HASE 2010, San Jose, CA
November 4, 2010

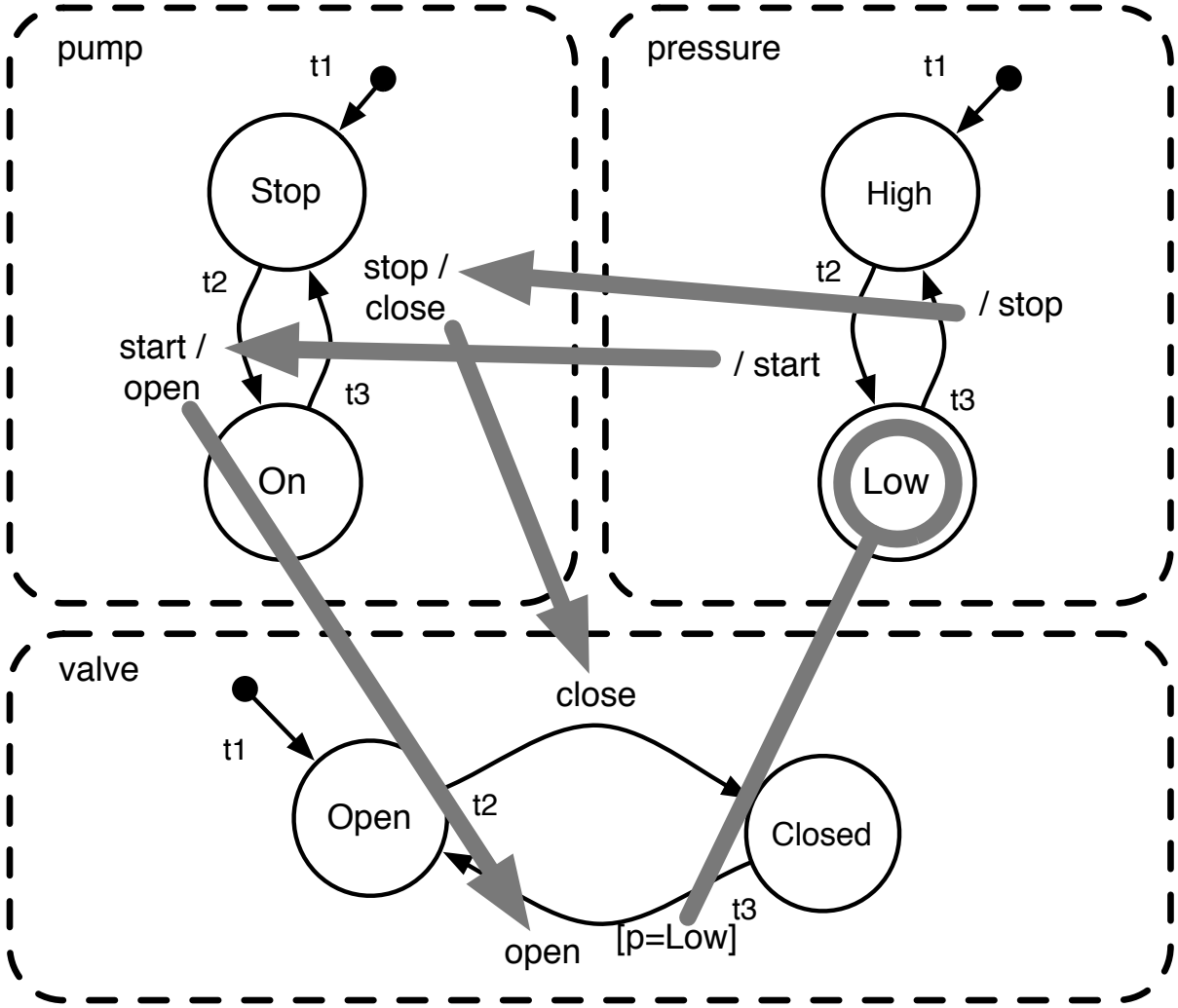
Automatic Fault Behavior Detection and Modeling by a State-based Specification Method

Luca Pazzi, Matteo Interlandi, Marco Pradelli
Department of Engineering Sciences
University of Modena and Reggio Emilia
Via Vignolese 905
I-41125 MODENA, Italy

Outline

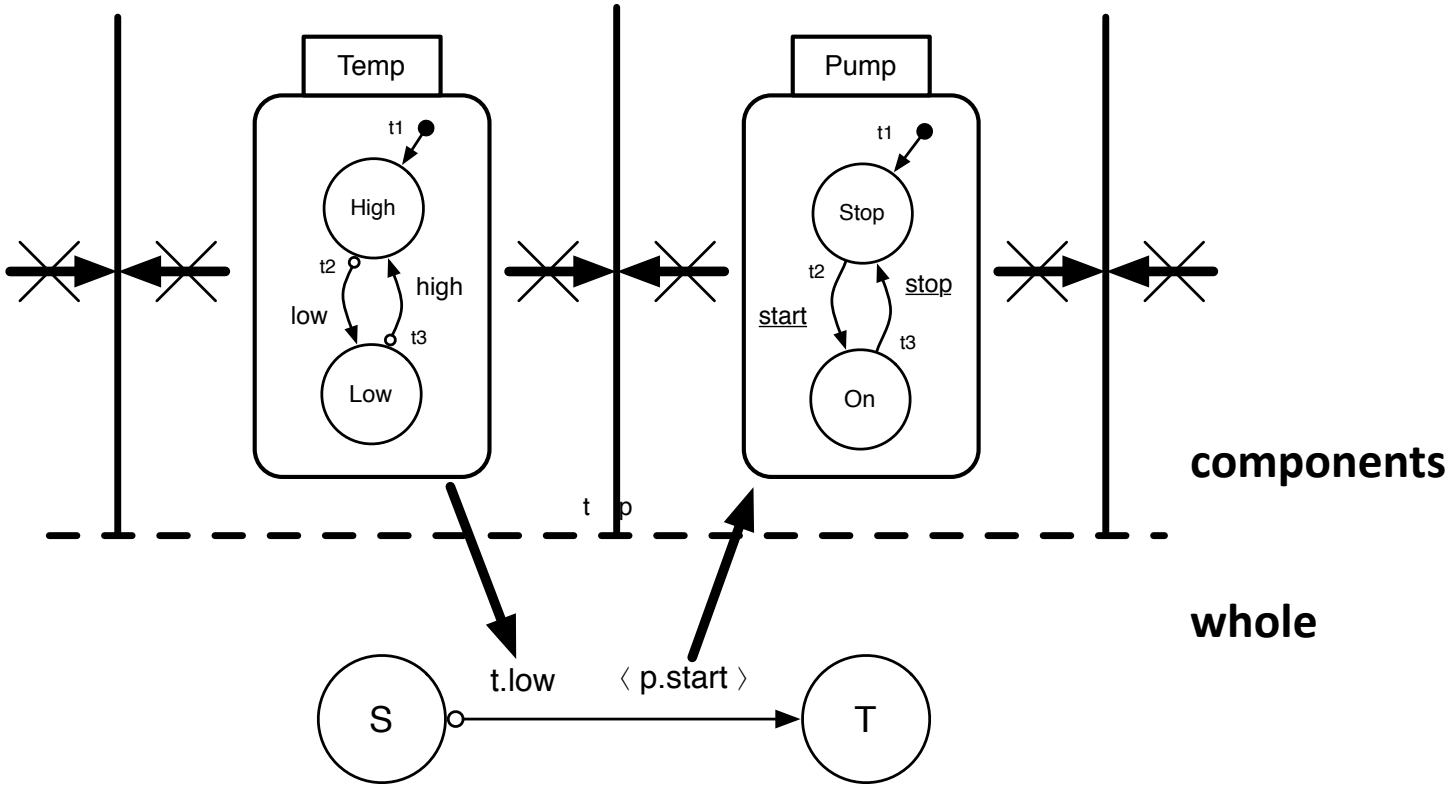
- State of the art in state-based modeling;
  - Open problems:
    - no **computable** semantics;
    - no **modular** constructs;
    - **Implicit** modeling of behavior;
- Our Proposal;
  - State semantics determination at design time;
  - State invariant enforcement
    - “exit zone” determination at design time;
  - **Explicit** representation of compound behavior
    - Feasibility of faulty behavior deduction from regular behavior;
- Conclusions.

Example of traditional Statecharts approach: a medical device [from CBMS2008]

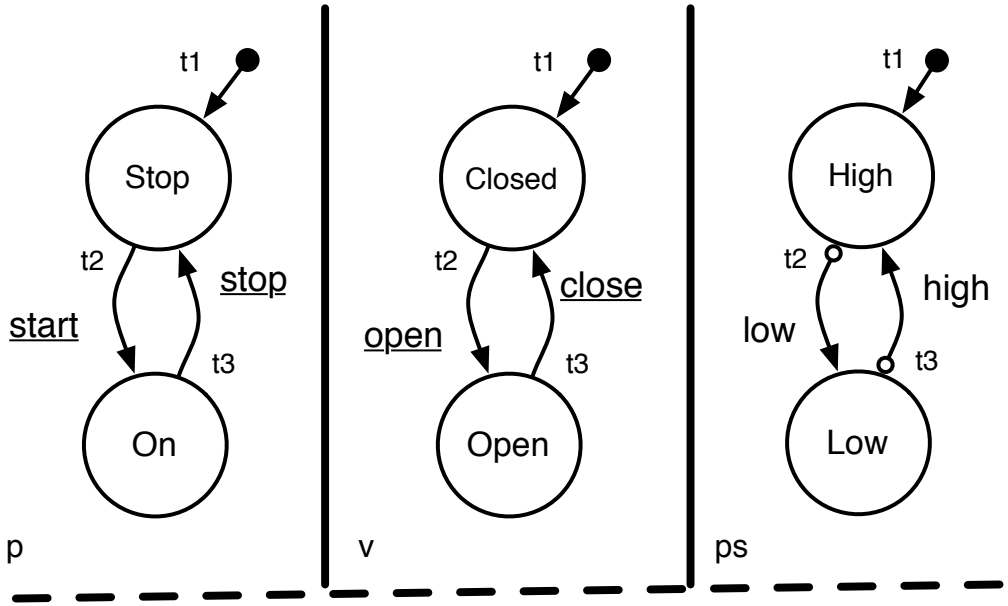


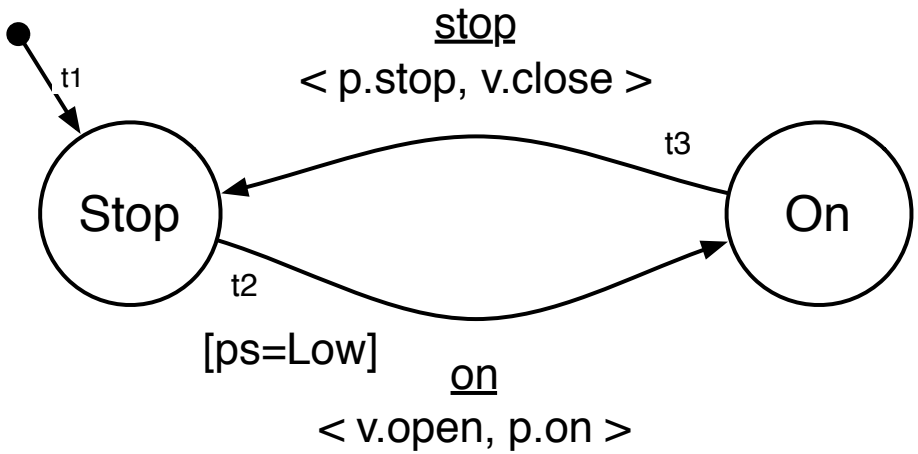
Part-Whole Statecharts Rationale

- Events are *not* allowed to travel among components;
  - **components** are only allowed to communicate with a specific section, hosting an *additional* state machine, called **whole**;



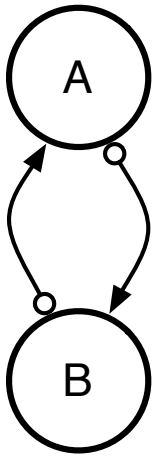
Example of explicit modeling approach: the medical device [from CBMS2008]



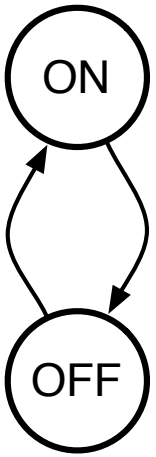


Uncontrollable transitions/events

- We identify two kinds of events that are represented by means of transitions: **controllable** ones and **uncontrollable** ones;
- Typical example: actuators and sensors:
  - **Actuators** are **controllable** and it is reasonable to assume that they are not able to autonomously take actions.
  - Sensors output is not controllable, it is a view of the environment. **Sensors** are represented by means of **automatic** transitions;



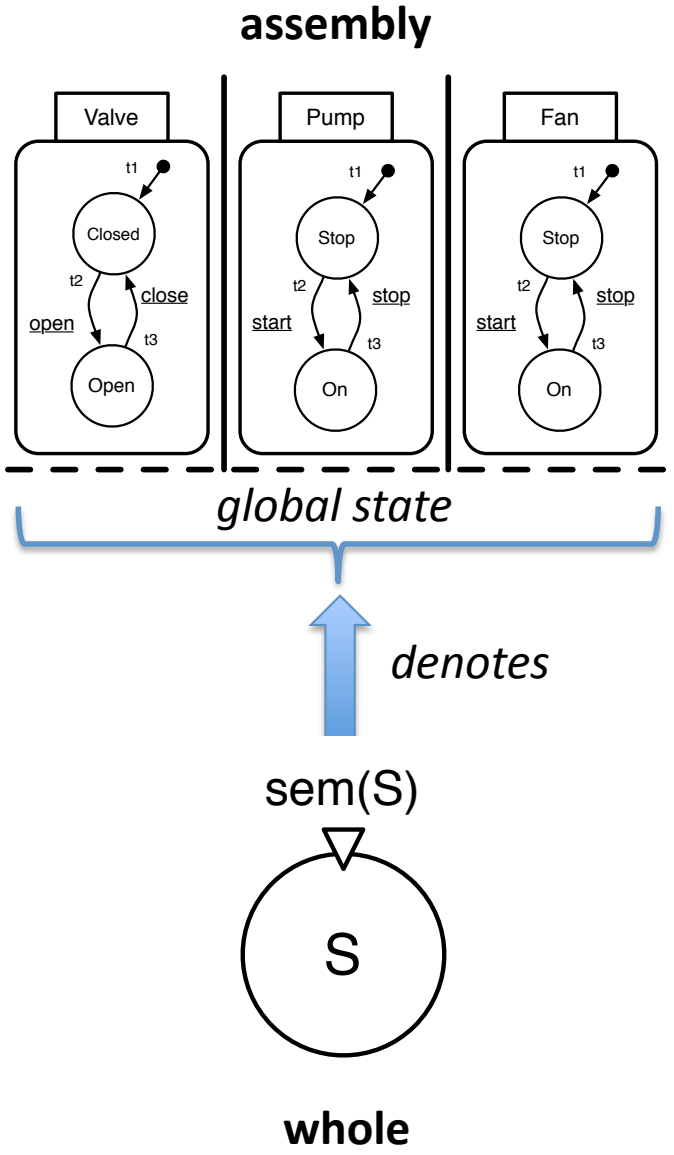
A generic sensor: **automatic transitions** are identified by a small circle.



A generic actuator.

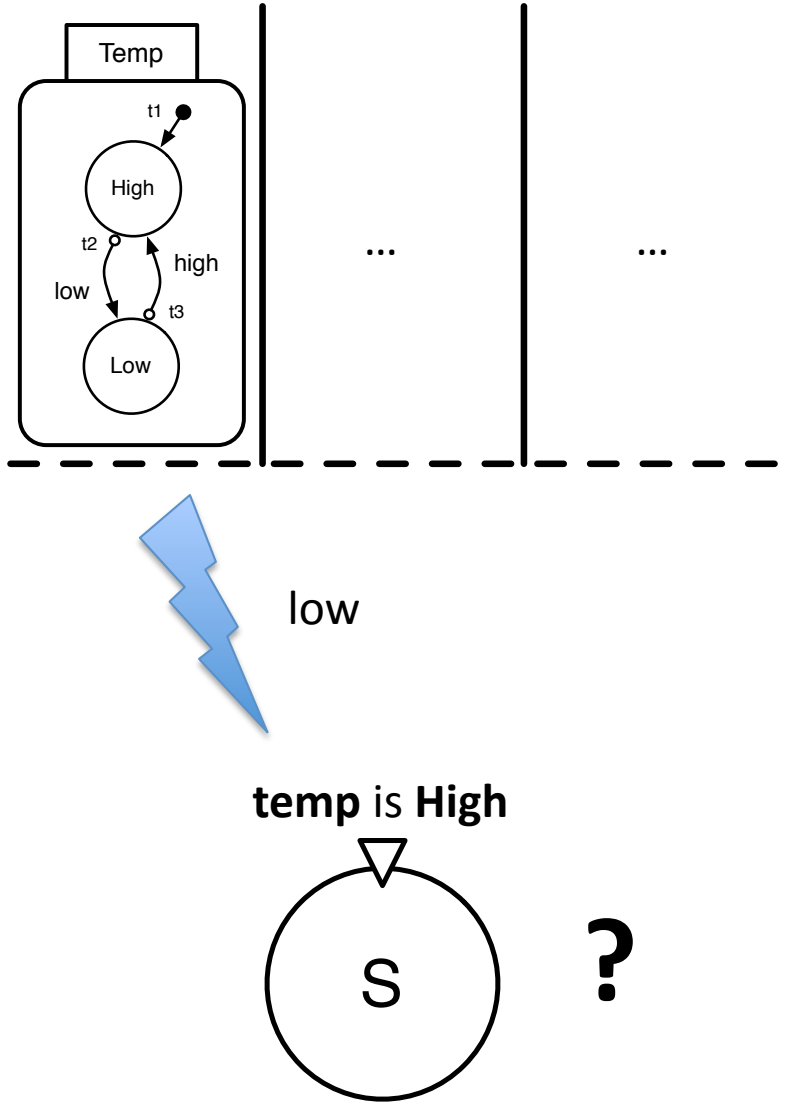
State semantics in PW Statecharts

- State semantics: each state S of the **whole** may be *formally* associated with a logical proposition sem(S) which *denotes* one or more global states of the **assembly** of components:
  - Example:
    (**valve** is **Closed** *and* **pump** is **Off**) *or* (**valve** is **Open** *and* **pump** is **On** *and* fan is **On**)
- When the system is in state S the assembly will be found in a global state which satisfies its state semantics.



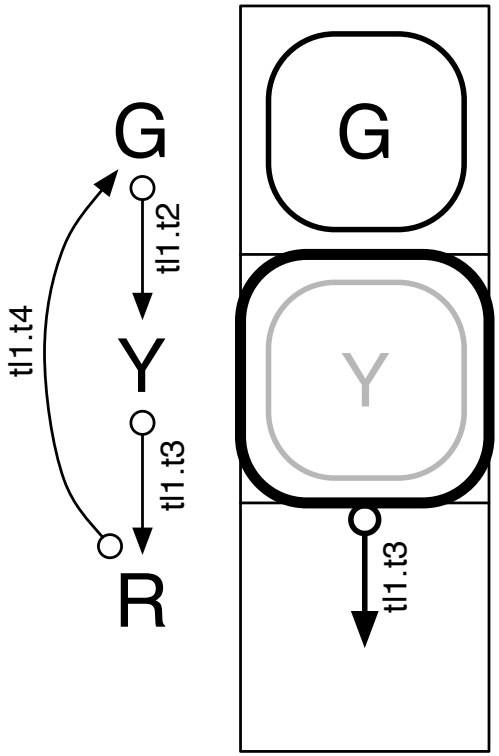
From State Semantics to State Invariants

- Determining State Semantics is not enough for guaranteeing **state invariants**;
- Components in the assembly may in fact undertake *autonomous transitions*, thus invalidating state invariants;
- What happens if there is a change of state which invalidates the state invariant?



State invariant exit conditions

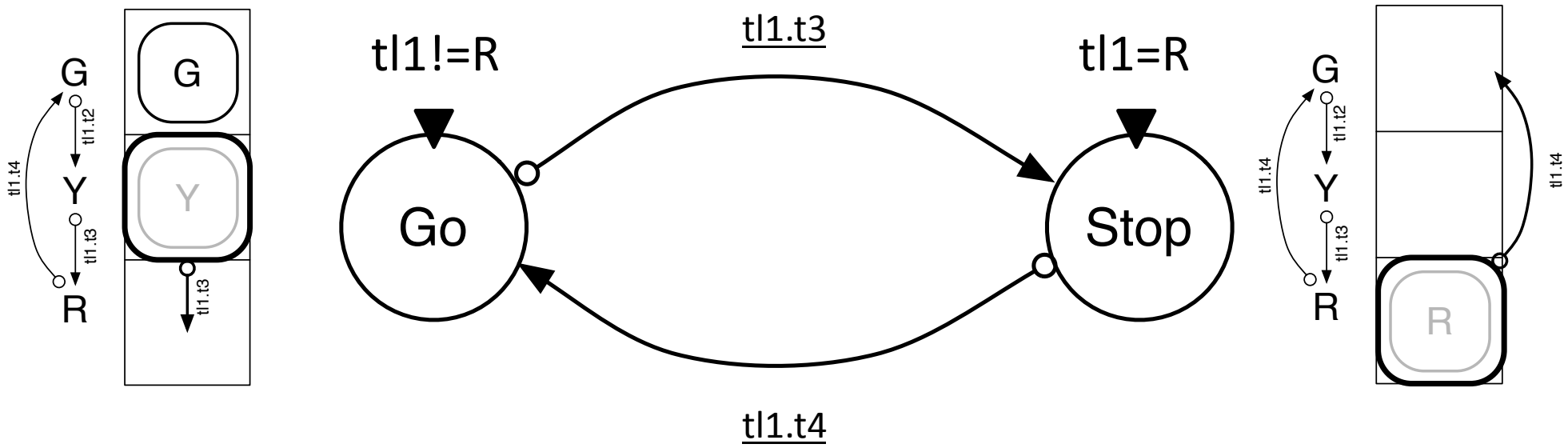
- The idea is to individuate *which part* of a state proposition may be invalidated by the happening of an *uncontrollable transition*;
- It is thus required to consider both the proposition and the state machine whose state transitions may invalidate it;
- The state invariant "tl != R" may be invalidated only when the traffic light is in state "Y" and transition t3 from Y to R happens;



traffic light is not **Red**

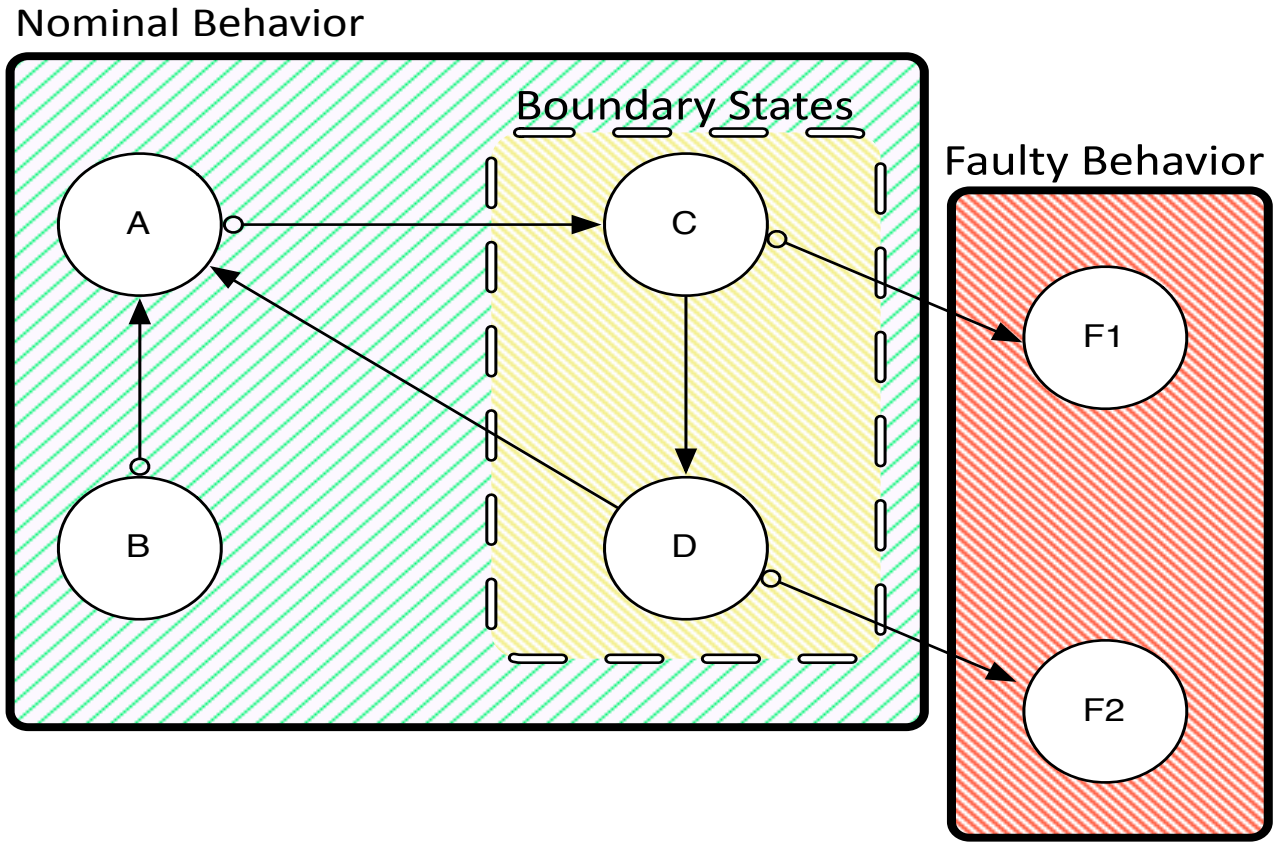
Exit conditions

- In order to keep the state invariants of state **Go** and **Stop** satisfied we have to insert state transitions which are able to react to the state changes which invalidate the respective state invariants.
- We say that such state transitions *cover* exit conditions.
- The integrity of the design requires to cover all the exit conditions in order to have the related state invariants always satisfied;



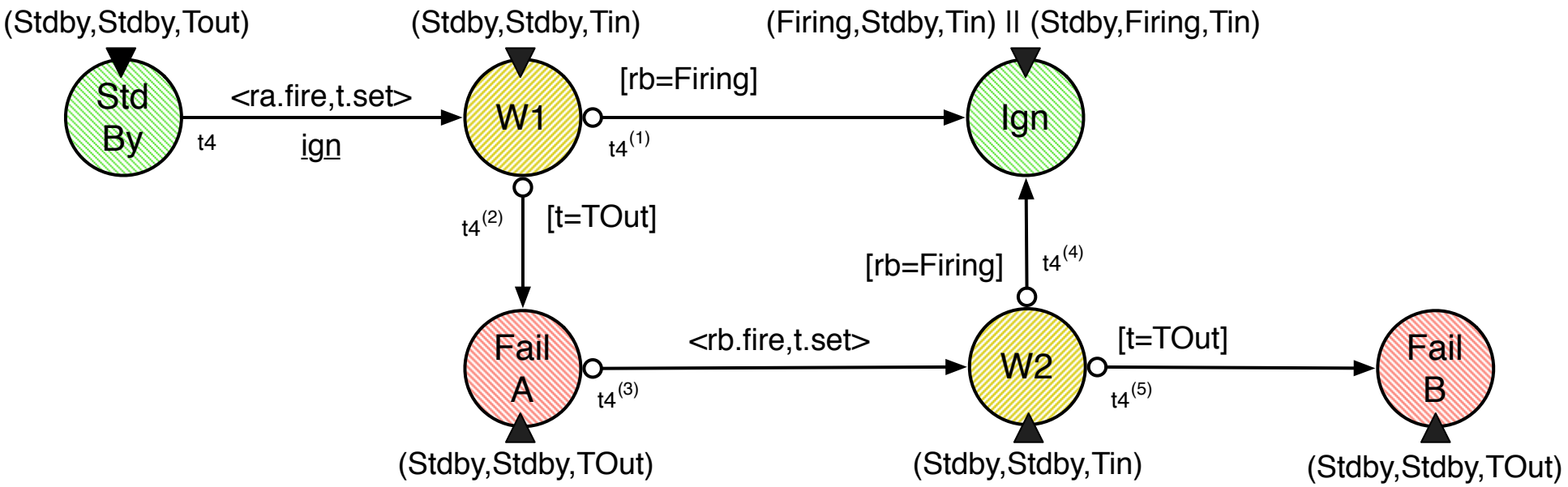
Boundary states

- Two sets of states:
  - **Nominal** where the system behavior is considered correct;
  - **Faulty** where the system behavior is considered wrong;
- Each state which covers an exit condition through a transition that ends in a state considered faulty can be considered as a **boundary** state;
- Working on the boundary states permits to increase the system fault tolerance.



From nominal to faulty behavior

- **Abnormal behavior** can be revealed by means of the algorithm that calculates the exit zones;
- Thanks to the exit zones algorithm **implicit behavior** emerges and can be managed;
- Possibility to model reacting or reconfiguring behaviors when some faults occur.



A look at the compositional flavour of the approach

- A system is **Fault Tolerant** once:
  1. it is assembled from FT components **AND**
  2. its components interact in a **fault tolerant** manner.

- Easy to enounce, difficult to realize;

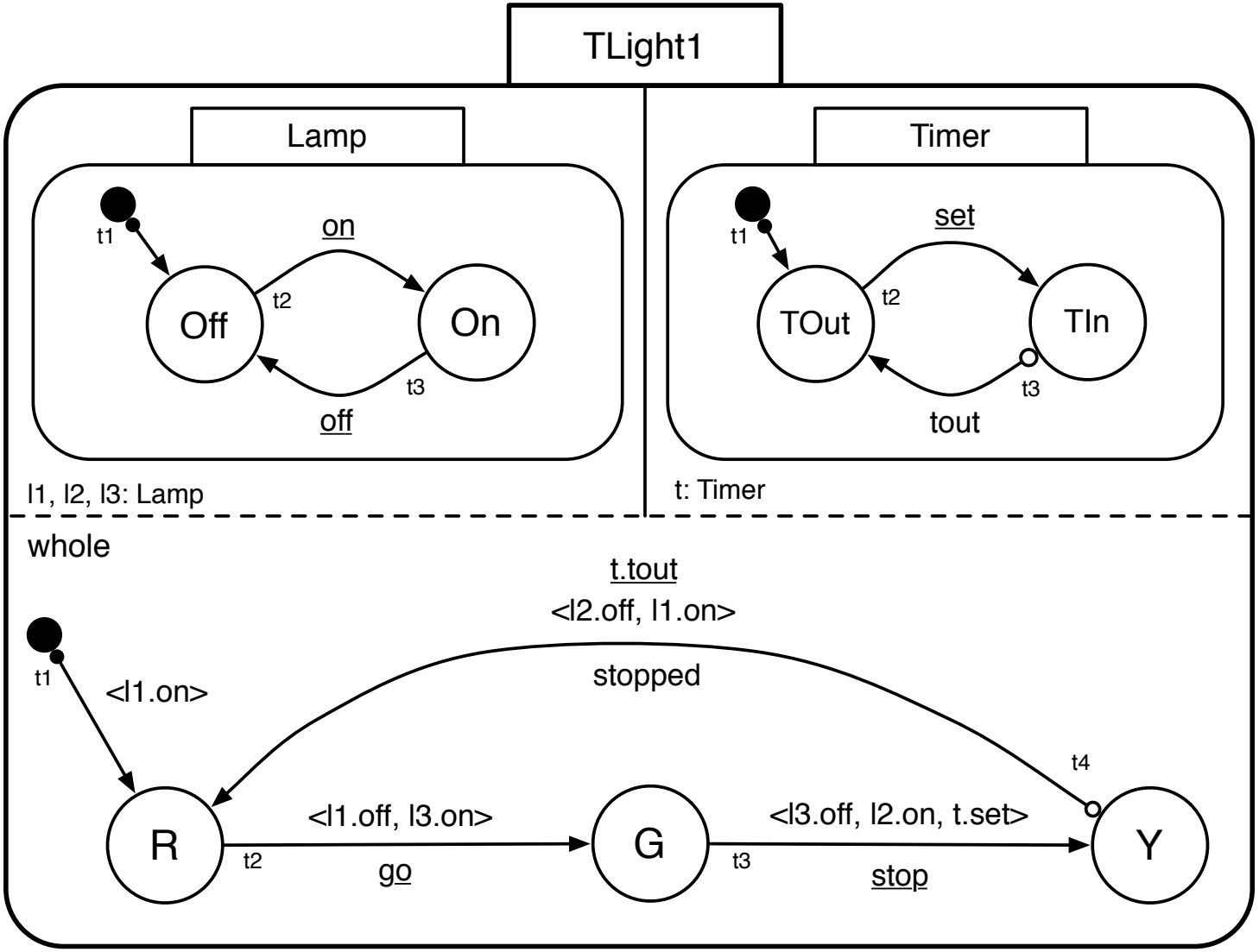
- PW Statecharts recursive composition satisfies both requirements;

Conclusions

- We propose a formalism which
  - gives Statecharts **modular constructs**;
    - benefits to Software Engineering Quality factors;
  - allows **state semantics / state invariants** to be specified at **design time;**
  - exit zones algorithm allows to **unveil abnormal behavior;**
  - allows to combine fault tolerant modules in order to create **system** that is **fault tolerant** .

- Thank you!

PWS recursive structure



Compositionally Checked Design