State Based Modelling

Budapesti Műszaki és Gazdaságtudományi Egyetem
Hibatűrő Rendszerek Kutatócsoport
BEHAVIOURAL MODELLING
Structural and Behavioural Modelling

- **Structural**
  - static
  - whole and part, components
  - connections

- **Behavioural**
  - dynamic
  - timeliness
  - states, processes
  - reactions to the environment (context)

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”. 

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn".
Viewing Points of Behavioural Modelling

- What the system „does”?  
  - Event based models  
  - Process based models  
  - ...  

- What are the properties of the system now, and how do they „change”?  
  - State based models
Motivating Example: Virtual Keyboard

- What happens, if the top left corner is touched?
  - Q, q, 1 or =
  - Is it fully determined by „the past” only?
Basic Terms: Discrete Events

- **Event**
  - instantaneous change (e.g. at the input/output of the system)

- **Event stream**
  - e.g. one per input/output – one per data source

- **Event space** – \{allowed events\}
  - Readable input values / emittable output values

- **Series of** (instantaneous) **Events**, \(\leq 1\) at a time

Event stream: smart phone status messages
Event space: \{Email, SMS, Battery low\}
Event Based Programming
STATE PARTITIONING
Key Concept: State Space

The **state space**
- is a set of distinct system states,
- from which always **exactly one** element (the **current state**) is characteristic for the system at a time.

**Examples: state spaces**
- days: \{Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday\}
- States of the microwave oven: \{full power, defrost, off\}

**Examples: current state**
- Today is *Thursday*.
- The microwave oven is *off*. 
Properties of the State Space

“always exactly one element is characteristic for the system”

- Not any set of states can be a state space!
  - **Completeness**
    - Always at least one of the states is active
    - Counter example (not a state space!)
      - \{Monday, Tuesday, Thursday, Saturday\} not complete
  - **Mutual exclusivity**
    - Only a single state can be current in a moment
    - Counter examples (no state spaces!)
      - \{Working day, Weekend, Afternoon\} comp. but not exclusive
      - microwave oven: \{door is open, switched off\}
Why Are These Properties That Important?

- On the 29th February at the airport of Düsseldorf
State Refinement, State Abstraction

State refinement and state abstraction, respectively, are set refinement and set abstraction on a state space that result in a new state space.

- (other kinds of abstraction will be discussed later ...)

Repetition:
Set refinement
State Refinement, State Abstraction

- State refinement/abstraction:
  Set refinement/abstraction of a state space
  - \( \{Mo, Tu, We, Th, Fr, Sa, So\} \)
  - Working day, Weekend

- Abstraction of the microwave oven
  - \{full power, defrost, off\} \(\rightarrow\) \{on, off\}
  - „off“ state was not refined

IS IT TRUE?
Motivation for Refinement

- State refinement: Why?
  - Adding implementation detail during a design process
    - e.g. knowing the possible power levels of the oven is important for designing its power adaptor
  - Specialization / extension
    - e.g. a more advanced oven may contain timer
  - Joint behavior of several subsystems (see later)

- More information, more knowledge
  - What is the trade-off?
Motivation for Abstraction

- State abstraction: Why?
  - Useful if the abstract states are „uniform“
    • Merged sub-states are similar in certain aspects
  - Details may be irrelevant for some design phases
    • Easier to work with smaller, simpler state space
    • Less storage for the states, easier processing
    • Hidden details can be changed
    • Corner-case: stateless model (|S| = 1)
  - Sometimes only limited amount of information can be disclosed
  - Frequent form: decomposition (see it later)
Multiple correct state spaces of a single system

E.g.: two disjoint state spaces of the microwave
  o according to power: \{full power, defrost, off\}
  o according to door position: \{open, closed\}
  o not completely independent: if the door is open, power must be switched off

State space of the subsystem
  o It is decidable without knowledge of the system state, which state of the subsystem is the current one.
(Direct) Product of State Spaces

- Considering two state spaces together
  - $S_1 = \{\text{full power, defrost, off}\}$
  - $S_2 = \{\text{open, closed}\}$

<table>
<thead>
<tr>
<th>$S_1 \times S_2$</th>
<th>open</th>
<th>closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>full power</td>
<td>full power and open</td>
<td>full power and closed</td>
</tr>
<tr>
<td>defrost</td>
<td>defrost and open</td>
<td>defrost and closed</td>
</tr>
<tr>
<td>off</td>
<td>off and open</td>
<td>off and closed</td>
</tr>
</tbody>
</table>

Remark: $|S_1 \times S_2| = |S_1| \cdot |S_2|$
(Direct) Product of State Spaces

Direct product of state spaces

- **Composition operation** over the component state spaces
- that results in a new state space (**product state space**),
  - which is formed as the Descartes product of the sets of the component state spaces.

In the product state spaces corresponds

- to each combination of the states of the component state spaces
- a combined state (**state vector**) 

\[
\{\text{AM,PM}\} \times \{1\text{h}..12\text{h}\} \times \{0\text{m}..59\text{m}\} \\
\cup \\
\langle PM, 12h, 08m \rangle
\]
Projection of the State Space to a Component

**Projection** to a component is

- a state abstraction operation
- that from the product state space
  - keeps one or more components,
  - and neglects the others.

\[
\{\text{AM,PM}\} \times \{1\text{h..12h}\} \times \{0\text{m..59m}\}
\subseteq
\langle PM, 12h, 08m\rangle
\]

\[
\downarrow
\]

\[
\langle PM, 12h\rangle
\]

(like the projection of tables)
# Refined Composition of the State Space

- **Dependent state variables**
  - Not all combinations can actually manifest
  - Composite state space is more fine than the product

<table>
<thead>
<tr>
<th>$S_1 \times S_2$</th>
<th>open</th>
<th>closed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>full power</strong></td>
<td>full power and open</td>
<td>full power and closed</td>
</tr>
<tr>
<td><strong>defrost</strong></td>
<td>defrost and open</td>
<td>defrost and closed</td>
</tr>
<tr>
<td><strong>off</strong></td>
<td>off and open</td>
<td>off and closed</td>
</tr>
</tbody>
</table>

**Remark:** projection is still an abstraction relationship
Refined Composition of the State Variables

- Composite state space is more fine than the product
  - ... because we excluded two composite states
  - ... the refined state space has fewer states!
    - until now state space refinement resulted in a higher number of states
    - after refinement the number of states can go up or down
    - the important thing: more is known about the system, more precise description is provided
  - therefore: less systems satisfy the model

<table>
<thead>
<tr>
<th>$S_1 \times S_2$</th>
<th>open</th>
<th>closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>full power</td>
<td>full power and open</td>
<td>full power and closed</td>
</tr>
<tr>
<td>defrost</td>
<td>defrost and open</td>
<td>defrost and closed</td>
</tr>
<tr>
<td>off</td>
<td>off and open</td>
<td>off and closed</td>
</tr>
</tbody>
</table>
Decomposition of the State Variables

- Decomposition: reversing production / composition
  - off and open
  - off and closed
  - defrost and closed
  - full power and closed

- The projected state variables are abstractions

- Why do we decompose?
  - to process state variables separately
  - to store state variables separately

\[
S_1 = \{\text{full power, defrost, off}\}
\]
\[
S_2 = \{\text{open, closed}\}
\]
Example from Our Profession

- Where does state-based modelling play a role in IT?
- Social network – relation between Jack and Jill
  (certain functions depend on this, e.g. visibility of uploaded pictures)

  - State variables:
    - Did Jack mark Jill as a friend?
    - Vice versa

<table>
<thead>
<tr>
<th>$S_1 \times S_2$</th>
<th>no relation</th>
<th>Jack knows Jill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill knows Jack</td>
<td>friends</td>
<td></td>
</tr>
</tbody>
</table>

store separately:
- in RAM
- in a database (permanent)
Example from Our Profession

- What is the motivation behind state abstraction?
- Social networks: only a part of the database can be visible for a single user
  - Privacy
    - We have no right to learn whether Jack knows Jill or not
  - Smaller data traffic
  - Decomposition of a software system
    - Simpler view component (HTML + CSS + JavaScript) if only the relevant information is available
    - More secure, more flexible
    - Single implementation of access control policies
Example from Our Profession

- Virtual keyboard for touchscreens
  - state variables?
Example from Our Profession

- **Programming**: how can we store the state?
  - Variable with appropriate value domain (object field, etc.)

```c
enum VirtualKeyboardState {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK,
    NUMBERS_COMMON_SYMBOLS,
    RARE_SYMBOLS
}

// ...
VirtualKeyboardState keyboardState;
```

- **Extension**: store the state of SHIFT key!
  - the alphanumeric mode „remembers” the state of the SHIFT
Programming: how can we store the state?

- Extension: store the state of SHIFT key!

```cpp
enum VirtualKeyboardStateWithMemory {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK,
    NUMBERS_COMMON_SYMBOLS_WITH_LOWER_CASE,
    NUMBERS_COMMON_SYMBOLS_WITH_UPPER_CASE_ONCE,
    NUMBERS_COMMON_SYMBOLS_WITH_UPPER_CASE_LOCK,
    RARE_SYMBOLS_WITH_LOWER_CASE,
    RARE_SYMBOLS_WITH_UPPER_CASE_ONCE,
    RARE_SYMBOLS_WITH_UPPER_CASE_LOCK
};

VirtualKeyboardStateWithMemory keyboardStateWithMemory;
```

- Phenomenon is called **state space explosion**
Example from Our Profession

- Programming: how can we store the state?
  - compact solution: multiple state variables

```cpp
enum VirtualKeyboardFacet {
    ALPHABETIC,
    NUMBERS_COMMON_SYMBOLS,
    RARE_SYMBOLS
}
enum CapsState {
    LOWER_CASE,
    UPPER_CASE_ONCE,
    UPPER_CASE_LOCK
}
// ...
VirtualKeyboardFacet keyboardFacet;
CapsState capsState;
```

- decomposition of state variables
Potential infinite (or even continuum) state spaces

- e.g. state variables of an airplane
  - $v \in \mathbb{R}$ speed
  - $h \in \mathbb{R}$ flight altitude
  - $\alpha \in [-\pi/2, \pi/2]$ rise angle

- The state change can be continuous
  - e.g. rise of the airplane: $\frac{\partial h}{\partial t} = v \sin \alpha$

But for typical IT system models

- discrete states (no continuous change)
- often finite state space (counter example: counter $\in \mathbb{N}$)
- instantaneous state transitions, fixed states inbetween
SIMPLE (MEALY) AUTOMATA
Potential infinite (or even continuum) state spaces

- e.g. state variables of an airplane
  - \( v \in \mathbb{R} \) speed
  - \( h \in \mathbb{R} \) flight altitude
  - \( \alpha \in [\frac{-\pi}{2}, \frac{\pi}{2}] \) rise angle

- The state change can be continuous
  - e.g. rise of the airplane: \( \frac{\partial h}{\partial t} = v \sin \alpha \)

But for typical IT system models

- **discrete states** (no continuous change)
- often **finite state space** (counter example: counter \( \in \mathbb{N} \))
- instantaneous **state transitions**, fixed states inbetween
State Transitions

- **State space: $S$**
  - e.g. $S = \{Mo, Tu, We, Th, Fr, Sa, So\}$
- $s(t) \in S$
  - The current state as a function of time
Repetition: Binary Relation

- Binary Relation:
  - subset of the Descartes product of two sets
  - \( R \subseteq D_1 \times D_2 \)

\[
R = \{(a, 1), (a, 2), (c, 2), (d, 4)\}
\]
What are possible sequences of states?

- State space \( S = \{Mo, Tu, We, Th, Fr, Sa, So\} \)
- Event space: state transition relation \( R \subset S \times S \)

\[
\begin{align*}
& r_1 = (Mo, Tu) \\
& r_2 = (Tu, We) \\
& r_3 = (We, Th) \\
& r_4 = (Th, Fr) \\
& r_5 = (Fr, Sa) \\
& r_6 = (Sa, So) \\
& r_7 = (So, Mo)
\end{align*}
\]

- Also called a state graph

From Th can the system move to Fr
Remarks about the State Graph

- It is possible that ...
  - ... the graph is complete $\rightarrow$ all transitions are allowed
    - $S_{\text{kitten}} = \{\text{sleeping, playing, drinking}\}$
  - ... not every state is reachable from each state
    - $S_{\text{glass}} = \{\text{empty, full, broken}\} \rightarrow$ no path $\text{broken} \sim \text{empty}$
  - ... some states have multiple successors

Non-determinism

- Possible sources:
  - Behaviour of the modelled system
  - Abstraction introduced during modelling

<table>
<thead>
<tr>
<th>off and open</th>
<th>off and closed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>defrost and closed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

E.g. ignored internal variables, input from environment, ...
SIMPLE (MEALY) AUTOMATA
State Graph Example: ATM

ATM

- Off
  - turnOff
  - turnOn
- Self Test
  - failure
- Idle
  - service
  - turnOff
  - cardInserted
  - cancel
- Maintenance
  - service
  - cancel
- Out of Service
  - failure
- Customer Authentication
  - cardInserted
  - cancel
- Selecting Transaction
  - cancel
- Transaction
  - failure

turnOff
failure
Labelling Transition with Events

- **Label of the transition**
  - Instantaneous event
  - Transition can be associated with an event

- **Possible interpretations:** the event can be...
  - ... the result of the transition (post condition)
  - ... the cause/trigger of the transition (pre condition)

- **A Transition can have multiple labels**
  - Reading input / Writing output
Memory Hook: Mealy Finite State Machine

- Initial state $s_0 = s(t=0)$
- All transitions deterministic, reading input and writing output
- Compare to automaton model in classic system theory

Question:
- What does this machine do?
- If rule $r_6$ transfer to $s_0$?
Extensions of Mealy Machine

- Nondeterministic model (wrt. input)
- „Spontaneous” transition without reading input
  - Internal, effect of non-modelled events
  - Heating finished, oven switches off (timer not modeled)
- Multiple output channels (separate event streams!)
  - Disjoint signal sets
  - The rule emits signals to a specific subset of channels
- Multiple input channels (separate event streams!)
  - The rule reads input signal from a subset of channels
  - This can cause non-determinism
Extended State Machine

$M_1$

$\{0,1\}$

$i_1$

$\{a,b,c\}$

$i_2$

$\{0,1\}$

$O_1$

$\{d,e,f\}$

$O_2$
Unspecified Input

- Do we need a rule for every possible input in each state?
  - If no rule $\rightarrow$ transition is not allowed
    - Theoretically it is so, but for real systems it is not realistic
    - What happens if such a situation occurs anyway?
  - If no rule $\rightarrow$ invisible loop transition
    - All unspecified input tokens are consumed but ignored
  - If no rule $\rightarrow$ invalid model
    - E.g. critical embedded system
    - Multiple rules are also invalid (determinism is required)

- If there are rules for each case $\rightarrow$ fully specified
Examples: Comm. Protocol

- Packet based communication protocol
  - Packets can be lost, can be damaged
  - Acknowledge, resend

  (For more details see course Computer Networks)
Examples: Virtual Keyboard

- Virtual keyboard

![Diagram of Virtual Keyboard]

- \{BUTTON_1, \ldots\}
- \{q, Q, 1, =, \ldots\}

- lower case
- upper case
- upper case lock
- Numbers, common symbols
- rare symbols

- \textsc{Button}_1/q
- \textsc{Button}_1/O
- \textsc{Button}_1/1
- \textsc{Button}_1/=
Examples

- Programming: implementing state machine
  - Branching condition:
    - State variables and
    - input
  - All branches:
    - output emission (if any)
    - State transition (if needed)

```java
void handleKey(KeyCode input) {
    switch(input) {
        case BUTTON_1:
            switch(keyboardState) {
                case LOWER_CASE:
                    emit('q');
                    break;
                case UPPER_CASE_ONCE:
                    keyboardState = LOWER_CASE;
                    break;
                case UPPER_CASE_LOCK:
                    emit('Q');
                    break;
                case NUMBERS_COMMON_SYMBOLS:
                    emit('1');
                    break;
                case RARE_SYMBOLS:
                    emit('=');
                    break;
            }
        case SWITCH_1:
            //...
        break;
    }
}
```
Remark

- If the (state machine) model is...
  - ... detailed enough (deterministic), and
  - ... formalized in a way, which can be processed
    - E.g. domain specific languages (protocol design)
    - E.g. standard modeling notation (UML)
- ... can be translated automatically into source code
  - E.g. code generation for a communication protocol
  - E.g. development of an embedded controller
- ... or can be executed with an interpreter
  - E.g. IT system management application
STATE MACHINE EXTENSIONS
State Chart Languages

- (Harel) State Chart = state machine +
  - State hierarchy
  - Orthogonality
  - Variables
  - Pseudo states
  - ...

- E.g.
  - Yakindu
  - UML
Calculator Example
CALCULATOR

State Hierarchy

- Off
- On
- Off
- Digit
- On
- Off
- Digit
- Operator
- On, Digit
- Eval
- Displaying result
- Reading operand1
- Reading operand2
State Hierarchy

CALCULATOR

Off

Reading operand1

Operator

Reading operand2

Eval

Displaying result

Digit

Digit

On

On, Digit

On

On

Off

Off

Off
State Hierarchy

CALCULATOR

On

Off

Reading operand1

Displaying result

Reading operand2

On

Eval

Digit

Operator

On

Off

Digit

Digit
CALCULATOR

- State configuration: \{Off\}
**State Hierarchy**

- **CALCULATOR**

  - **State configuration**: \{On, Reading operand1\}
State Hierarchy

- State configuration: \{On, Reading operand2\}
State Hierarchy

- State configuration: \{On, Reading operand2\}

Two active states $\rightarrow$ What about mutual exclusivity?
- A statechart is not a state space
- Only states on the same hierarchy level give a mutual exclusive set of states. (but not a state space, either)
ATM Example

BANK ATM

Off

Self Test

Idle

Maintenance

Out of Service

Serving Customer

Customer Authentication

Selecting Transaction

Transaction

turnOff

turnOn

failure

service

failure

service

failure

failure

turnOff

cancel

cardInserted

cancel
Example: robot hoover
**Orthogonality**

### Place of the Robot Hoover

- **Wet surface**
- **Pile surface**
- **Flat surface**

- **w | p**
- **f | p**
- **w | f**

### Mode of the Robot Hoover

- **Inactive**
- **Cleaning**

  - **Deactivate | Activate**
Orthogonality

PLACE AND MODE OF THE ROBOT HOOVER

- Wet surface
- Pile surface
- Flat surface
- Cleaning

Wet | Pile surface | Flat surface | Cleaning

w | p | f

inactive

deactivate | activate
Orthogonality

State configuration: \{pile surface, inactive\}
Orthogonality

**PLACE AND MODE OF THE ROBOT HOOVER**

- State configuration: \{pile surface, cleaning\}
Needs further refinements: Transitions are excluded $\rightarrow$ States may become unreachable
Definition: Asynchronous Product

The **asynchronous product** of (Mealy-) state machines is a **composition operation** over the component state machines (also called the regions. The **result** of the composition is a (Mealy-) state machine.

- **State space:** the direct product of the state spaces of the regions
- **Initial state:** every region in its initial state
- **Transition rules:** all transition rules in which
  - exactly one region makes a transition,
  - while all other regions keep their actual states.
Variables

- Infinite counter
  - \( S = \mathbb{N} \)

- Introduction of variables: \( x \)
  - (like a separate region)

\[
(count, \{ x \mapsto 0 \}) \rightarrow (count, \{ x \mapsto 1 \}) \rightarrow \ldots
\]
Variable + Guard Condition

- **Cycle counter**
  - $S = \{0, 1, \ldots, i\}$

- **With Guard Conditions:**
  - tick $[x < i]$ / $x := x + 1$
  - tick $[x \geq i]$ / $x := 0$
Pseudo States

- Pseudo state:
  - Semantically it is not a state:
    - There is no time instant when it represents the state of the system
  - Syntactically it is a state:
    - Can be the start or the end state of a transition

\[
\begin{align*}
\text{x:= 0} & \quad \text{count} \\
\text{[x < i] / x:= x + 1} & \quad \text{tick} \\
\text{[x \geq i] / x:= 0}
\end{align*}
\]
How can we model cooperation?

- How exactly are the two regions **not independent**?
Cooperation by **guards**

- The condition of a transition in one region is a state in the other region.
Asynchronous Product: Cooperation

**PLACE AND MODE OF THE ROBOT HOOVER**

inactive on pile surface

inactive on wet surface

inactive on flat surface

cleaning on pile surface

cleaning on wet surface

cleaning on flat surface

w | p [false]

p | f [false]

w | f [false]

w | f [true]

p | f [true]

deactivate | activate

deeactivate | activate

deeactivate | activate
Example: Synchronous Product

- Fridge with compressor & lamp
  - Common input: door

  COMPRESSOR & LAMP OF THE FRIDGE

<table>
<thead>
<tr>
<th>cooling</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>close</td>
</tr>
<tr>
<td>open</td>
<td>open</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>light</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
</tr>
<tr>
<td>open</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dark</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
</tr>
<tr>
<td>open</td>
</tr>
</tbody>
</table>
Example: Synchronous Product

- Reading the input together → synchronous steps
  - (States may become unreachable)
**Synchronous Product**

- **Product of state machines**
  - Composing the model from state regions (components)
  - State space: **direct product** of the region state spaces
  - Initial state: $n$-tuple of the initial states of every region

- **Transition rules: all transition rules in which**
  - each region makes a transition **at the same time**.
  - “gluing” transitions, union of the labels
Mixed Product

- **Product of state machines**
  - Composing the model from state regions (components)
  - State space: **direct product** of the region state spaces
  - Initial state: n-tuple of the initial states of every region

- **Transition rules: sometimes synchronised**
  - Basically an asynchronous composition ...
  - ... but in some cases the regions act together

A simple case of synchronisation: there are (also) common inputs
Example: Mixed Product

- Further refinement required
  - Transitions may vanish
  - (States may become unreachable)

---

**MAGNETRON**

**DOOR**

**DOOR & MAGNETRON**

Neglected inputs: „imaginery” loops
**Mixed Product**

- **Product of state machines**
  - Composing the model from state regions (components)
  - State space: *direct product* of the region state spaces
  - Initial state: n-tuple of the initial states of every region

- **Transition rules: sometimes synchronised**
  - Basically an asynchronous composition ...
  - ... but in some cases the regions act together

---

A simple case of synchronisation: there are (also) common inputs

Advanced cooperation: *rendezvous* (internal synchronising event)
Overview of Products

- **Product of state spaces**
  - **Direct product**: \( S_1 \times S_2 \times \ldots \times S_n \)
    - Composed states: \( n \)-tuples of the states of the components

- **Product of state machines**
  - The state space is always (refinement of) the direct product
  - **Synchronous product**
    - The components/regions step always at the same time
  - **Asynchronous product**
    - The components/regions step always one at a time
  - **Mixed product**
    - Sometimes at the same time and sometimes one at a time
Overview of Products

- **Product of state spaces**
  - **Direct product**: $S_1 \times S_2 \times \ldots \times S_n$
    - Composed states: n-tuples of the states of the components

- **Product of state machines**
  - The state space is always (refinement of) the direct product
  - **Synchronous product**
    - The components/regions step always at the same time
  - **Asynchronous product**
    - The components/regions step always one at a time
  - **Mixed product**
    - Sometimes at the same time and sometimes one at a time

**Modes of cooperation**

- Guards
- Rendezvous