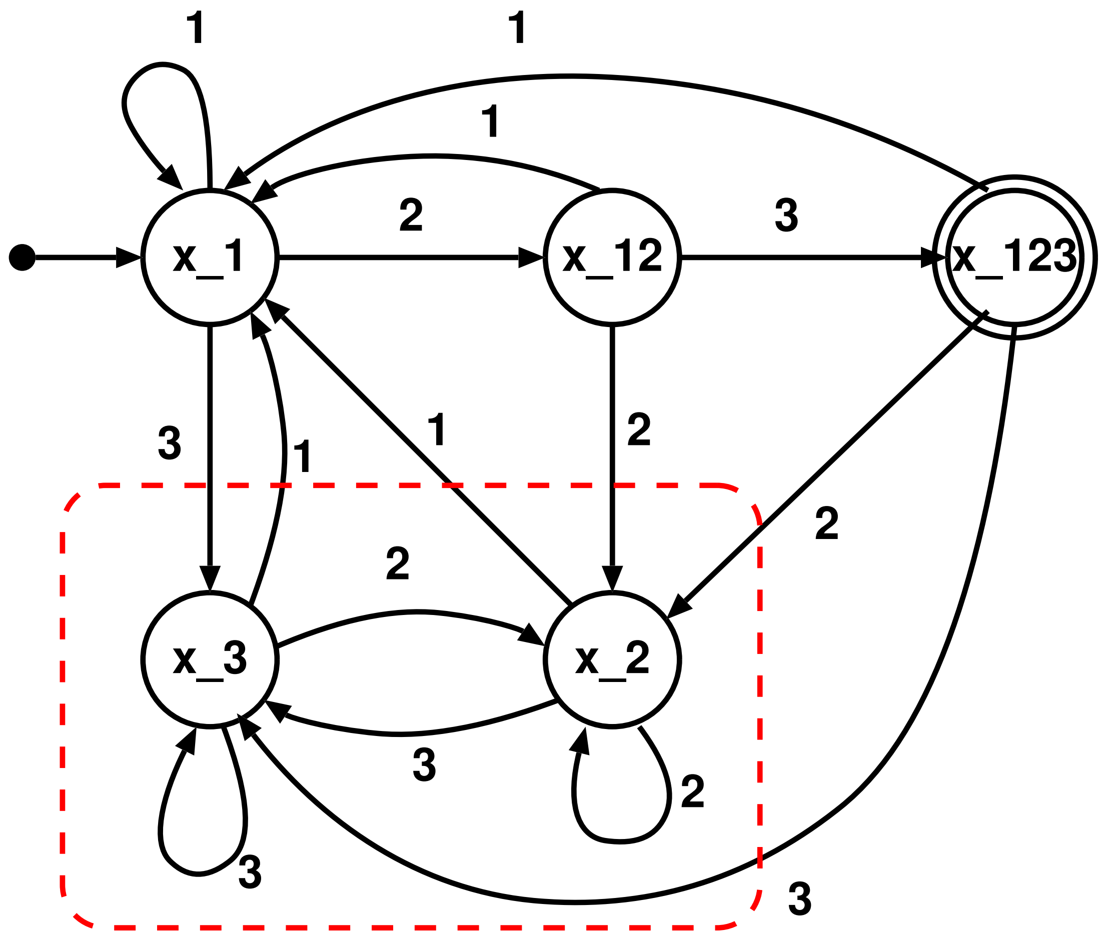
x and y are *equivalent* for as far as "accepting/rejecting" input strings is concerned.

State Aggregation Algorithm

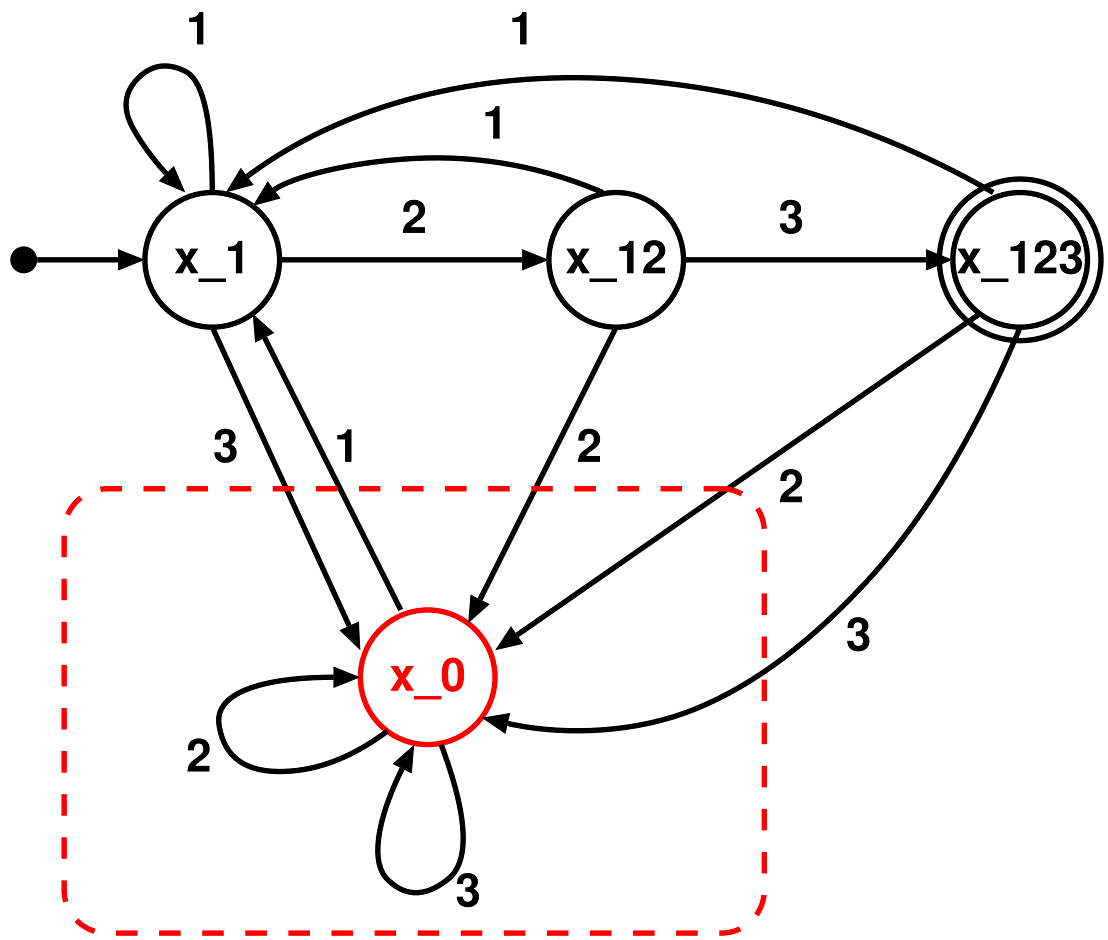
1. Mark (x,y) for all $x \in F, y \notin F$
2. For every pair (x,y) not marked in previous step:
   (a) If $(f(x,e), f(y,e))$ is marked for some $e \in E$, then:
      i. Mark (x,y)
      ii. Mark all unmarked pairs (w,z) in the list of (x,y). Repeat this step for each (w,z) until no more markings possible.
   (b) If no $(f(x,e), f(y,e))$ is marked, the for every $e \in E$:
      i. If $f(x,e) \neq f(y,e)$ then add (x,y) to the list of $f(x,e) \neq f(y,e)$

Pair which remain unmarked are in *equivalence set*

digit sequence (123) detector FSA



State Reduced FSA



State Automata to model Discrete Event Systems

- X is state space $--Q$
- All inputs are strings from an alphabet E (the events) $--X$
- State transition function $x' = f(x,e) --\delta$
- Allow X and E to be *countable* rather than finite
- Introduce *feasible events*

State Automaton

$$(E, X, \Gamma, f, x_0)$$

- E is a countable event set
- X is a countable state space
- $\Gamma(x)$ is the set of feasible or enabled events
  $x \in X, \Gamma(x) \in E$
- f is a state transition function,
  $f : X \times E \to X$, only defined for $e \in \Gamma(x)$
- x_0 is an initial state, $x_0 \in X$

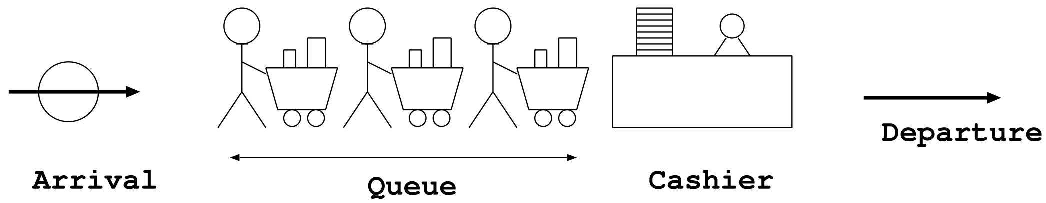
$$(E, X, \Gamma, f)$$

omits x_0 and describes a class of State Automata.

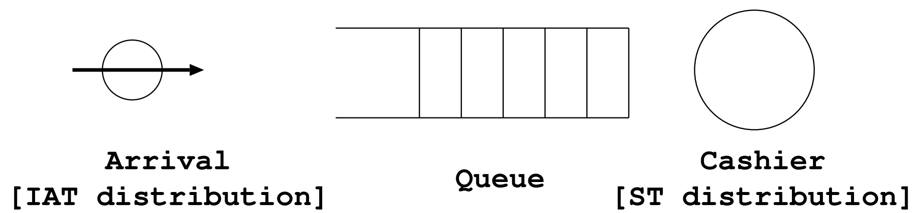
Feasible/Enabled Events

- On transition diagram: not feasibe $\Rightarrow$ not marked
- Meaning: *ignore* non-feasible events
- Why not $f(x,e) = x$ for non-feasible events ?

State Automata for Queueing Systems

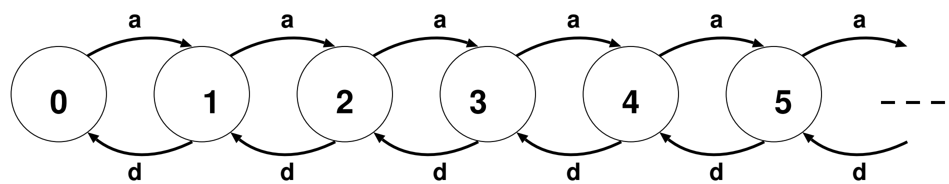


Physical View



Abstract View

State Automata for Queueing Systems: customer centered



$$E = \{a, d\}$$

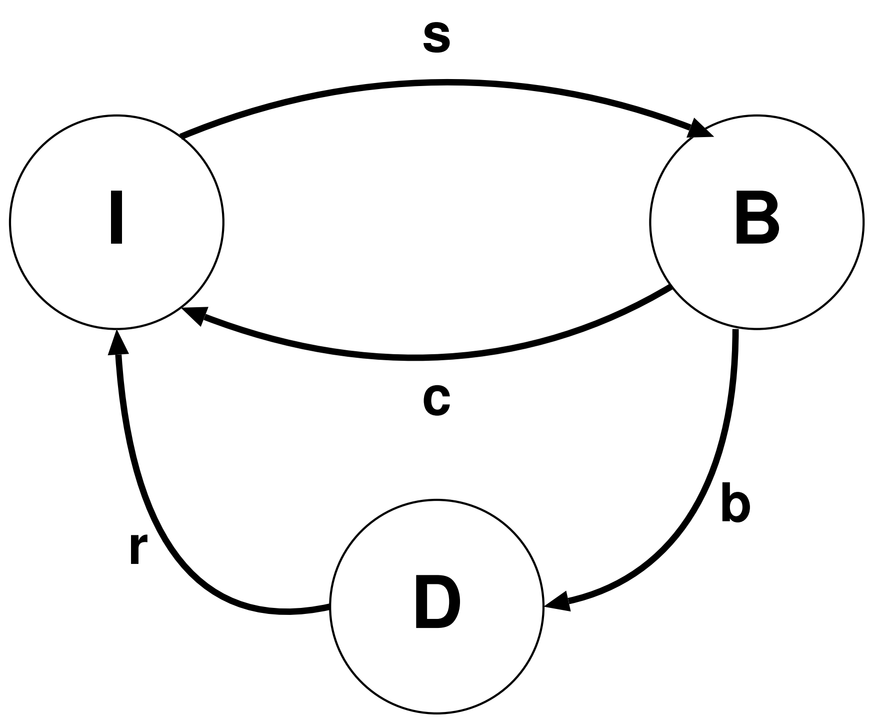
$$X = \{0, 1, 2, \ldots\}$$

$$\Gamma(x) = \{a, d\}, \forall x > 0, \Gamma(0) = \{a\}$$

$$f(x, a) = x + 1, \forall x \geq 0$$

$$f(x, d) = x - 1, \forall x > 0$$

State Automata for Queueing Systems: server centered (with breakdown)



State Automata for Queueing Systems: server centered (with breakdown)

$$E = \{s, c, b, r\}$$

Events: s denotes service starts, c denotes service completes, b denotes breakdown, r denotes repair.

$$X = \{I, B, D\}$$

State: I denotes idle, B denotes busy, D denotes broken down.

$$\Gamma(I) = \{s\}, \Gamma(B) = \{c, b\}, \Gamma(D) = \{r\}$$

$$f(I, s) = B, f(B, c) = I, f(B, b) = D, f(D, r) = I$$

Interpretations/Uses

- *Generate* all possible behaviours.
- *Accept* all allowed input sequences $\Rightarrow$ code generation.
- *Verification* of properties.

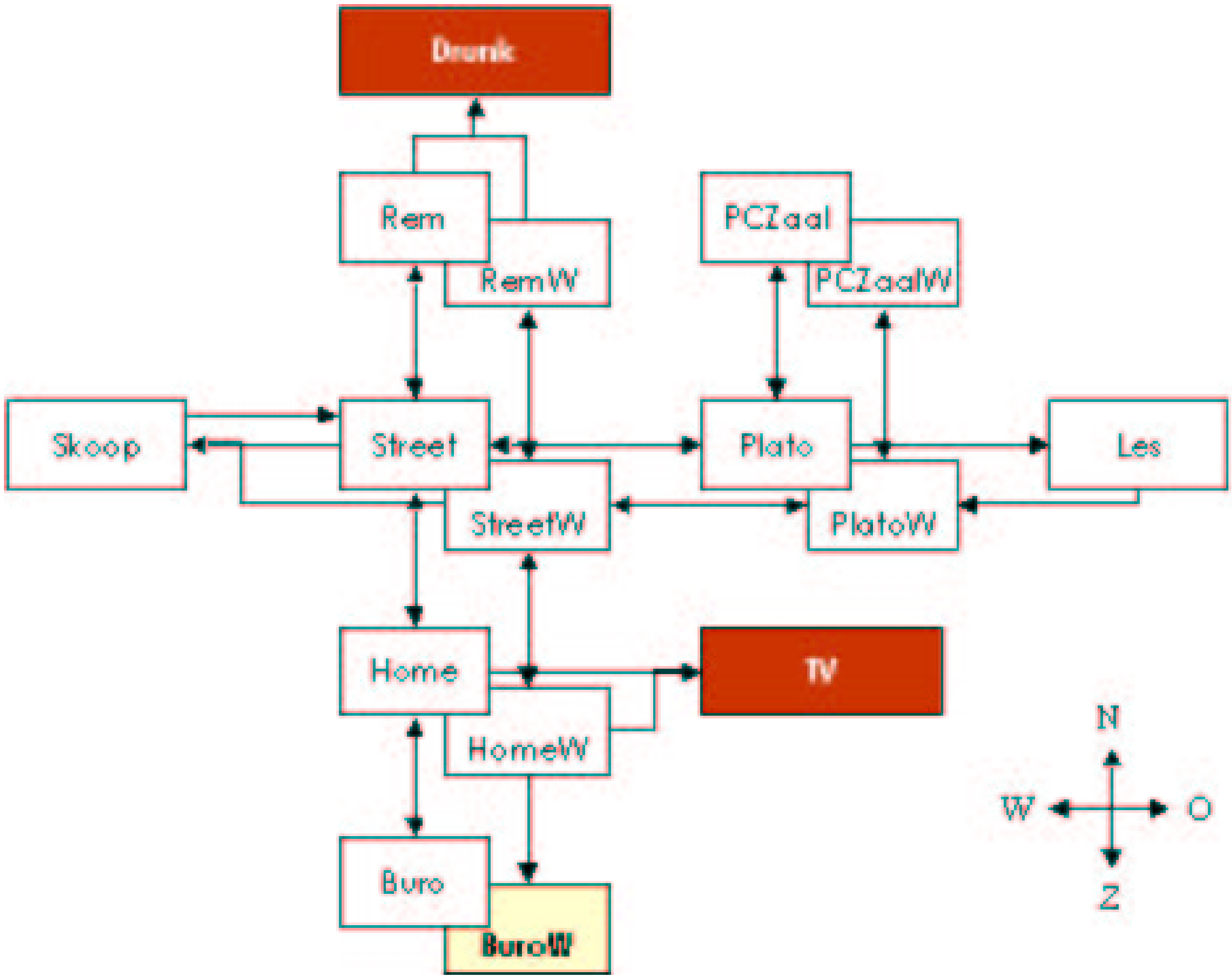
State Automata with Output

$$(E, X, \Gamma, f, x_0, Y, g)$$

- Y is a countable output set,
- g is an output function

$$g : X \times E \to Y, e \in \Gamma(x)$$

State Automata for Adventure Games



State Automata (later: Statecharts) for Graphical User Interface Specification

Limitiations/extensions of State Automata

- Adding time ?
- Hierarchical modelling ?
- Concurrency by means of $\times$
- States are represented explicitly
- Specifying control logic, synchronisation ?