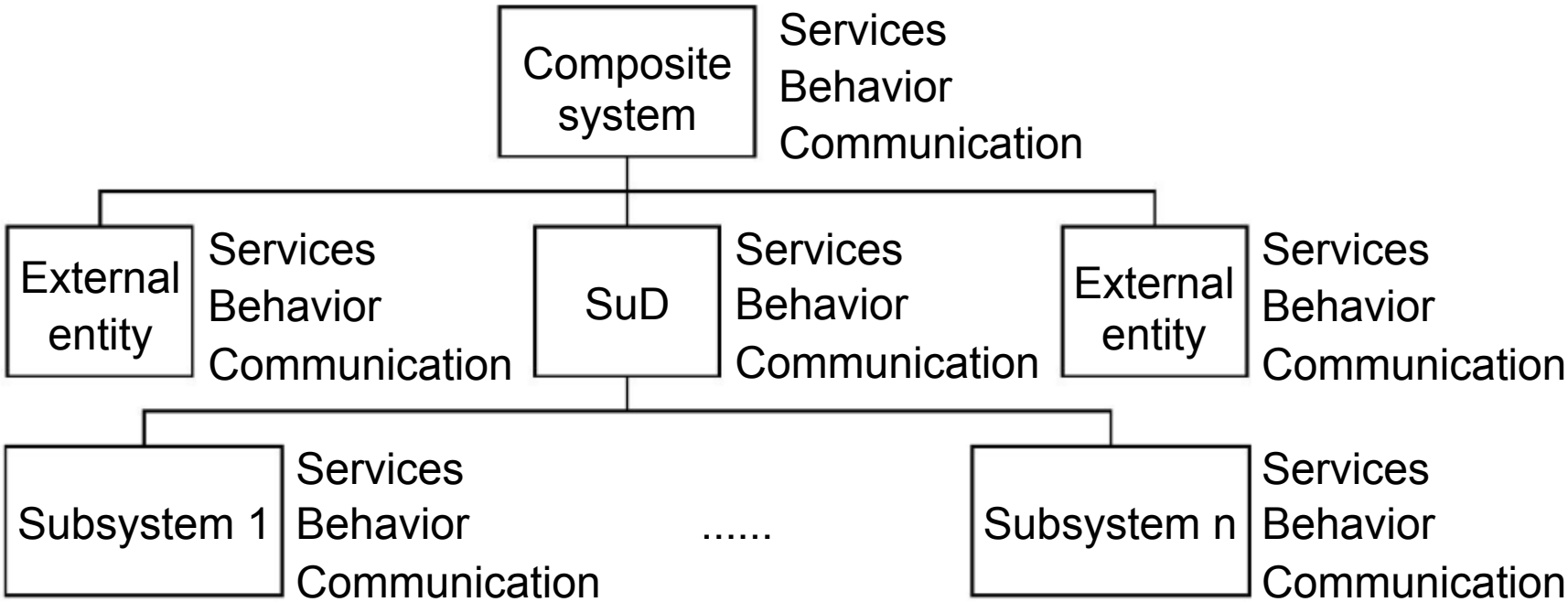
Slides for
Design Methods for Reactive Systems: Yourdon, Statemate and the UML

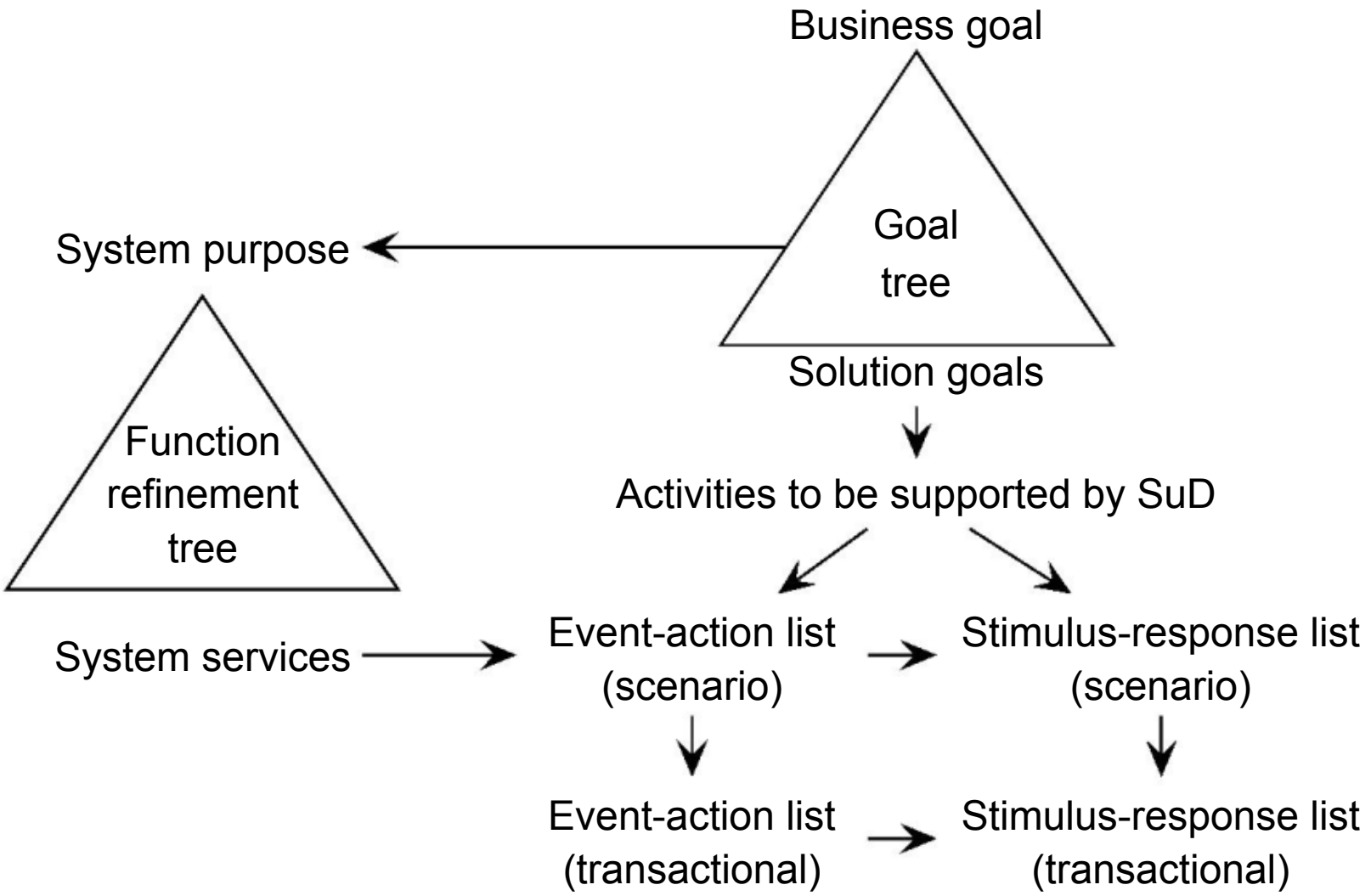
Roel Wieringa
Department of Computer Science
University of Twente,
the Netherlands
roelw@cs.utwente.nl
www.cs.utwente.nl/∼roelw

List of Slides

Part IV. Behavior Notations



Road map of ways to find behavior descriptions of the SuD



Example 1: Training Information System

We take the route through service descriptions to desired environment behavior.

Download Joiners service description

- **Triggering event:** Coordinator requests to download list of joiners from the personnel information system.
- **Delivered service:** Download the list of people from the Personnel Information System who have joined the company since the previous training.
- **Assumptions:** The data in the Personnel Information System reflects the situation accurately with a time lag of not more than one working day.

- Declarative service description! Not a scenario.

Example 1: Desired event-action pairs

Now we give behavioral details.

Environment event	Desired action
E1 Time to download list of joiners	Personnel Information System knows that list of joiners is requested.
E2 Personnel Information System sends list of joiners.	Coordinator knows that list has been downloaded.
E3 System has been waiting for list of joiners too long.	Coordinator knows that list has not been downloaded.

- The SuD is not mentioned here.
- We describe desired behavior in the itshape environment.

Example 1: Required stimulus-response behavior

Stimulus	Current system state	Response	Next system state
Receive event "download list of joiners" from coordinator.	System contains no list of joiners.	send event "send me a list of joiners" to personnel information system.	System is waiting for list of joiners.
Receive list of joiners.	System is waiting for list of joiners.	Confirm to coordinator.	System contains list of joiners.
A timer times out.	System is waiting for list of joiners.	Inform coordinator of problem.	System contains no list of joiners.

Example 1: Which steps did we take?

- From desired system services we derived desired transactional event-action pairs in environment.
  - The service descriptions describe what is valuable to the environment.
  - The event-action pairs operationalize this as desired behavior in environment.
- From event-action pairs we derive stimulus-response pairs at the interface of the system.
  - To make the SR pairs transactional, a system state is introduced, which represents part of the environment state.

Compare our road map.

Example 2: Heating controller: Behavior to be enforced.

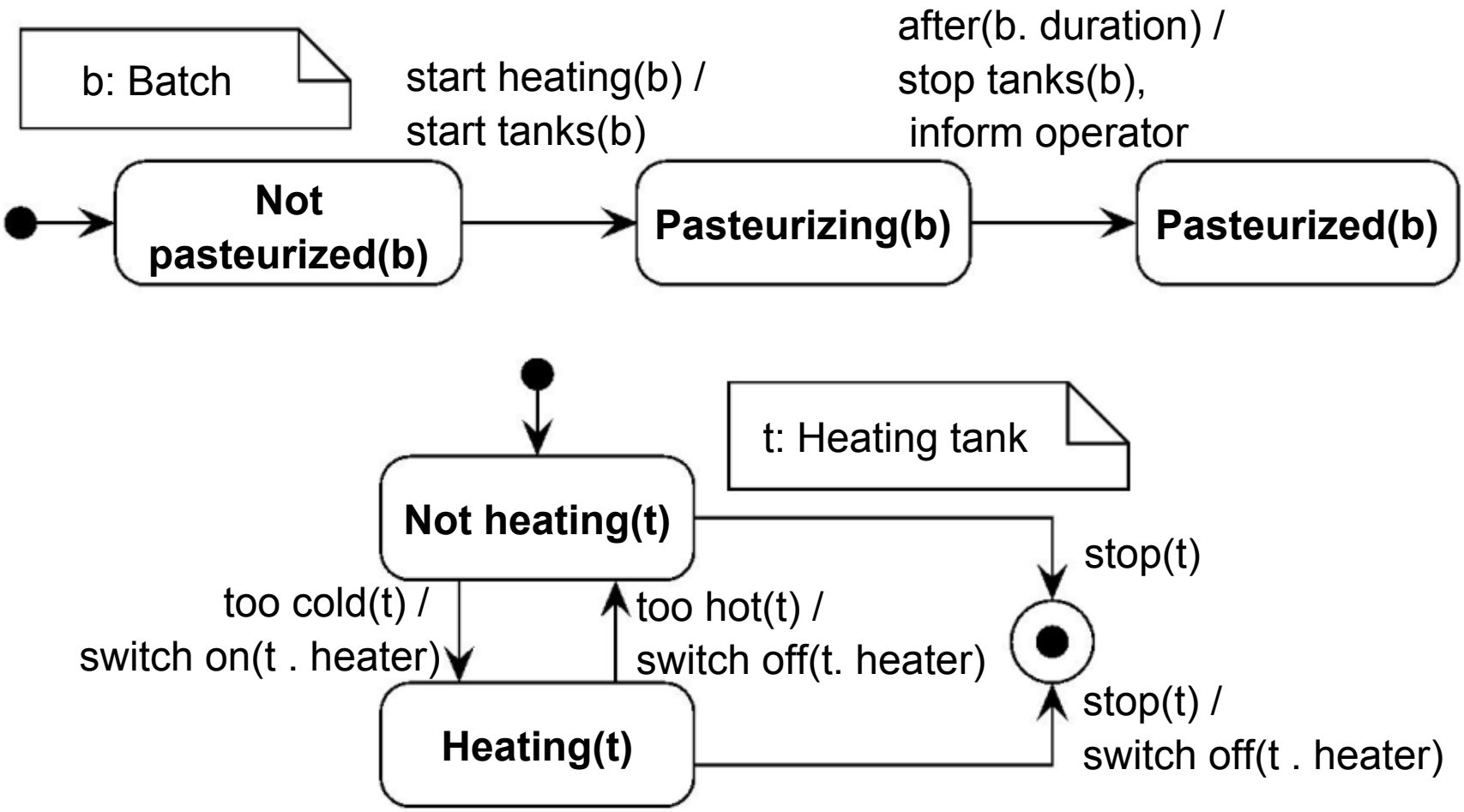
We start from a model of desired environment behavior.

Event	Desired action
Operator gives command to start heating batch b.	A heating process for the heating tanks of b is started. If at the start of the process, temperature in a tank is too low, the heater of that tank is switched on. When during the process, a tank becomes 5 degrees Celsius warmer than the desired temperature, its heater must be switched off. When it becomes 5 degrees Celsius colder than the desired temperature, its heater must be switched on. When the heating process has lasted for the duration of the recipe, heating must stop and the operator must be notified of this fact.

Example 2: Same behavior in atomic transitions

Event	Subject domain state	Desired action
E1 Operator gives command to start heating batch b		Heaters of tanks of b that are below recipe temperature, are switched on.
E2 Temperature in tank t rises 5 degrees above recipe temperature	The juice in t is being heated.	The heater of t is switched off.
E3 Temperature in tank t falls 5 degrees below recipe temperature	The juice in t is not being heated.	The heater of t is switched on.
E4 The heating duration has passed, counted since the start of heating of b.	b is being heated.	• Heaters of b that are on, are switched off. • Operator is informed.

Example 2: Same behavior described by two diagrams



Example 2: Definitions

- **start tanks(b: Batch).** For each tank t of b, signal "start(t)".
- **stop tanks(b: Batch).** For each tank t of b, signal "stop(t)".

These supplement the diagram.

Example 2: Stimulus-response behavior that enforces environment behavior

Stimulus	Current controller state	Desired response	Next state
S1 Operator gives command to start heating batch b	Not heating t and not heating b.	Switch on heaters of tanks with low temperature.	Heating t and b.
S2 Every 60 seconds.	Heating t and measured temp $>$ desired temp $+ 5$	Controller switches off the heater of t.	Heating t.
	Heating t and measured temp $<$ desired temp $- 5$	Controller switches on heater of t.	Heating t.
S4 Recipe time since the start of heating of b has passed.	Heating b.	• Switch off heaters of b that are on. • Inform operator.	Not heating b.

Example 2: Questions

The behavior brought about by the SR pairs is not exactly the same as the behavior that is desired.

- What are the differences?
- Why are these acceptable?

Example 2: Which steps did we take?

(Compare our roadmap)

- Desired environment behavior was modeled by means of non-atomic event-action pair.
- This was decomposed into atomic event-action pairs.
- From this we derived desired system behavior as a set of atomic stimulus-response pairs, that would approximately bring about desired environment behavior.

Part IV: Main points

- Behavior occurs in the environment and at all levels in the system.
- Find required system behavior by
  - Analyzing required system services or
  - modeling desired environment behavior.
- Behavior can be described in tabular format or by diagrams.

Structure of part IV

- State transition lists and tables
- State transition diagrams
- Execution semantics
- Modeling and design guidelines

Chapter 11. State Transition Lists and Tables

- Range from informal to formal
- Can be used at system level down to component or software object level.
- Different kinds of lists and tables:
  - Event list
  - Stimulus-response list
  - State transition table
  - Decision table

At system level, the lists are usually informal. At software object level, they are usually formal.

The use of behavior descriptions

Behavior descriptions can be used to represent

- Assumed environment behavior
- Desired environment behavior
- Required system behavior
- Required component behavior

You cannot tell how a behavior description is used by reading it.

Each behavior description must be supplemented by an indication of its intended use.

Event lists

List of event descriptions and, for each event, a description of its effect.

Example: description of assumed device behavior.

- **light on(b).** If b is not already in state On(b), then it enters state On(b). Otherwise, nothing changes.
- **light off(b).** If b is not already in state Off(b), then it enters state Off(b). Otherwise, nothing changes.

Example event list: desired subject domain behavior

Event	Desired action
Operator gives command to start heating batch b.	A heating process for the heating tanks of b is started. If at the start of the process, temperature in a tank is too low, the heater of that tank is switched on. When during the process, a tank becomes 5 degrees Celsius warmer than the desired temperature, its heater must be switched off. When it becomes 5 degrees Celsius colder than the desired temperature, its heater must be switched on. When the heating process has lasted for the duration of the recipe, heating must stop and the operator must be notified of this fact.

Example state transition table: Desired subject domain behavior

Event	Subject domain state	Desired action
E1 Operator gives command to start heating batch b		Heaters of tanks of b that are below recipe temperature, are switched on.
E2 Temperature in tank t rises 5 degrees above recipe temperature	The juice in t is being heated.	The heater of t is switched off.
E3 Temperature in tank t falls 5 degrees below recipe temperature	The juice in t is being heated.	The heater of t is switched on.
E4 The heating duration has passed, counted since the start of heating of b.	b is being heated.	• Heaters of b that are on, are switched off. • Operator is informed.

Effect descriptions

- **Transactional.** Description of one state transition.
  - Intermediate states abstracted away (i.e., atomic).
  - Passage of time abstracted away (instantaneous).
- **Scenario.**
  - Description of several state transitions, with intermediate states (i.e. not atomic).
  - Progress of time.

To transform a scenario description into a transactional list, we need to introduce states.

State transition tables (STTs)

List of transactional entries of the form (event, current state, actions, next state).

Variables are local to one entry. Bound in the left-hand side.

Stimulus	**Current controller state**	**Controller response**	**Next controller state**
pass doors(c)	Opened(c)		Opened(c)
	Closing(c)	open doors(c)	Opened(c)
10 seconds after the most recent execution of c.state := Opened	Opened(c)	close doors(c)	Closing(c)

Adding an initial state and initialization action

Initially		close doors(c)	Closing(c)
Stimulus	**Current controller state**	**Controller response**	**Next controller state**
pass doors(c)	Opened(c)		Opened(c)
	Closing(c)	open doors(c)	Opened(c)
10 seconds after the most recent execution of c.state := Opened	Opened(c)	close doors(c)	Closing(c)

Transformation table: no state change

Event	Current state	Action
arrive(b, c)	Destination request(b, c) and Atfloor(b, c)	• stop motor(b, c) • open doors(c) • light off(b)
	Forward request(b, c) and Atfloor(b, c)	• stop motor(b, c) • open doors(c) • light off(b) • show direction(c)
	Outermost reverse request(b, c) and Atfloor(b, c)	• stop motor(b, c) • open doors(c) • light off(b) • reverse and show direction(c)

Transformation table = Decision table

bd, bf, br: Request button

c: Elevator cage

Destination request(bd, c) and Atfloor(bd, c)	T	F	T	F	T
Forward request(bf, c) and Atfloor(bf, c)	F	T	T	F	F
Outermost reverse request(br, c) and Atfloor(br, c)	F	-	-	T	T
stop motor(c)	X	X	X	X	X
open doors(c)	X	X	X	X	X
show direction(c)	X	X	X		
reverse and show direction(c)				X	X
light off(bd)	X		X		X
light off(bf)		X	X		
light off(br)				X	X

An STT with next states but without actions

Current state	button	Event	Next button state
On(b)		light off(b)	Off(b)
		light on(b)	On(b)
Off(b)		light on(b)	On(b)
		light off(b)	Off(b)

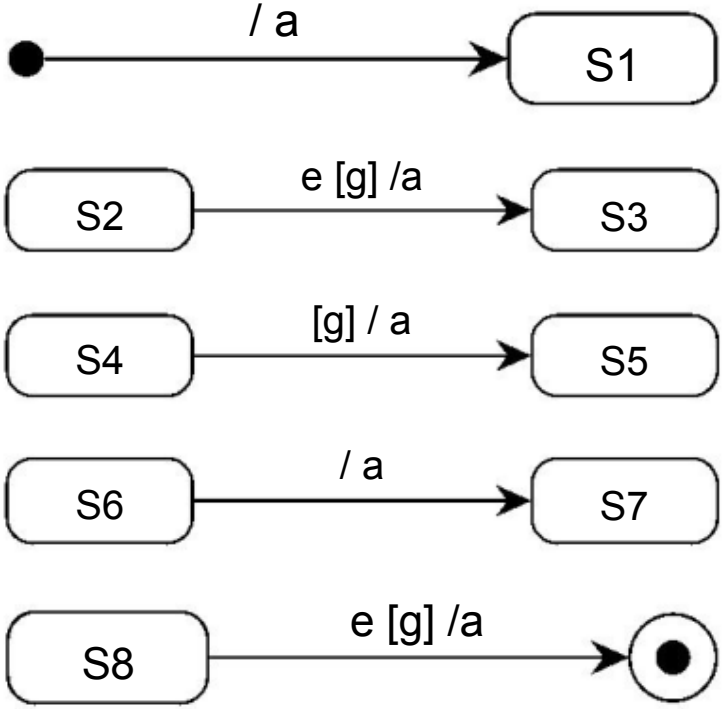
Main points

- Tabular behavior descriptions range from informal (event list) to formal (state transition table).
- Can be used for assumed or desired behavior of environment or system.
- List entry can describe a transition or a scenario.
- Variables can be declared for a table; binding is local to one entry.
- Stateless "transition" is really decision rule.

Chapter 12. State Transition Diagrams

- Good at showing the structure of behavior.
- Not good at showing unstructured behavior.
- Can only show a limite amount of information because restricted to one sheet of paper.
- Tables and lists on the other hand can show unlimited amounts of information, because they can continue on any number of sheets.

Mealy diagram constructs



Initialization actions a, initial state S1.

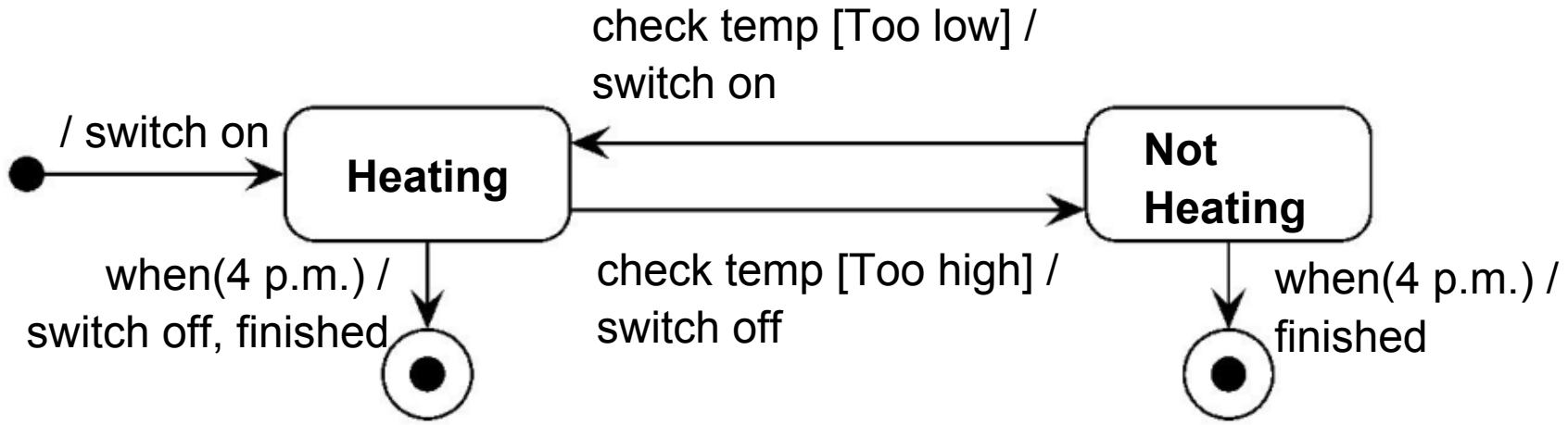
Trigger e, guard g, actions a.

Take transition when g becomes true, or immediately.

Take transition immediately.

Transition to final state.

Example

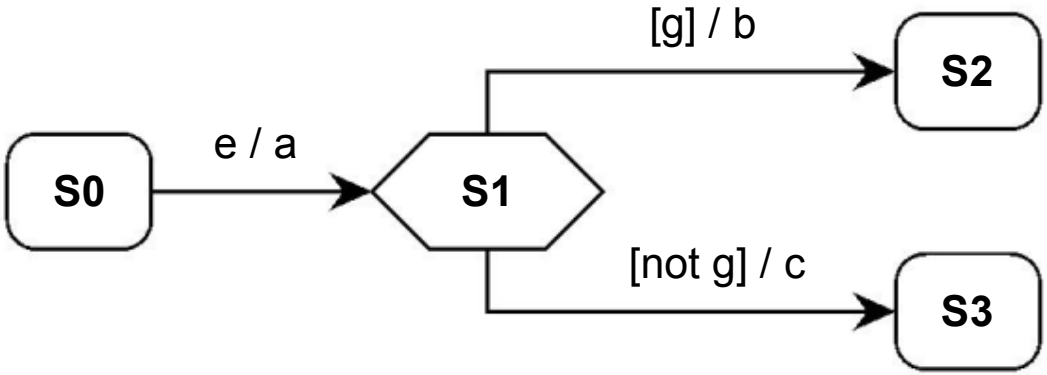

check temp [Too low] /
switch on
/ switch on
Heating
Not
Heating
when(4 p.m.) /
switch off, finished
check temp [Too high] /
switch off
when(4 p.m.) /
finished

Events

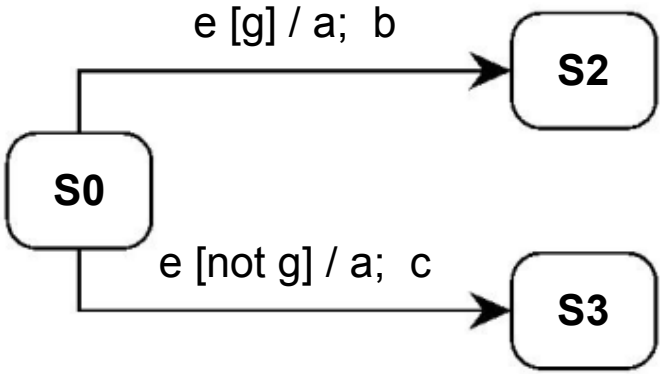
An **event** is a discrete change in the condition of the world.

- **Named events.** We gave a name to it.
- **Condition change event.** A Boolean condition g becomes true.
- **Temporal event.** Significant moment in time. Relative temporal event: after(t), absolute temporal event: when(t).

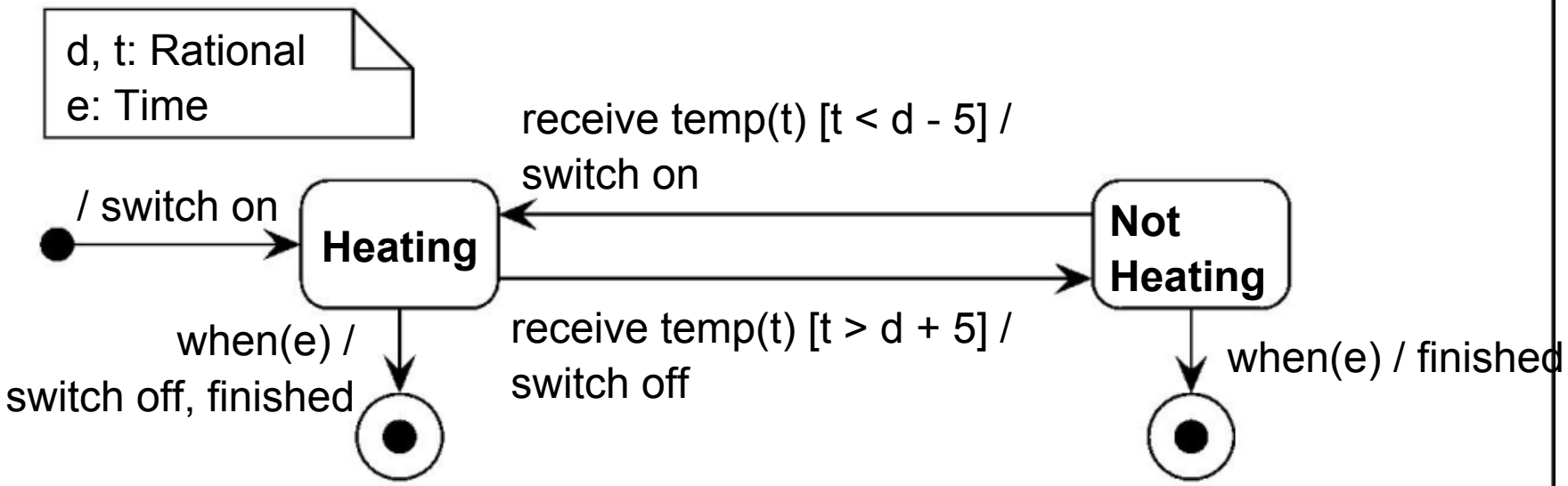
Decision states



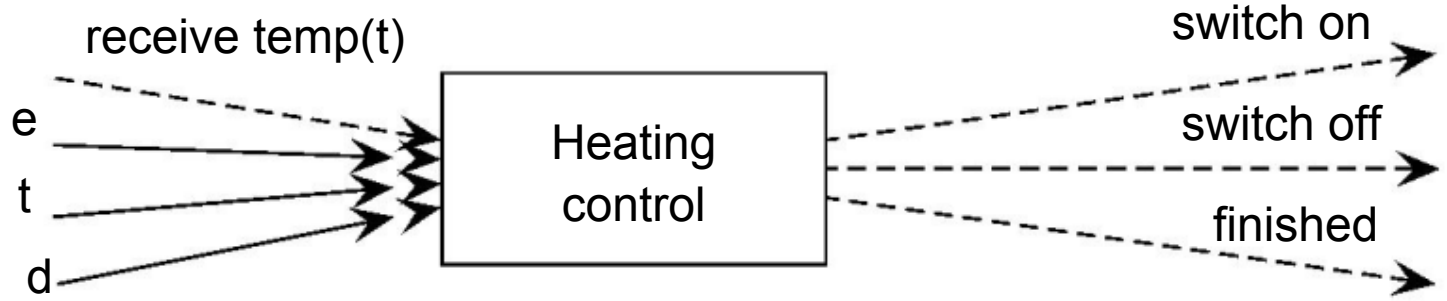
If g is not affected by a, this can be replaced by:



Variables

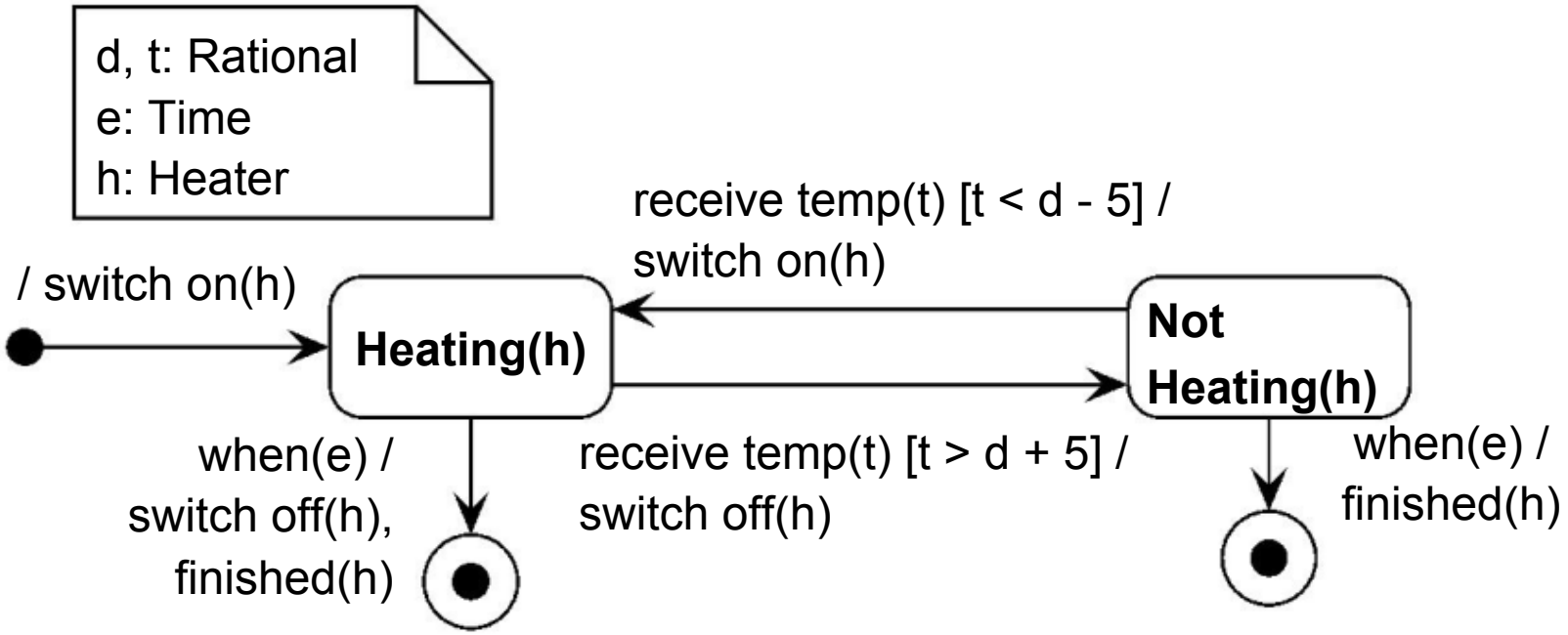


Interface of this machine:



Locality

Variables are local to a transition, except the identifier variable of a diagram.

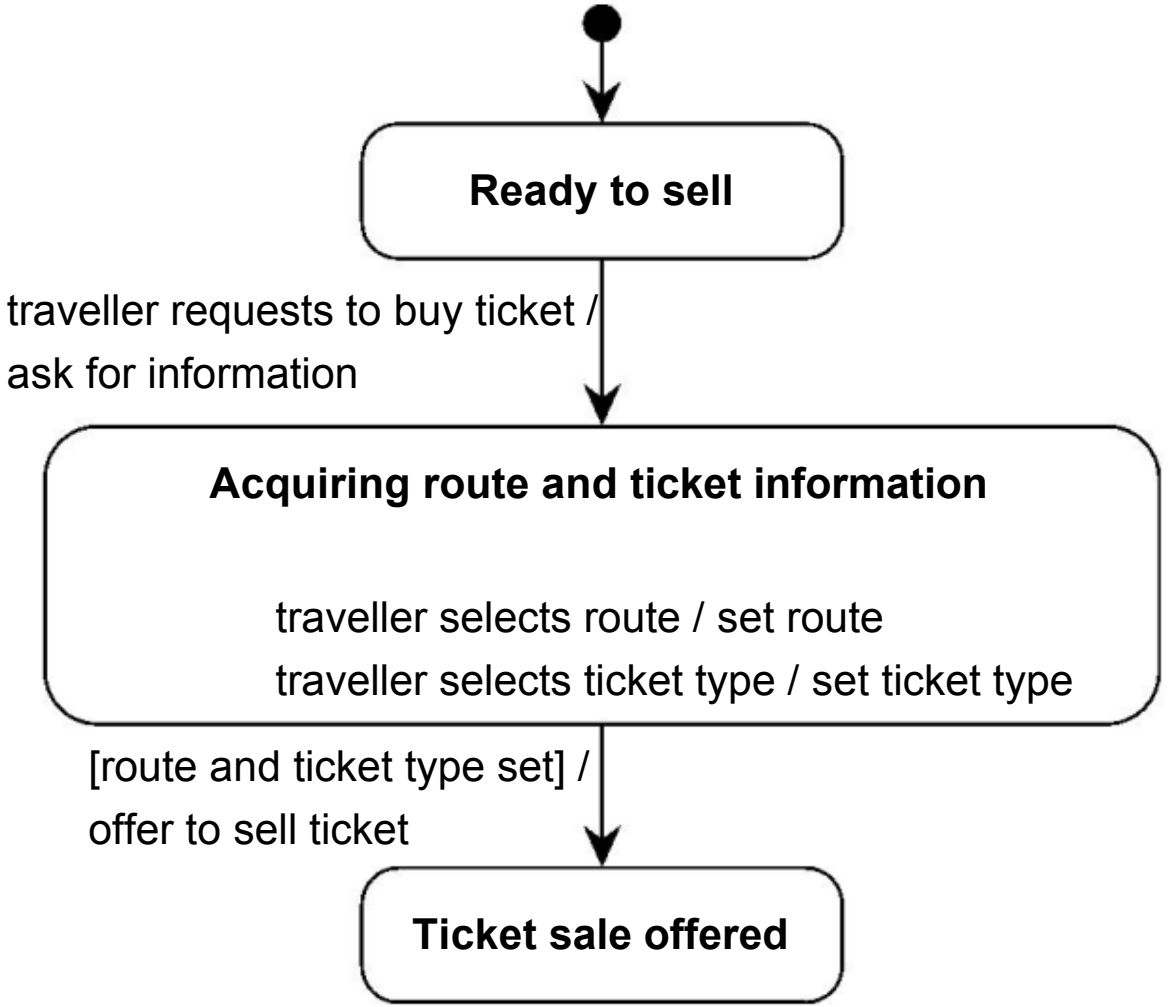


Statecharts

Mealy diagrams plus

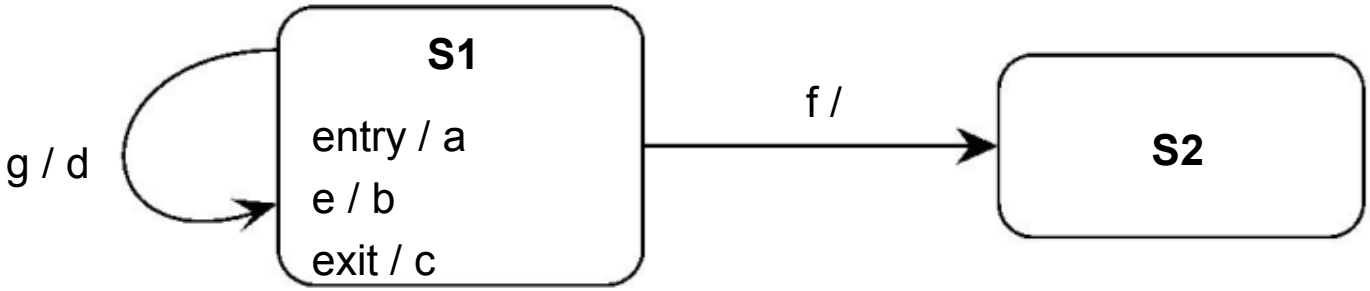
- State reactions
- State hierarchy
- Parallelism

State reactions



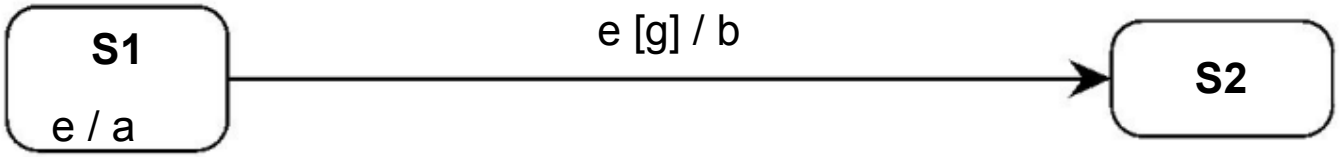
Saves space in diagram.

Example state reactions



What happens when g occurs?

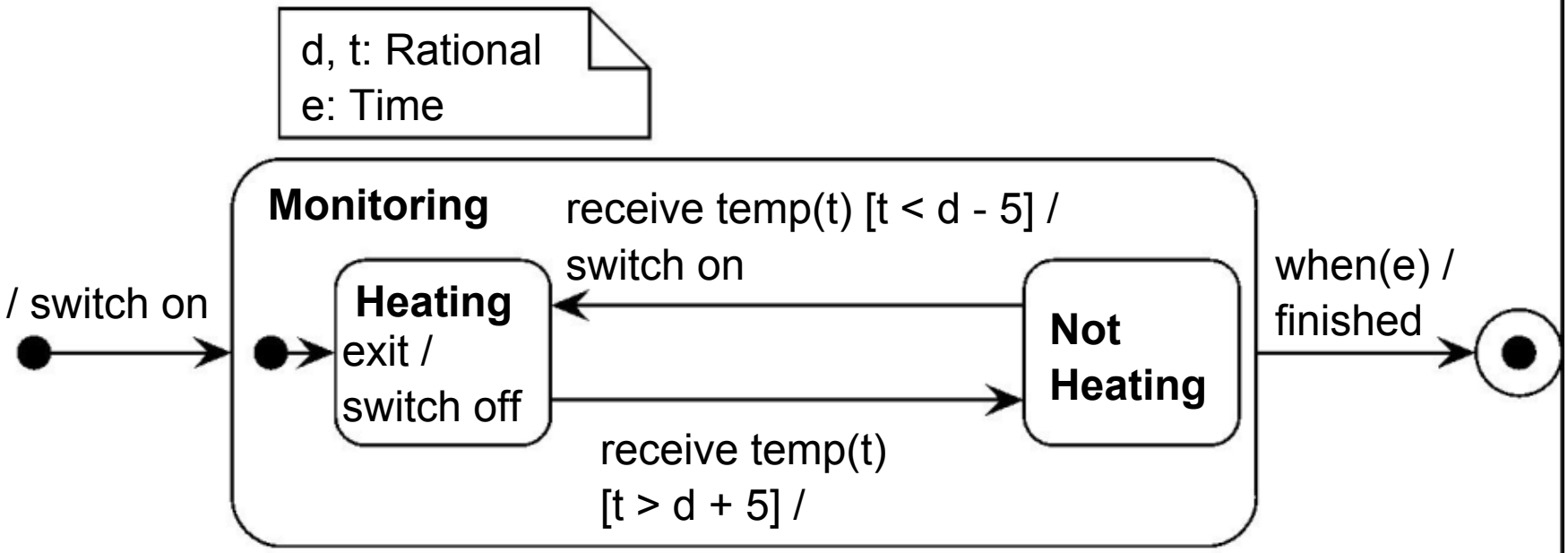
Conflict between state reaction and transition



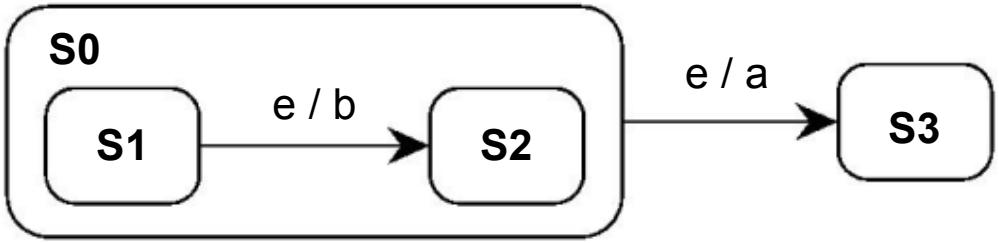
What happens when e occurs and g is true?

- Depends upon the semantics given to the notation by the analyst.
- All readers and authors of the diagram should use the same semantics!

Hierarchy


d, t: Rational
e: Time
Monitoring
receive temp(t) [t < d - 5] /
switch on
/ switch on
Heating
exit /
switch off
Not
Heating
when(e) /
finished
receive temp(t)
[t > d + 5] /

Conflicts between transitions at different levels

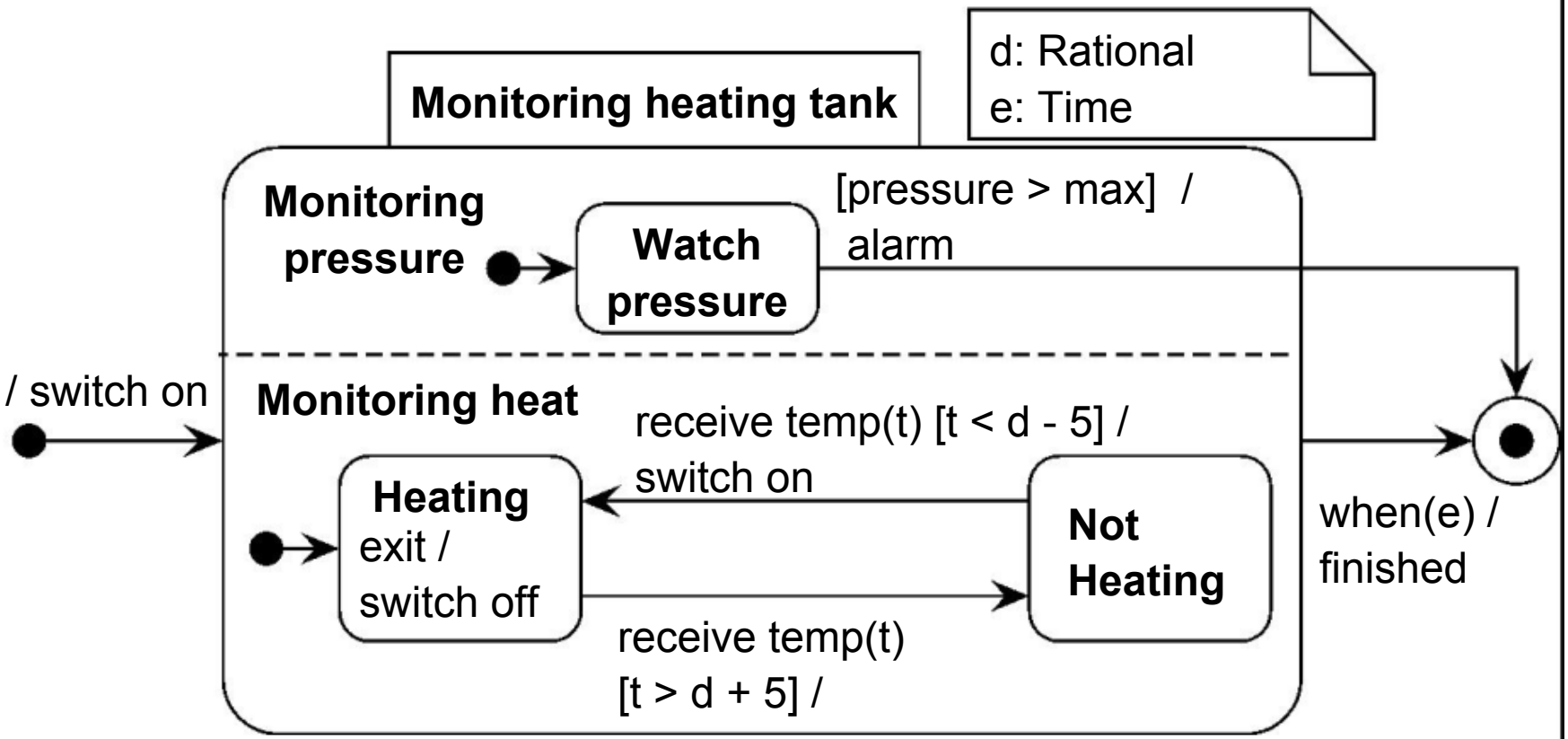


What happens when e occurs?

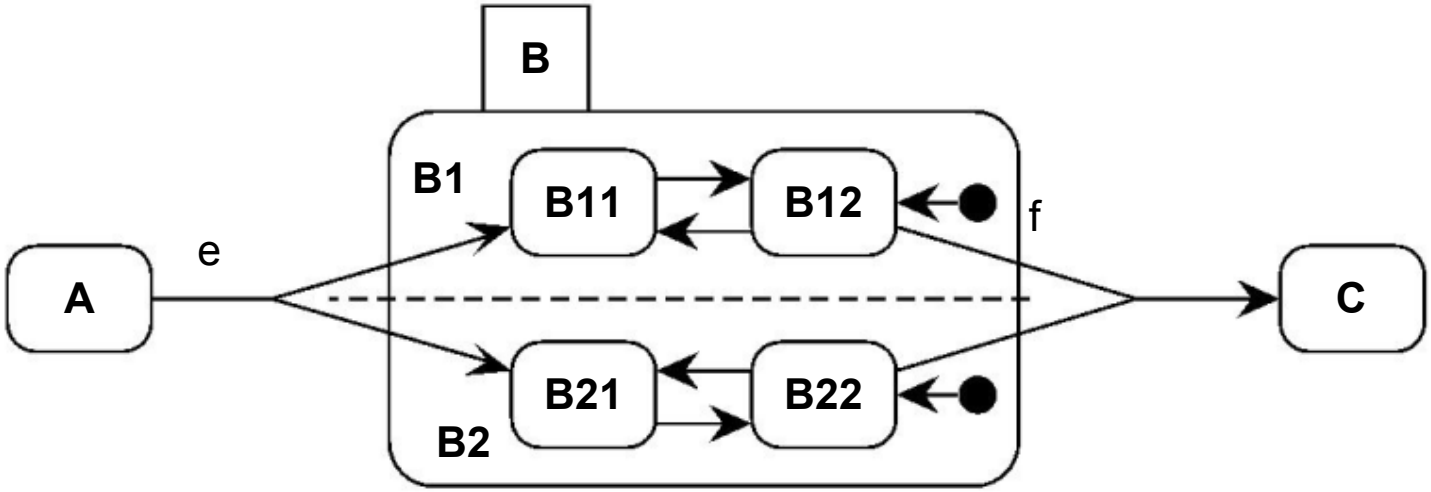
- UML: The lower transition is taken.
- Statemate: The higher transition is taken.

UML views lower-level behavior as specialization. Statemate sees higher-level behavior as more important, used e.g. to describe interrupts and error-handling.

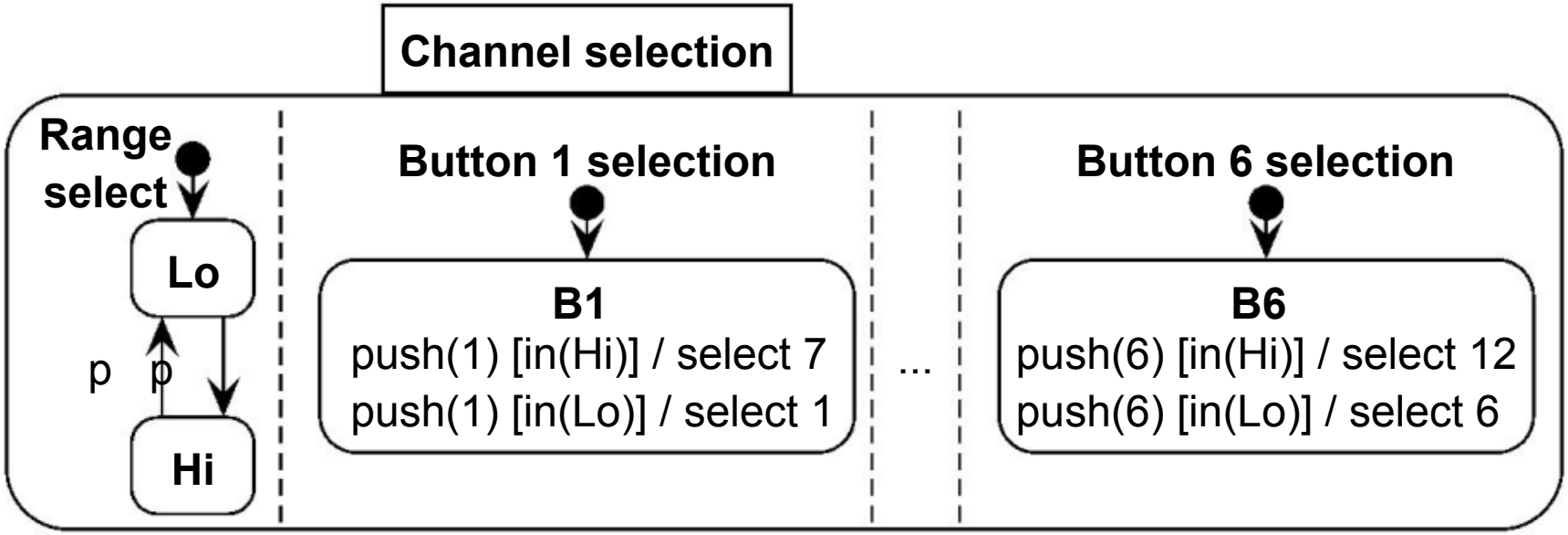
Parallelism



Hyperedges

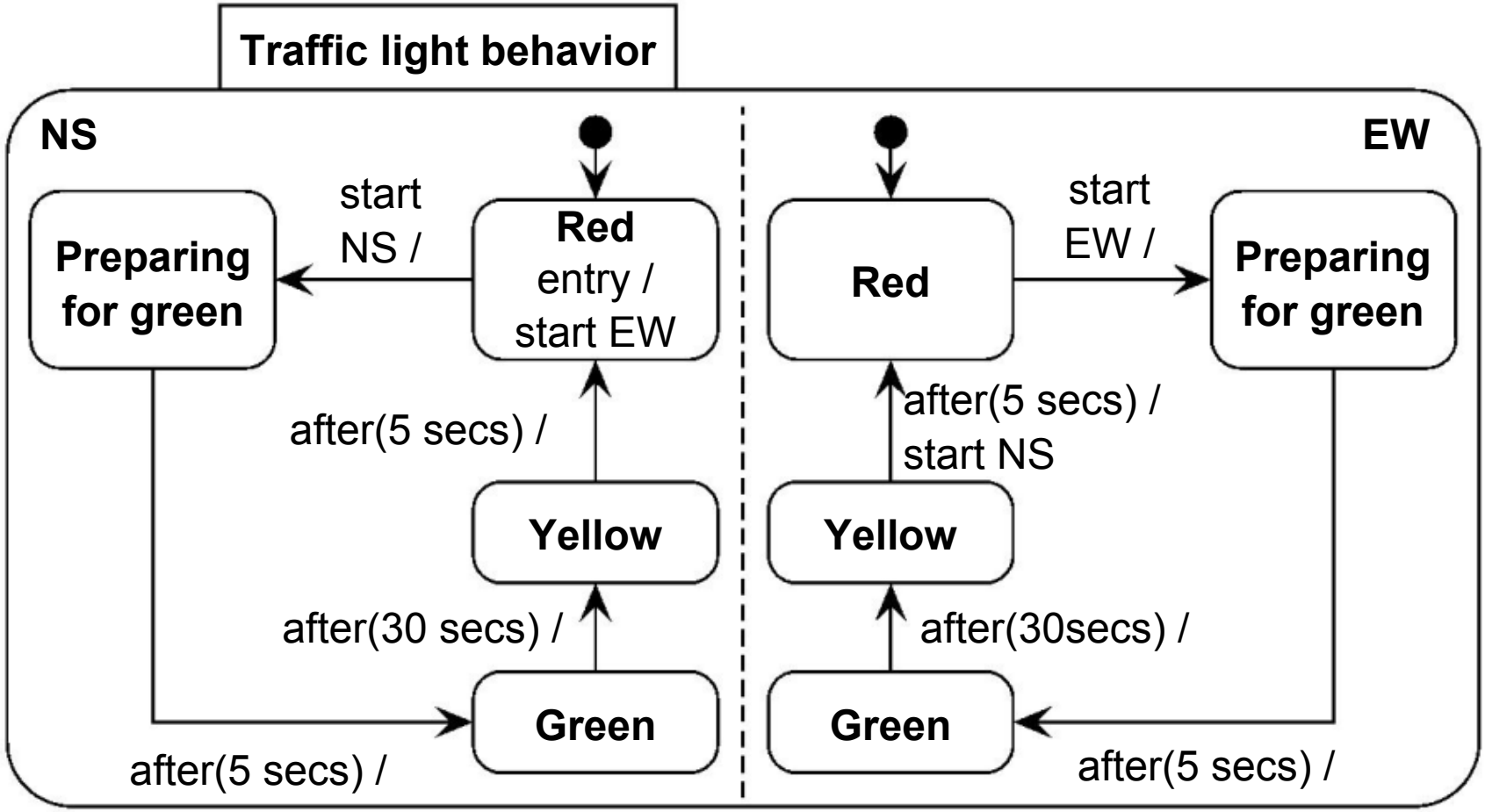


Breaking orthogonality by the in(State) predicate



Now the behavior in one state depends upon the state of some parallel component.

Breaking orthogonality by event broadcasting

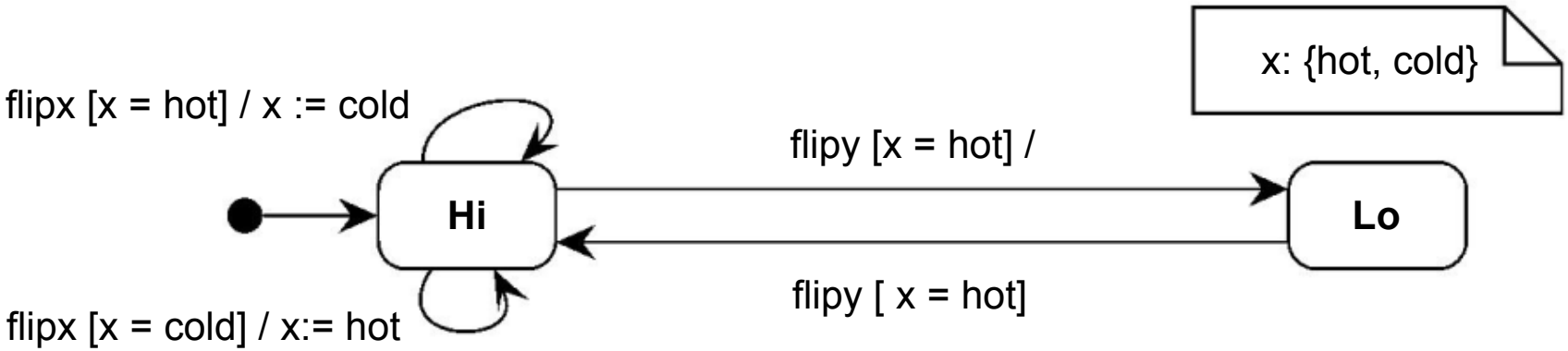


Now the behavior of one state depends upon events generated in some parallel component.

Main points

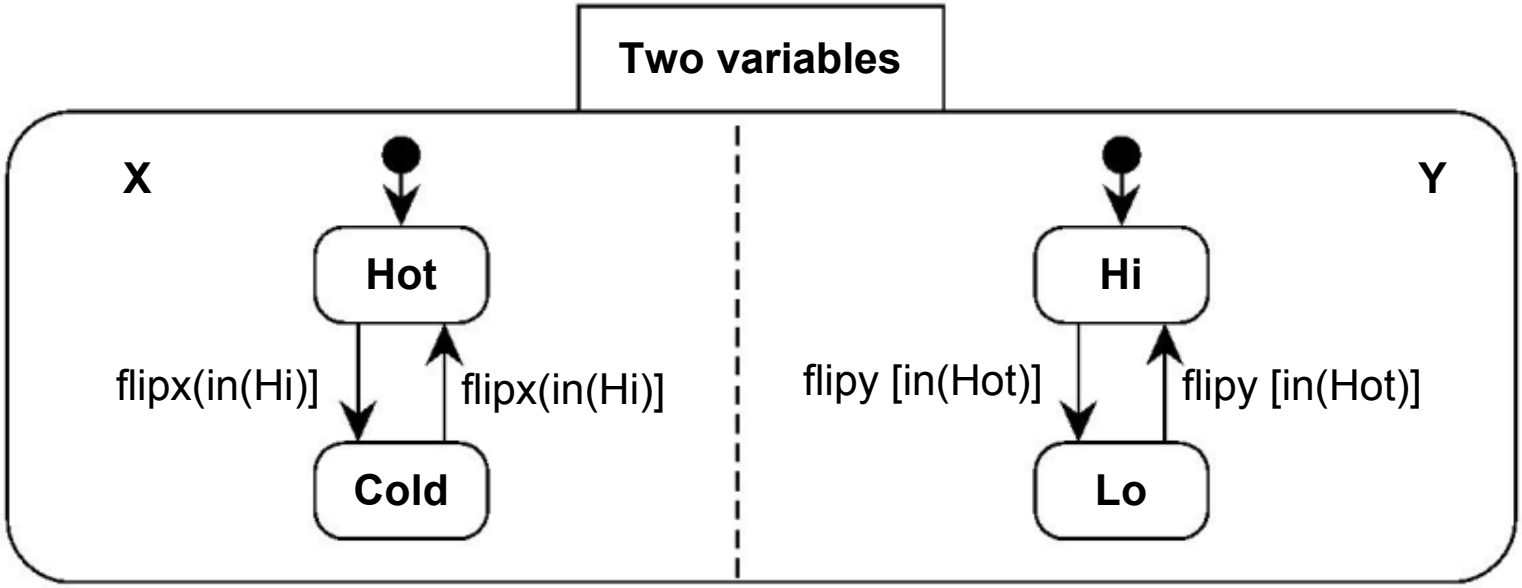
- STDs graphically represent reactive behavior, which consists of a discrete state space and `e [g] / a` transitions.
- Events can be named, condition change, or temporal.
- Decision states are unstable.
- STDs can have local variables. Only the identifier variable is global to the diagram.
- Statecharts extend Mealy diagrams with state reactions, state hierarchy, parallelism.
- The semantics of a diagram must be defined explicitly (see chapter 13).

Questions



- Eliminate the variables as follows: Define two parallel components, one component for each variable.
- Is the resulting behavior equivalent to this one?

Solution

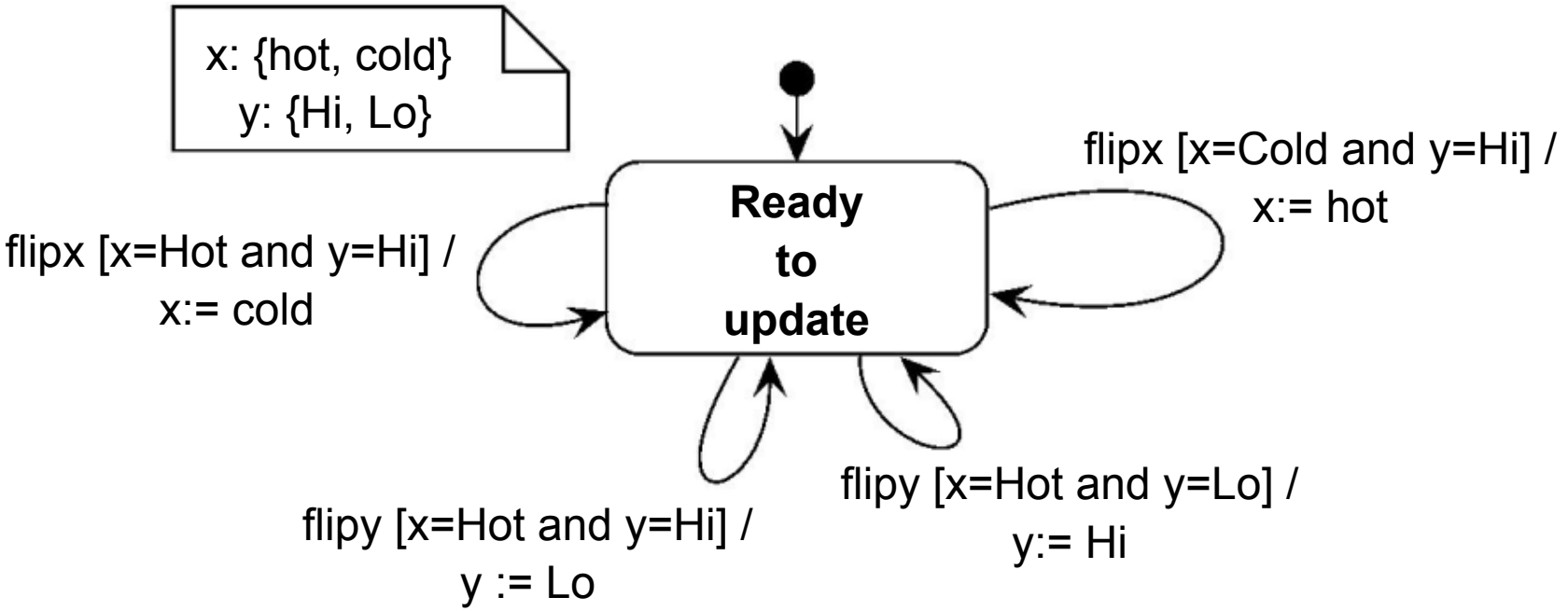


The important semantic difference between this statechart and the Mealy diagram is that the statechart can respond to a flipx and a flipy event at the same time.

Question

- Encode all state information in variables. The result should be an STD with one state only.
- Is the result equivalent to the original STD?

Solution



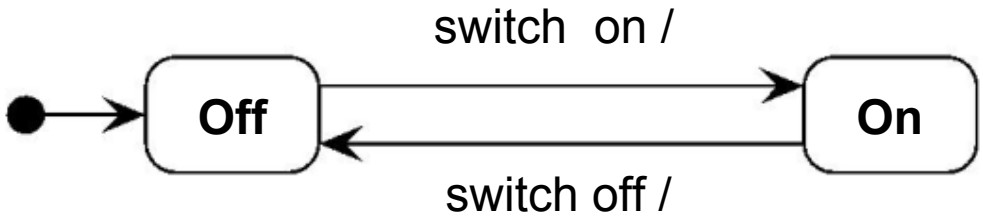
This diagram does not indicate the initial value of the variables.

Chapter 13. Behavioral Semantics

- STTs and STDs are descriptions of behavior.
- But they can be interpreted in many ways! That is a problem:
  - Designers, builders and users think they agree ...
  - ... but nevertheless expect different behavior to be implemented.
  - To execute a specification, it must be unambiguous.
  - To generate code, different code generation tools must use the same semantics.

In this chapter we review the relevant semantic choices.

Discretization

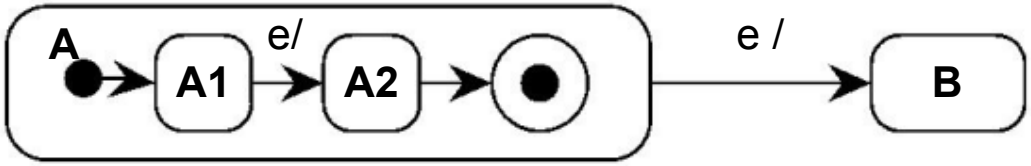


STDs and STTs are abstract, and therefore necessarily incomplete, descriptions of continuous behavior.

- **Atomicity.** Ignore intermediate states of each transition.
- **Possibility.** There are no slightly different events or states. switch on is possible when Off etc.
- **Isolation.** Whatever else happens, switch on in state Off leads to state On, etc.
- **Durability.** A state is stable until an event occurs that triggers an outgoing transition.

Wait States and Activity States: Semantic options

An activity state has internal activity, a wait state has not.



This creates the possibility for conflict: What happens when `e` occurs? Options:

- **Deferred response.**
- **Forced termination.**
  - Higher level has priority (Statemate).
  - Lower level has priority (UML).

Final states: Semantic options

Activity states create ambiguity for final state node. What happens when final state is reached?

- **Global termination:** bull's eye indicates that overall process terminates (Statemate).
- **Local termination:** Activity terminates; wait until `e` occurs (UML).

Pre- and postconditions: Problems

	Current state	Event	Action	Next state
1	P(x)	e(x, y)	a(x, y)	Q(x)

- Does P(x) become false?
- Should Q(x) be false in initial state?
- We will treat state descriptions in an STT as **preconditions**

  and **postconditions,** that constrain the current and next state. So:
  - Q(x) can have any value in initial state.
  - P(x) can have any value in next state.

  If you don't want this, write down truth value explicitly.

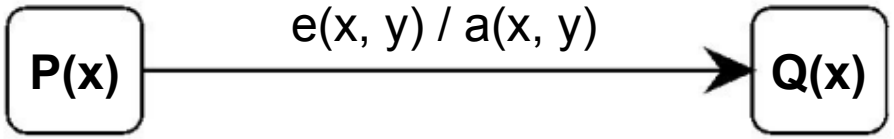
Pre– and postconditions: Possible solutions

	Current state	Event	Action	Next state
1	P(x)	e(x, y)	a(x, y)	Q(x)
2	P(x)	f(x, y)	a(x, y), b(y)	R(x)

- What happens with R(x) in transition 1? Not specified.
- If we must write down truth values explicitly, table entries become very large.
- **Derivation rules.** E.g. $\mathsf{Q}(x) \rightarrow \mathsf{R}(x)$. Part of domain knowledge or SuD spec.
- **Frame rule.** If the transition rule and derivation rules do not imply that P changes, then it does not change.

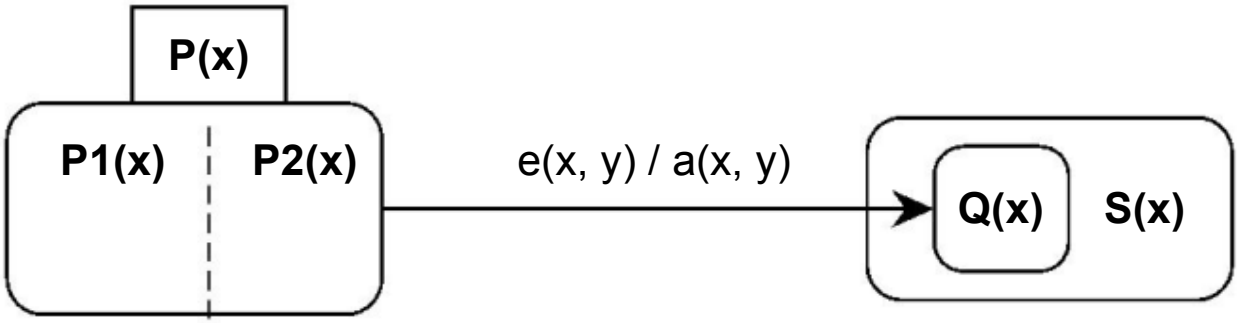
Pre– and postconditions in STTs and STDs

	Current state	Event	Action	Next state
1	P(x)	e(x, y)	a(x, y)	Q(x)



- Equivalent?
- No: STDs come with the interpretation rule that $S_i \leftrightarrow \neg S_j$ for $i \neq j$.
- So P(x) $\leftrightarrow \neg$ Q(x) in the STD!

Interpretation rules for statecharts

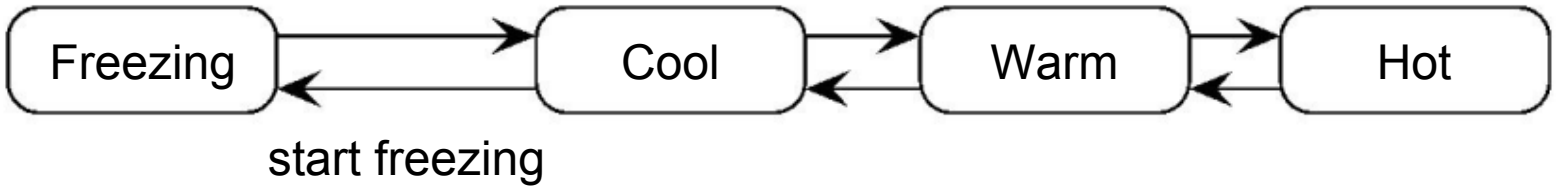


Use the following interpretation rules for statecharts:

- For all x, P(x) $\leftrightarrow$ not S(x).
- For all x, Q(x) $\rightarrow$ S(x).
- For all x, P1(x) $\rightarrow$ P(x).
- For all x, P2(x) $\rightarrow$ P(x).

Triggering (1)

The first problem with triggering is how we interpret what the diagram does *not* say.



Can start freezing occur in state Hot?

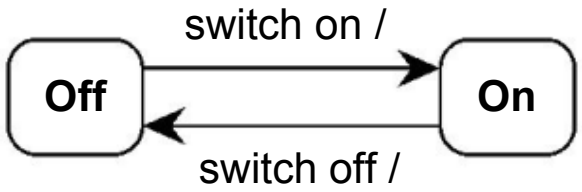
- **Physically impossible:** Contradicts the laws of nature.

If this is what we mean, we should add this interpretation rule to the diagram.

Triggering (2)

Definition:

- **switch on** is the event that leads the system from Off to On.



What does this mean? Can switch on occur in state On?

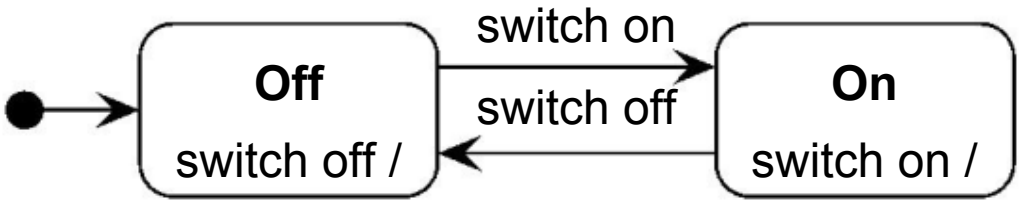
- **Logically impossible:** Contradicts the definition in the dictionary. Whatever happens in state On is not the switch on event.
- **Logically possible:** switch on is a certain physical event that can occur in many states. Update the dictionary with this.

Whatever we mean, add this to the diagram.

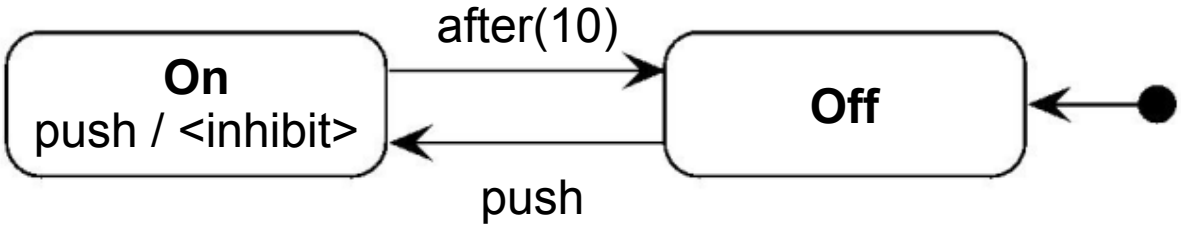
Triggering (3)

What happens if switch on *does* occur in state On?

- **Ignore:** Respond as if the event did not occur.

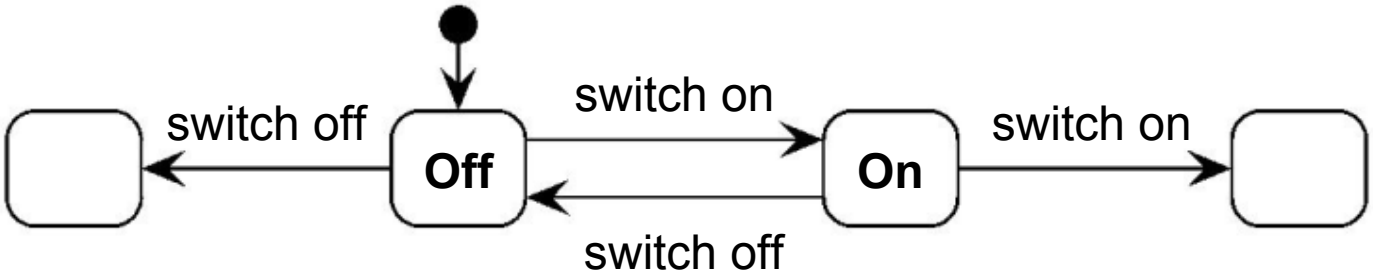


- **Inhibit:** Make the event physically impossible.



Triggering (4)

- **Unknown effect:** The system breaks down. This is the fragile semantics.



Make clear whether you agree on the ignore, inhibit, or fragile semantics as default interpretation for the STDs and STTs in a specification.

Triggering multiple transitions: The problem

	Current state	Event	Action	Next state
1	P(x)	e(x, y)	a(x, y)	Q(x)
4	R(y)	e(y, z)	b(y)	R(z)
5	R(x)	e(x, y)	c(x)	P(x)and not Q(x)

Suppose e(2, 4) occurs. Entries are instantiated as:

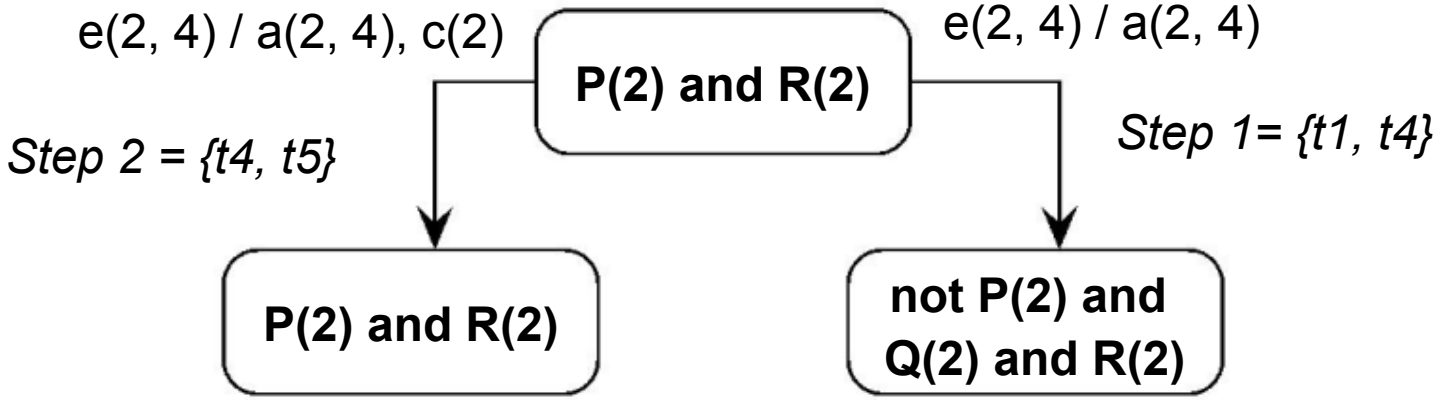
t1	P(2)	e(2, 4)	a(2, 4)	Q(2)
t4	R(2)	e(2, 4)	b(4)	R(4)
t5	R(2)	e(2, 4)	c(2)	P(2) and not Q(2)

Suppose P(2) and R(2) are true. What happens?

Triggering multiple transitions: Possible solutions

- **Step semantics.** Execute as many transitions as possible in one step. (Statemate and UML multi-threaded semantics)

Two possible steps:



- **Single-transition semantics.** Choose one transition non-deterministically. Forget about the others. (UML single-threaded semantics)

Responding to multiple events: The problem and possible solutions

Suppose e(3, 4) and f(3, 4) occur at the same time. What happens?

t1	P(3)	e(3, 4)	a(3, 4)	Q(3)
t2	P(3)	f(3, 4)	a(3, 4)	R(3)

- **Concurrent-event semantics.** Respond to all events not yet responded to. (Statemate)
- **Sequential-event semantics.** Respond to events in some sequence. (UML)

Multistep semantics: The problem

One transition can trigger further transitions. E.g. suppose f(3, 4) happens in state P(3):

t2	P(3)	f(3, 4)	a(3, 4), b(4)	R(3)

Action a(3, 4) in turn triggers an instance of entry 3:

t3	R(3)	a(3, 4)	b(3)	P(3) and not R(3)

What happens?

Multistep semantics: Possible solutions

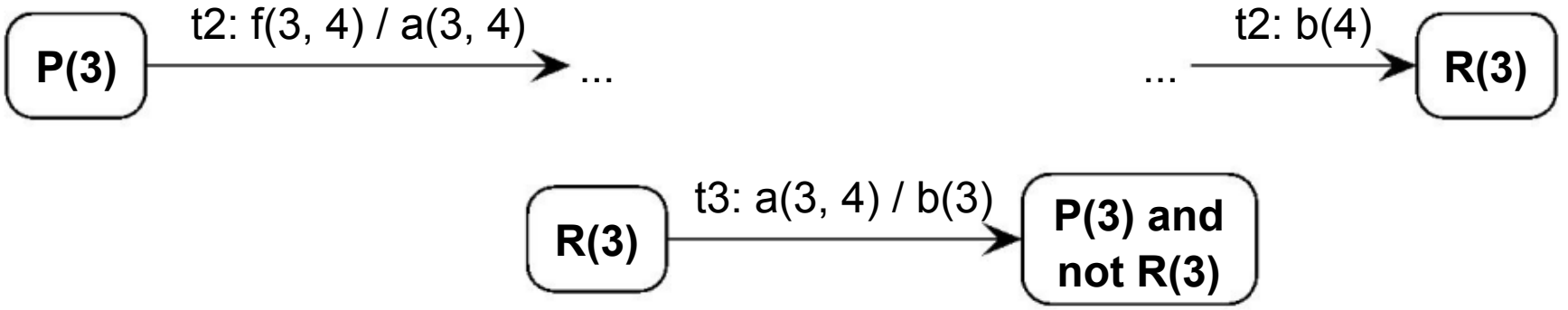
- **Single step semantics.** After each step, the behavior can respond to external as well as internally generated events. (Statemate option)
- **Superstep semantics.** The behavior first performs the complete multistep response before it is able to respond to external events. (Statemate option)
- **Delayed step semantics.** The behavior responds to internal and external events in some later step. (UML)

Action semantics

How is a complex action such as a, b, c executed?

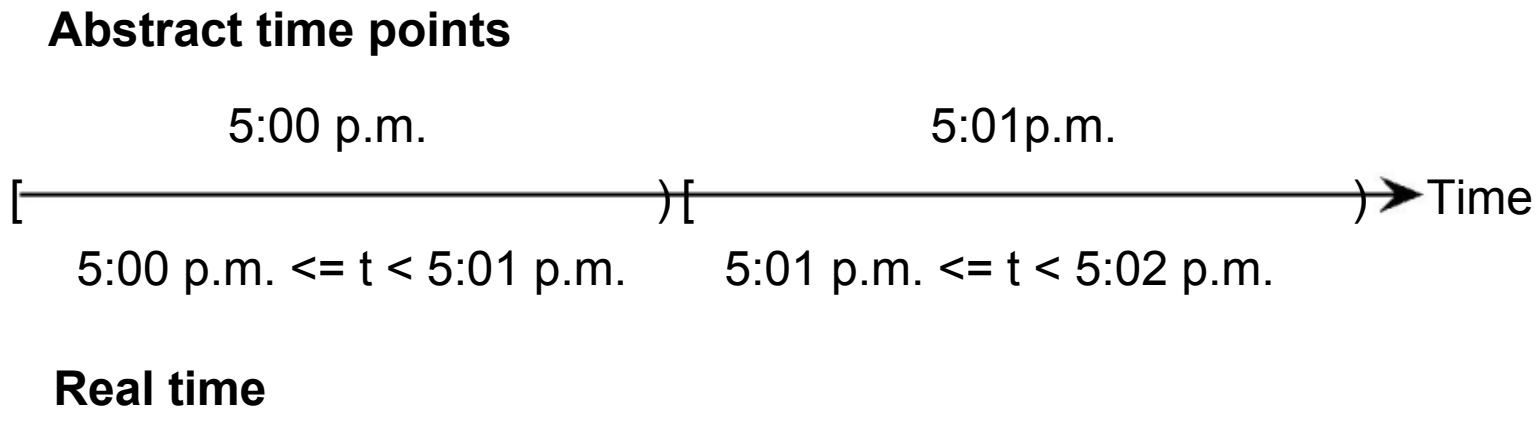
- **Concurrent action semantics.** The actions in a complex action s are all executed simultaneously. (Statemate)
- **Sequential action semantics.** The actions are executed in some sequence. (UML)

What if a UML action is an operation call?



Time

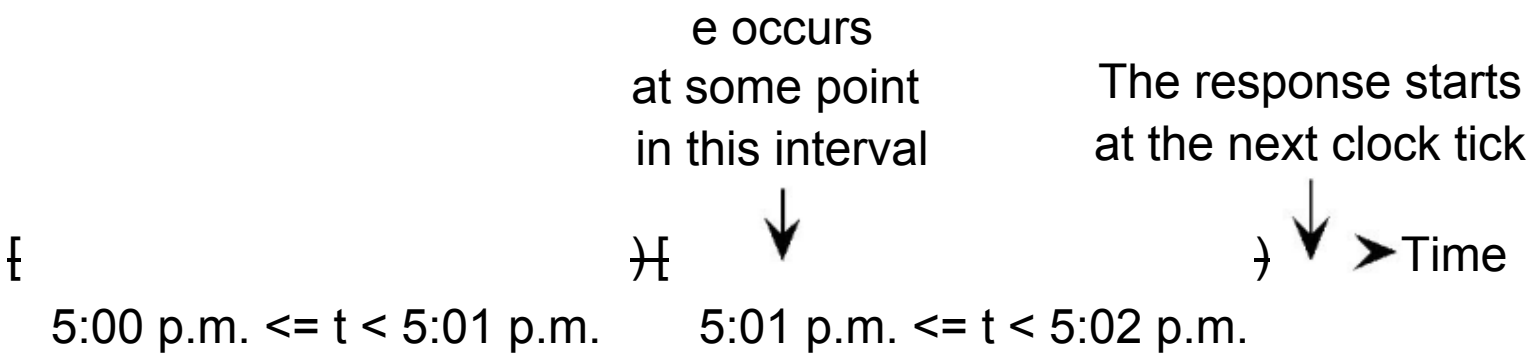
A "**time point**" is a time interval whose length is not significant for the description.



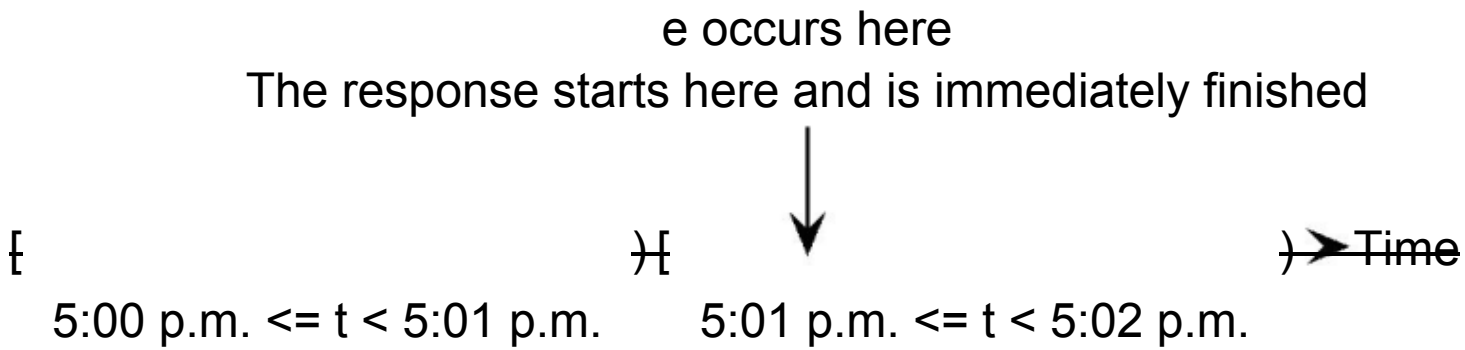
Events

Our events are abstractions: They are instantaneous.

- **Clock-driven semantics** (Statemate)



- **Event-driven semantics** (Statemate and UML)



Perfect technology asssumption

No implementation restrictions. Infinite speed, unlimited memory.

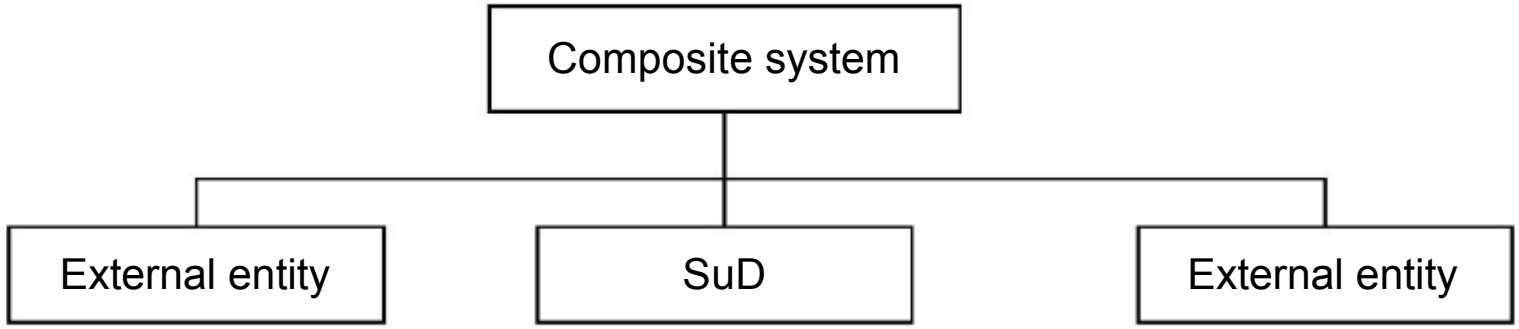
- Clock-asynchronous: Respond immediately to an event occurrence.
- Perfect technology: Response takes no time to compute.

Together this implies a superstep semantics.
This agrees well with requirements-level models.

Main points

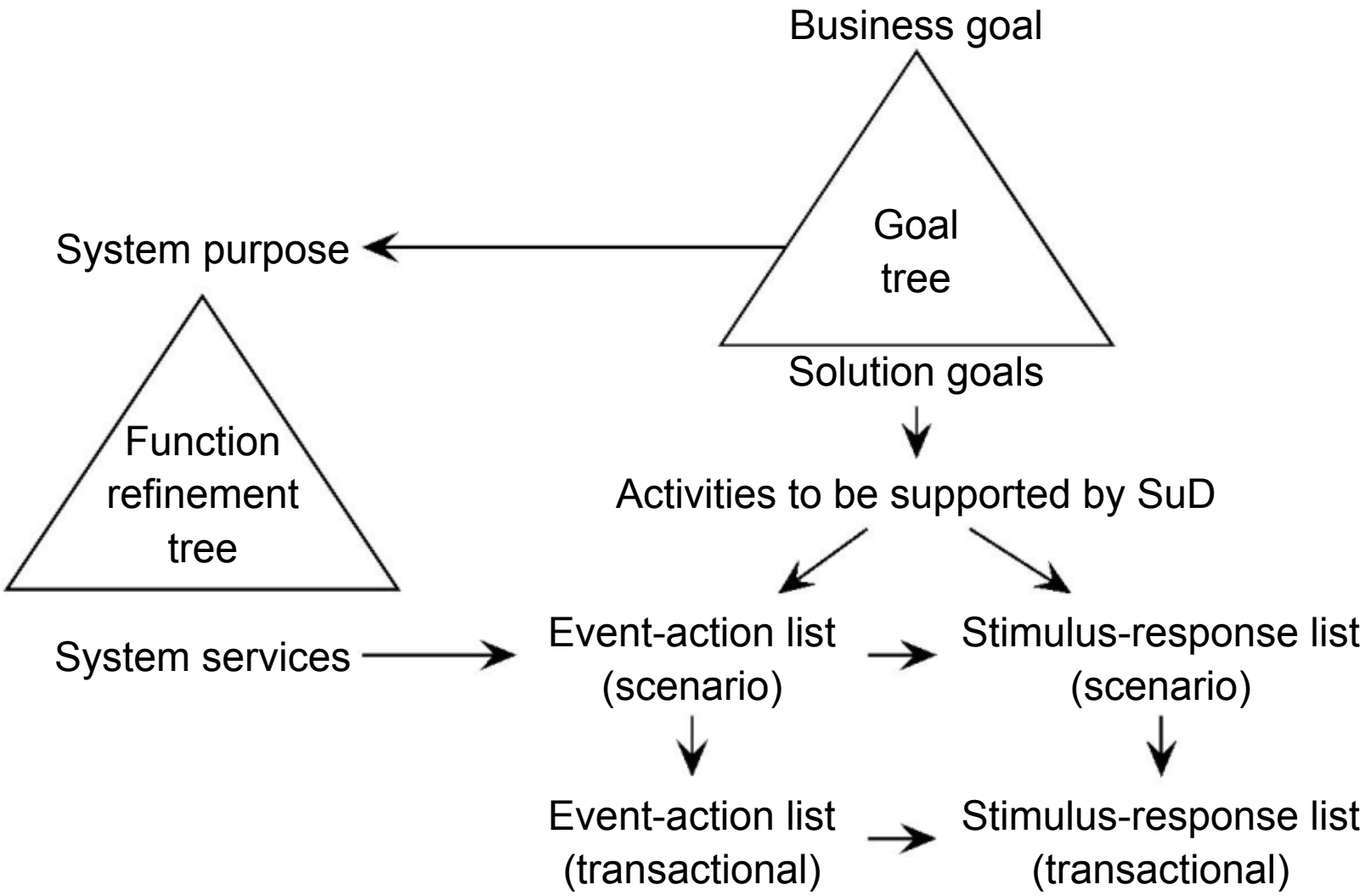
- STTs and STDs are describe discrete behavior that abstracts from continuous behavior in the real world.
- Conflict between transitions inside and out of activity states can be resolved differently.
- Final states can be global or local.
- Ambiguity in pre– and postconditions in STTs and STDs can be reduced by means of derivation rules and frame rules.
- The absence of event triggers can be interpreted in different ways: impossible, ingnore, inhibit, unknown.
- Triggering multiple transitions: Steps versus single transitions.
- Derived transitions: Single steps, supersteps, delayed steps.
- Action semantics: Concurrent or sequential action execution.
- Time: Clock-driven or event-driven semantics. Perfect technology assumption.

Chapter 14. Behavior Modeling and Design Guidelines

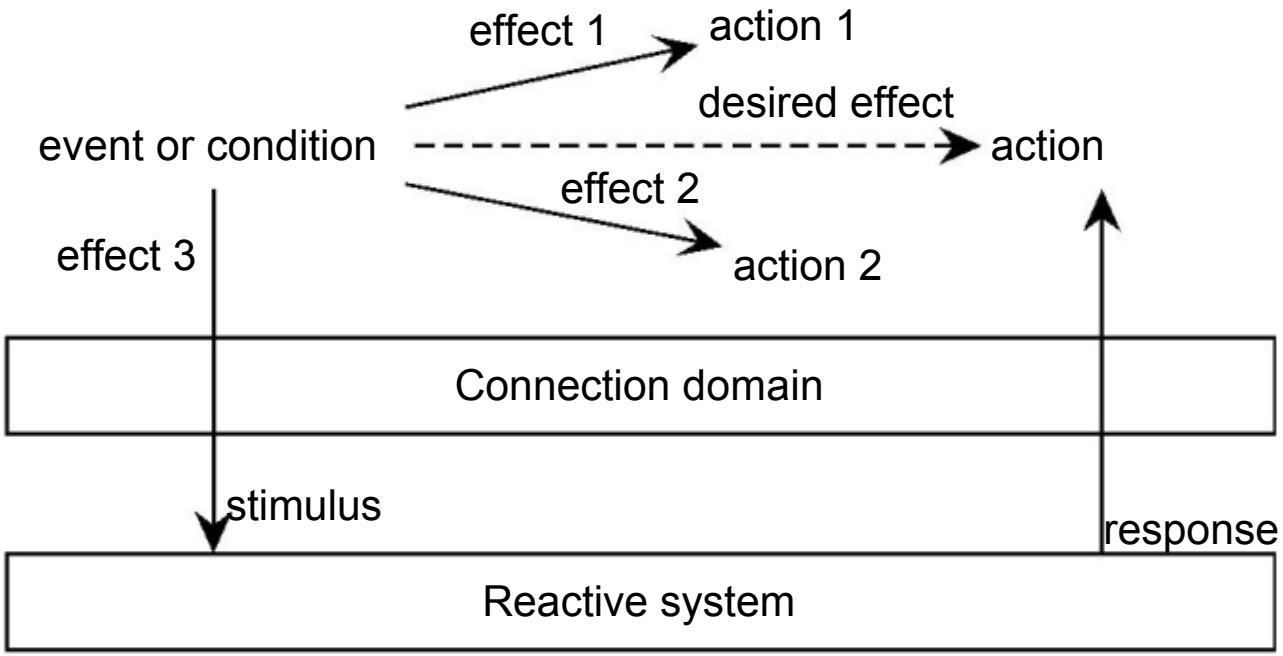


- **Systems engineering argument:** (System specification) and (Assumptions about environment) entail (Emergent properties of composite system).
- To find SuD specification, (1) describe desired emergent behavior, (2) derive system specification, (3) accumulate necessary assumptions when system behavior is not sufficient to produce desired emergent behavior.

Road map


Business goal
Goal tree
System purpose
Solution goals
Function refinement tree
Activities to be supported by SuD
System services
Event-action list (scenario)
Stimulus-response list (scenario)
Event-action list (transactional)
Stimulus-response list (transactional)

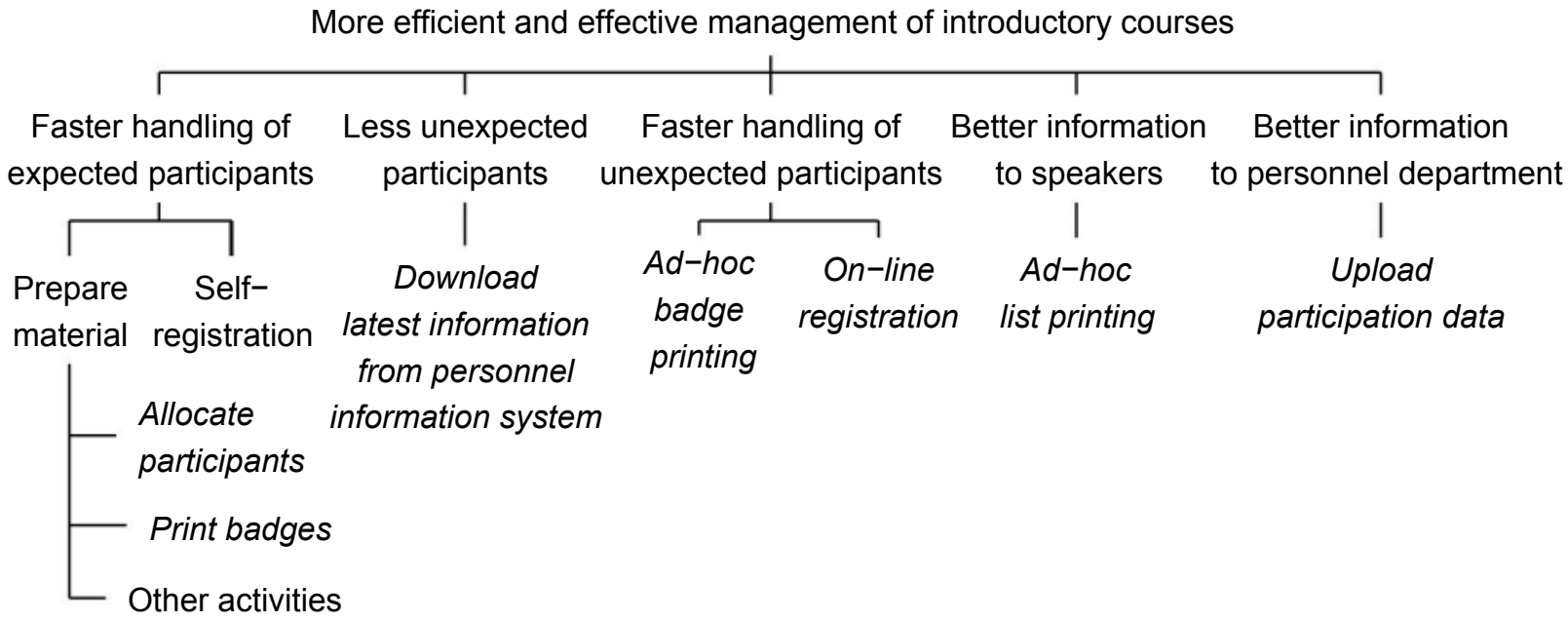
Bringing about desired emergent behavior



- The role of a reactive system is to bring about desired effects in the environment.
- Informative, directive, manipulative functions.

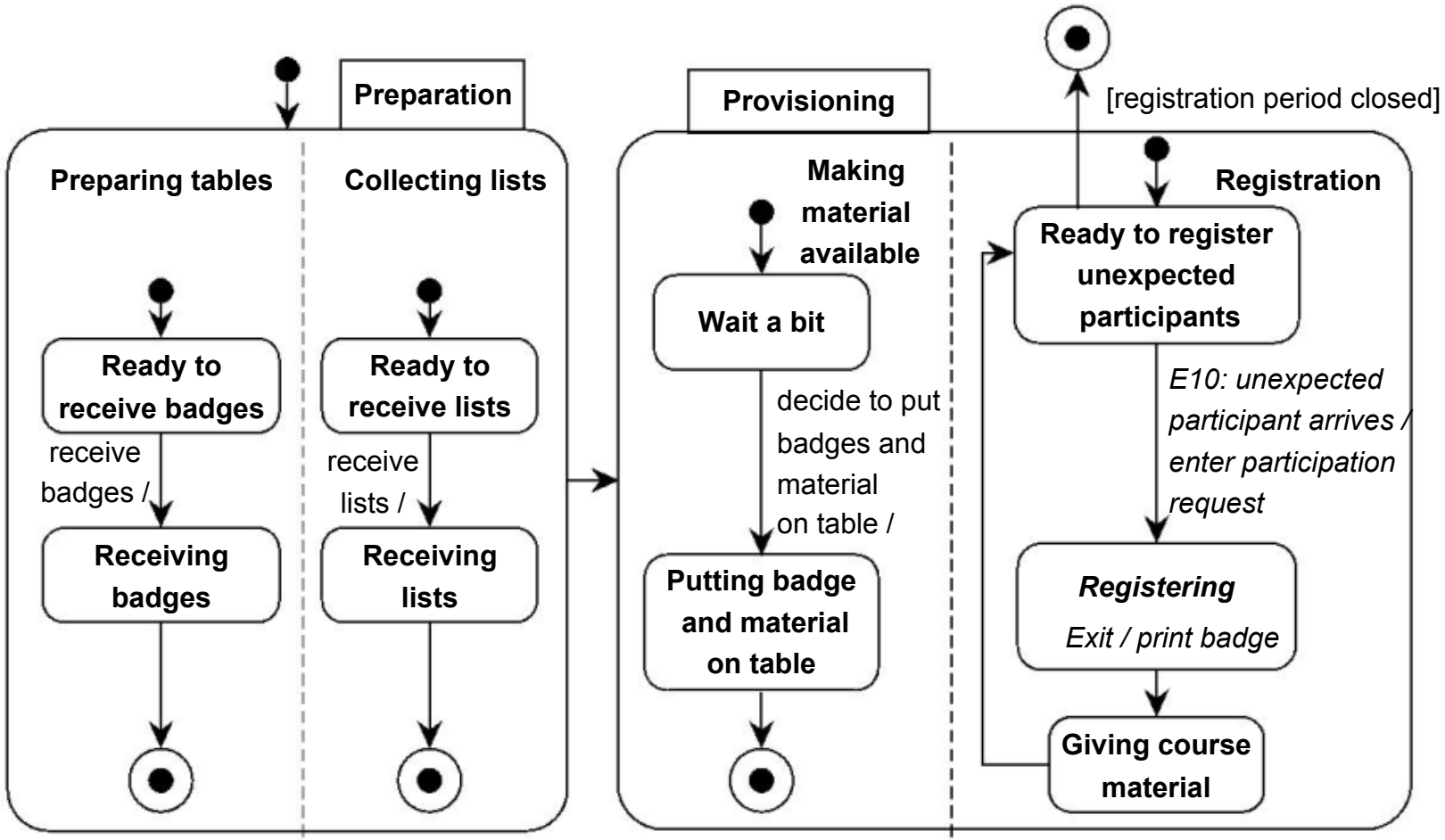
Example 1: Training department

Business goal tree of Training Department



Example 1: Training department

Workflow at the registration desk

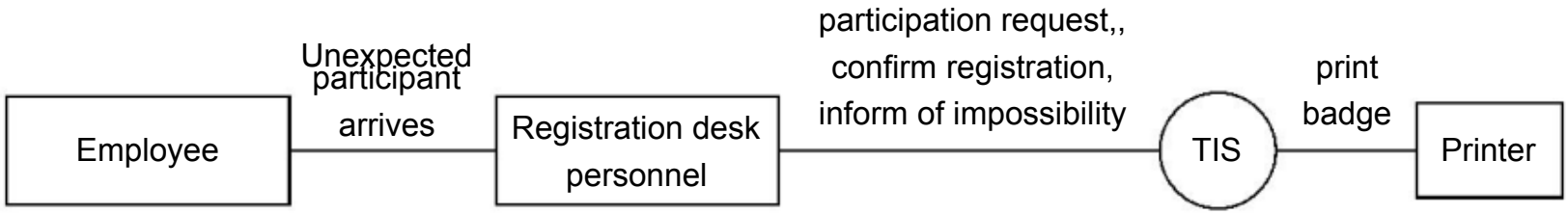


Example 1: Training department

Desired emergent behavior

Event	Desired action
E10 Unexpected participant arrives	If there is still room, the participant should be registered and a badge should be printed.

Embedding the system in its environment



Example 1: Training department

Desired stimulus-response behavior of system

Stimulus	Current state	Response	Next state
S10 Request to register participant is entered.	According to the data of the system, there is still room.	Allocate participant, inform to registration desk personnel, and send badge printing command to printer	System contains updated list of participants.
	According to the data of the system, there is no room.	Inform to registration desk personnel of impossibility.	System contains same list of participants.

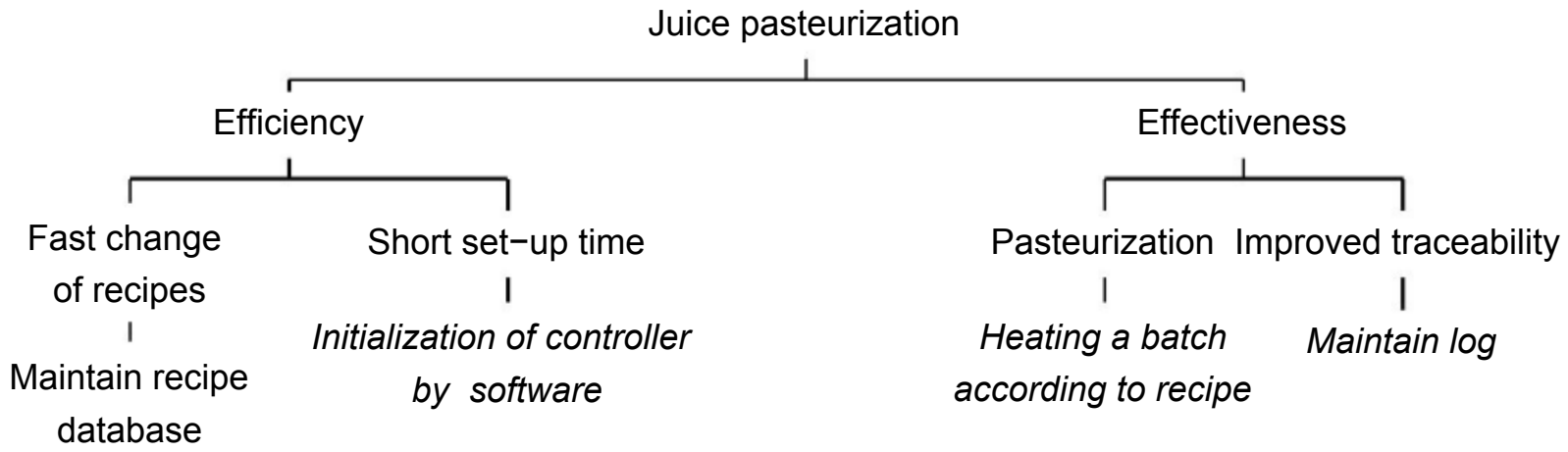
Example 1: Training department

Question

- Which assumptions did we make about the environment?

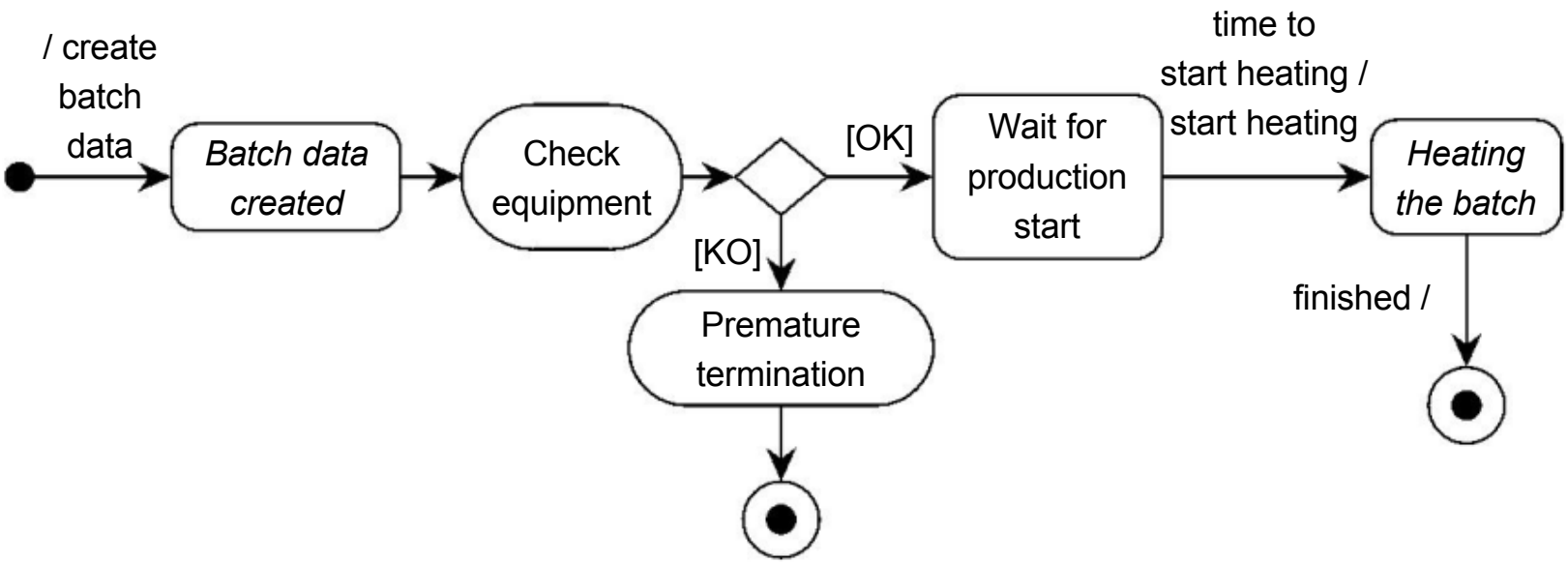
Example 2: Juice pasteurization

Business goal tree of pasteurization department



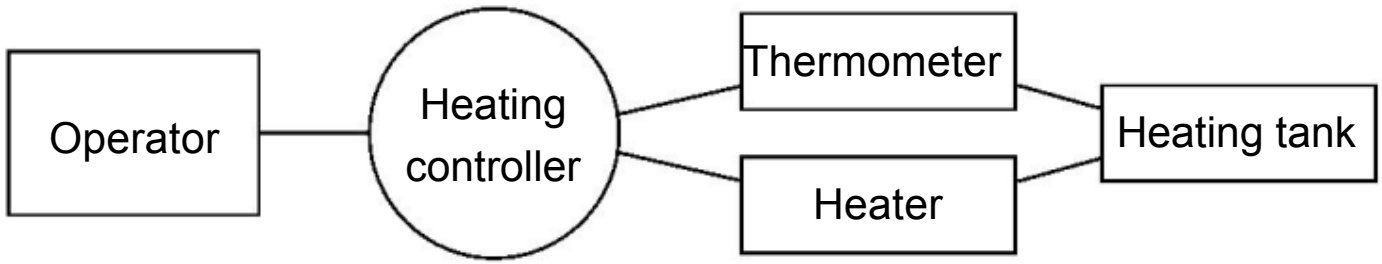
Example 2: Juice pasteurization

Operator workflow



Example 2: Juice pasteurization

Embedding the controller in its context



Example 2: Juice pasteurization

Desired emergent behavior

Event	Desired action
Operator gives command to start heating batch b.	A heating process for the heating tanks of b is started. If at the start of the process, temperature in a tank is too low, the heater of that tank is switched on. When during the process, a tank becomes 5 degrees Celsius warmer than the desired temperature, its heater must be switched off. When it becomes 5 degrees Celsius colder than the desired temperature, its heater must be switched on. When the heating process has lasted for the duration of the recipe, heating must stop and the operator must be notified of this fact.

Example 2: Juice pasteurization

Desired emergent behavior: Transactions

Event	Subject domain state	Desired action
E1 Operator gives command to start heating batch b		Heaters of tanks of b that are below recipe temperature, are switched on.
E2 Temperature in tank t rises 5 degrees above recipe temperature	The juice in t is being heated.	The heater of t is switched off.
E3 Temperature in tank t falls 5 degrees below recipe temperature	The juice in t is being heated.	The heater of t is switched on.
E4 The heating duration has passed, counted since the start of heating of b.	b is being heated.	• Heaters of b that are on, are switched off. • Operator is informed.

Example 2: Juice pasteurization

Required controller behavior

Stimulus	Current controller state	Desired response	Next state
S1 Operator gives command to start heating batch b	Not heating t and not heating b.	Switch on heaters of tanks with low temperature.	Heating t and b.
S2 Every 60 seconds.	Heating t and measured temp $>$ desired temp $+$ 5	Controller switches off the heater of t.	Heating t.
	Heating t and measured temp $<$ desired temp $-$ 5	Controller switches on heater of t.	Heating t.
S4 Recipe time since the start of heating of b has passed.	Heating b.	• Switch off heaters of b that are on. • Inform operator.	Not heating b.

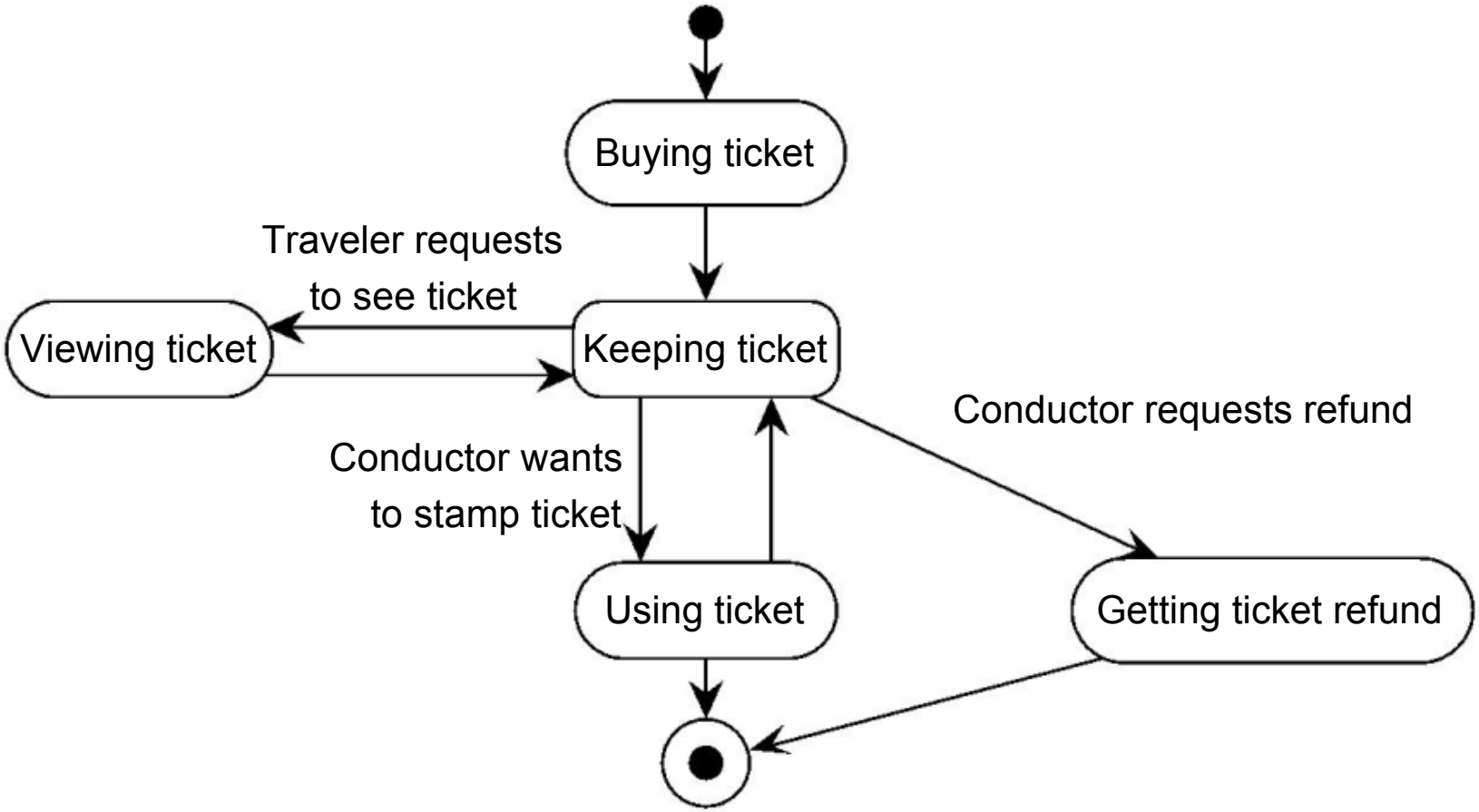
Example 2: Juice pasteurization

Assumptions about environment

- The operator works according to the operator workflow.
- The operator only gives the start command when the batch is in its heating tanks.
- The heater of a tank never breaks.
- The controller is connected to the heaters of the tanks.
- The thermometer of a tank never breaks.
- The controller is connected to the thermometers of the tanks.

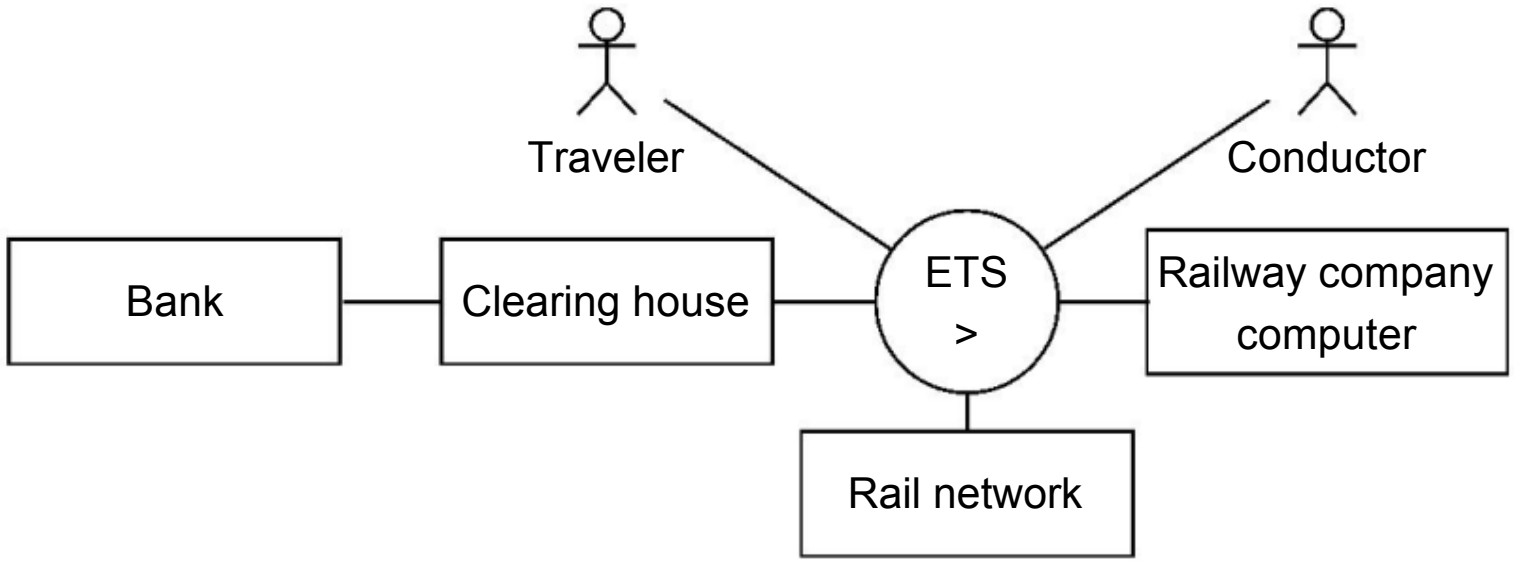
Example 3: Electronic Ticket System

"Workflow" of a ticket



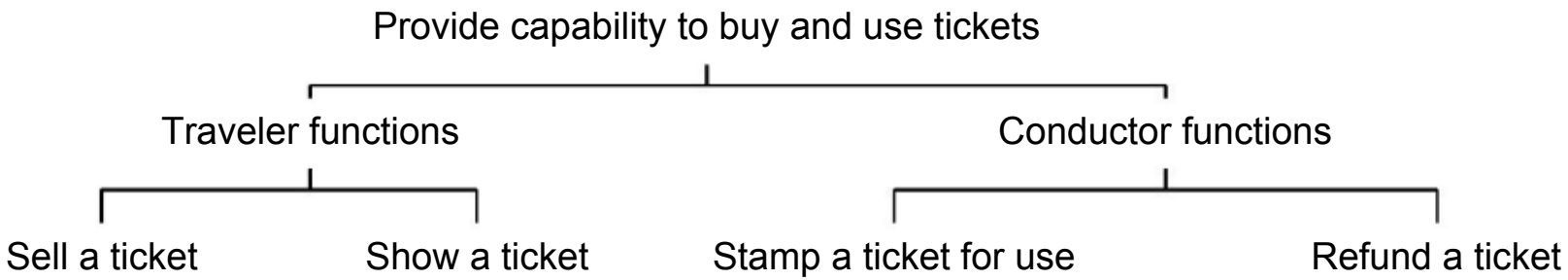
Example 3: Electronic Ticket System

The ETS in its logical context



Example 3: Electronic Ticket System

Required ETS functions



Example 3: Electronic Ticket System

Required ETS services

- **Name:** Sell a ticket.
- **Triggering event:** Traveler requests to buy a ticket.
- **Delivered service:** Allow a traveler to buy a ticket.

- **Name:** Show a ticket.
- **Triggering event:** Traveler requests to view a ticket.
- **Delivered service:** Display ticket attributes to the user.

- **Name:** Stamp a ticket for use.
- **Triggering event:** Request to stamp an unused ticket part.
- **Delivered service:** Mark the requested part of the ticket as used.

- **Name:** Refund a ticket.
- **Triggering event:** Request to refund an unused ticket part.
- **Delivered service:** Cause a refund and make invalid.

Example 3: Electronic Ticket System

Other desired properties

- A traveler cannot get a ticket without paying for it.
- The price of a ticket is withdrawn from the bank account associated with the smart card.
- A traveler who has paid for a ticket gets it.
- A refunded ticket cannot be used any more.
- A fully used ticket cannot be refunded.
- It is not possible to use a ticket twice.

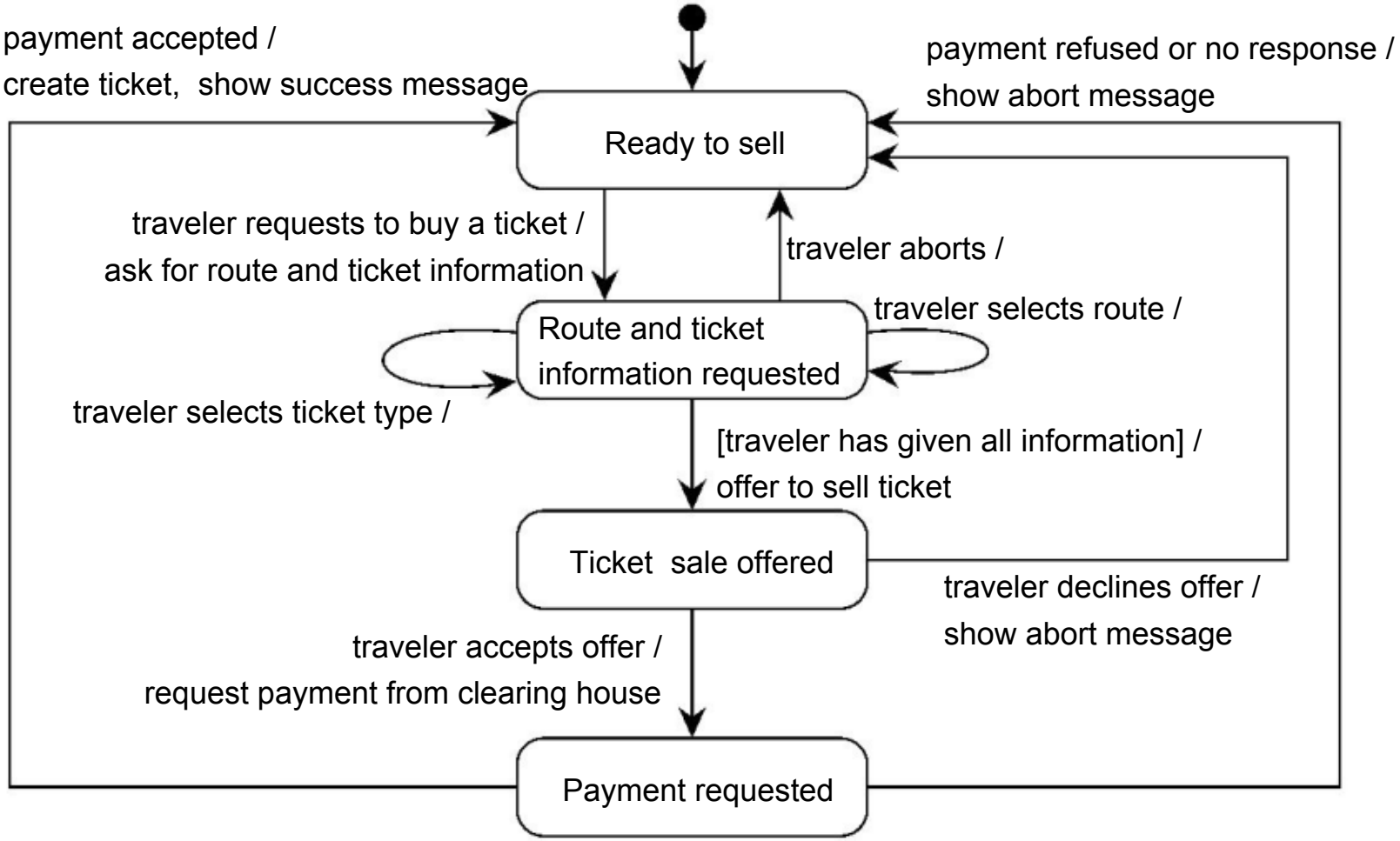
Example 3: Electronic Ticket System

Desired emergent behavior

Event	Current domain state	Desired action	Next domain state
Request to buy ticket	Any	Ticket created and paid for	Ticket and payment exist
Request to see tickets	Any	All tickets are displayed	Unchanged
Request to use ticket for a rail segment	Ticket exists and is valid for that rail segment	Create ticket stamp for that rail segment	Ticket exists and ticket usage stamp exists for this rail segment and ticket
Request to refund ticket	Ticket exists and not completely used	Create refund stamp	Ticket refunded and ticket is not longer valid

Example 3: Electronic Ticket System

Selling scenario



Example 3: Electronic Ticket System

What is so special about this example?

- Some subject domain entities are virtual.
- They are implemented by the system
- So some required system behavior *is* desired subject domain behavior (rather than causing it indirectly).

Summary of guidelines (1)

Finding states:

- √ Look for states in which the behavior is waiting for something to occur.
- √ Look for modes of behavior (e.g. normal-standby-emergency etc.)

Finding events:

- √ Look for signals, condition changes, temporal events to which the system must respond.
- √ Look for desired effects of the system. What triggers the system to produce such an effect?

Summary of guidelines (2)

Dealing with parallellism:

- √ If you cannot show with absolute certainty that two events occur in sequence, then assume they occur in parallel.

Introducing hierarchy:

- √ Introduce hierarchy to express commonality in behavior.
- √ Introduce hierarchy to introduce common response to some event (e.g. alarm, suspend, mode change, etc.)

Summary of guidelines (3)

Finding system behavior:

- √ Follow several paths through our road map and check whether they lead to the same behavior specification.
- √ Through desired causality to system transactions.
  - Identify business solution goals.
  - Identify solution activities in subject domain, user workflow.
  - Identify desired event-action in the environment.
  - Map to stimulus-response pairs of SuD; introduce connection domain if necessary.
  - Refine to transactions by introducing SuD states.

Summary of guidelines (4)

√ Through desired system services to system transactions.

- Identify business solution goals.
- Derive desired system services: triggers and added value, assumptions.
- Include connection domain.
- Refine to stimulus-response pairs; Introduce system states

Behavior Modeling and Design Guidelines: Main Points

- To find required system behavior, look for desired emergent composite system behavior.
- Reduce this to desired system behavior.
- Accumulate any assumptions about environment needed to

  show that desired system behavior is sufficient to create desired emergent behavior.