Aspect-Oriented Programming for Reactive Systems

Karine Altisen, Florence Maraninchi, David Stauch

Verimag

Aspect-Oriented Programming

– New approach in software engineering.
– Kiczales et al. Aspect-Oriented Programming. ECOOP'97

Aspect-Oriented Programming

- New approach in software engineering.
- Kiczales et al. Aspect-Oriented Programming. ECOOP'97
- Motivation : we cannot modularize all concerns in a object-oriented programming language. (some concerns "crosscut").
- Express functionalities outside the object structure in aspects.
- Implementation : AspectJ and others.

AspectJ Example

```
package pack;
class foo{
    public void visit(Object o){...}
     ...
}
 class bar{
    public String visit(String s){...}
    ... }

aspect logVisits {
    pointcut visits() :
        execution(public * pack.*.visit(..));
    before() : visits() {
        System.out.println(
            thisJoinPoint.getSignature().toString());
    }
}
```

We specify : – where we intervene : the pointcut
– what we do : the advice

Application to Reactive Systems

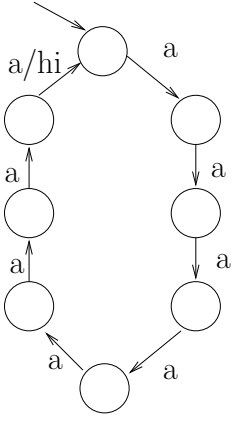
- **Idea : apply aspect-oriented programming to reactive systems.**
- **AspectJ and other approaches are little formalized.**
- **Impossible to use the existing tools : we need a formal semantics.**

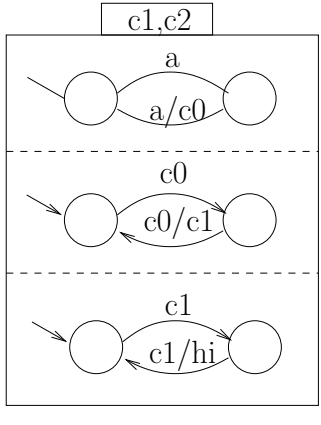
Parallel Structure

- **Synchronous languages have a parallel structure with broadcast.**
- **Parallel structure already powerful : “real” aspects.**
- **We want a simple language with a parallel structure.**
- **Subset of Argos.**

Synchronous Automata

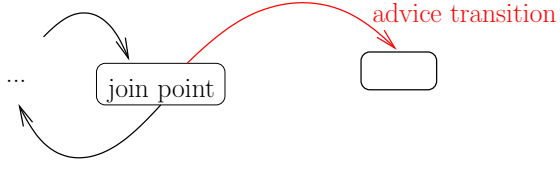
- **Formal framework for our propositions.**
- **Example modulo-8-counter.**
- **Operators : parallel product ($\|$) and encapsulation ($\setminus$).**
- **3-bit counter : $(B_1 \times B_2 \times B_3) \setminus \{b, c\}$.**





Aspects for Reactive Systems : a Proposal

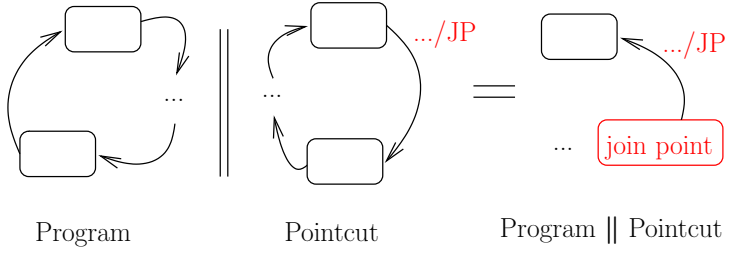
- New operators (with preservation of bisimilarity).
- Select states as join points.
- Advice : adding transitions :



- Two kinds of advice :
  - toInit
  - recovery

Pointcut

- **AspectJ pointcuts are syntax-based.**
- **We want something more semantic.**
- **Observer as pointcut.**
- **Mark joinpoints with a special output JP.**



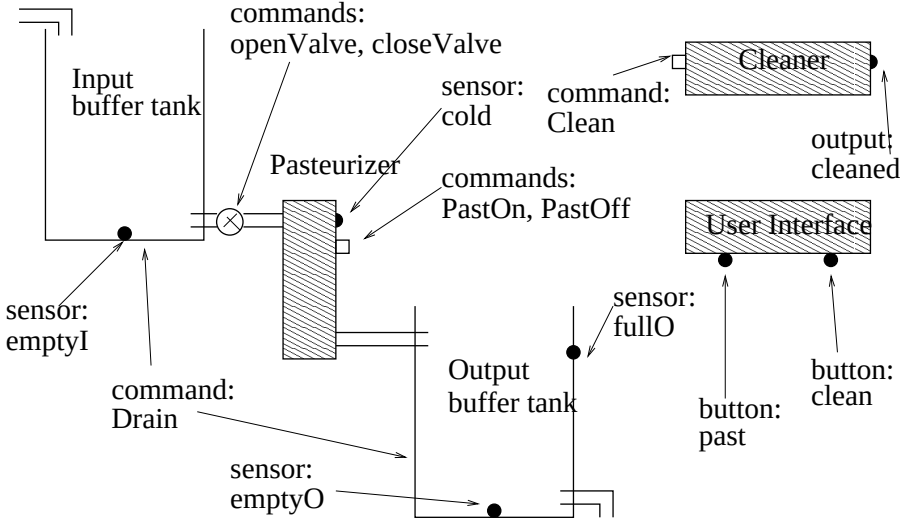
Proposition : toInit Aspect

- when an activation signal α is true, we go to a predefined state.
- α : boolean expression of fresh signals and inputs.
- Specification of target state :
  - trace σ on the inputs, starting in the initial state.
- Add outputs O to the new transitions.

- Example : reset (with empty trace).

Example : Pasteurizer

– **Juice processing plant.**



Controller

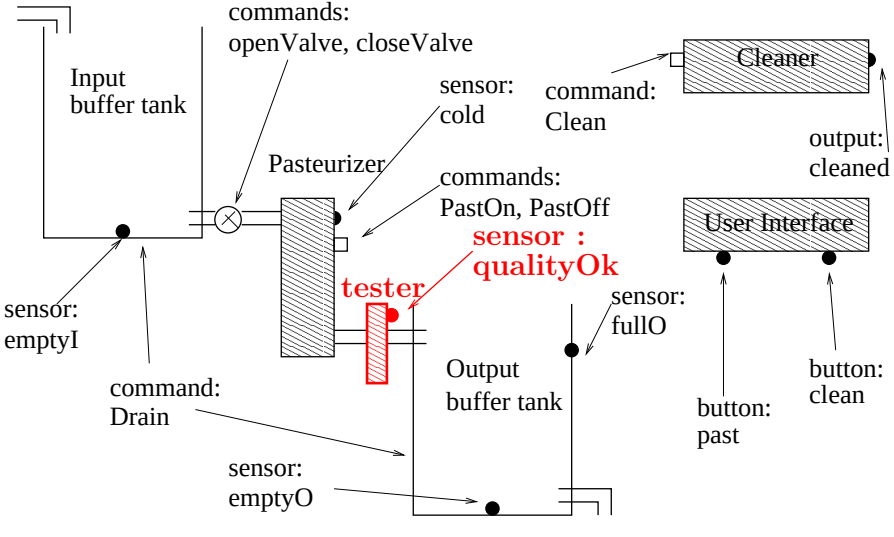
– **The pasteurizer can either pasteurize, or clean itself, or it is stopped.**

	Buttons and sensors :
clean	clean the pasteurizer (button)
past	pasteurize juice (button)
emptyI	the input buffer tank is empty
emptyO	the output buffer tank is empty
fullO	the output buffer tank is full
cold	the pasteurizer is cold
cleaned	the cleaner has finished cleaning

	Commands :
Clean	make the cleaner work
PastOn	switch the pasteurizer on
PastOff	switch the pasteurizer off
openValve	open the valve between the input buffer tank and the pasteurizer
closeValve	close the valve between the input buffer tank and the pasteurizer
Drain	empty the tanks

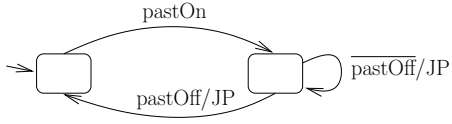
Pasteurizer – the Aspect

– **A quality controller must be added.**



Pasteurizer – the Aspect

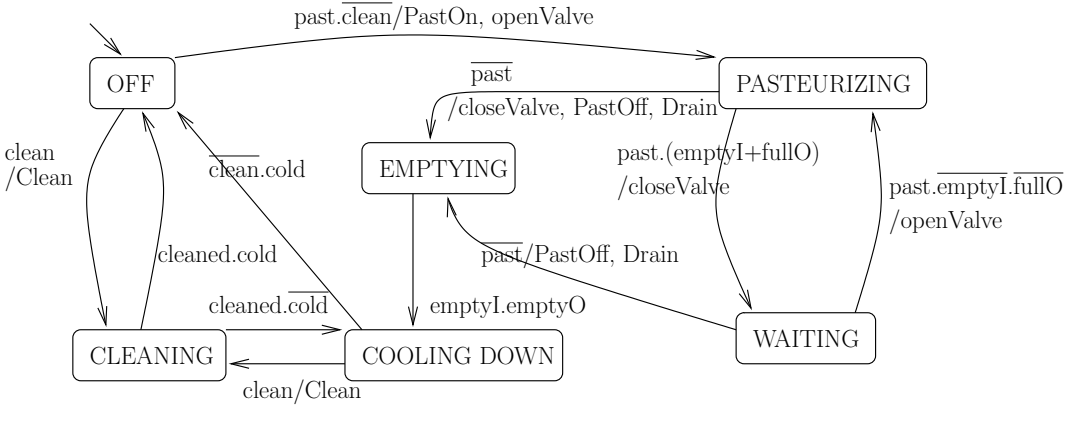
- **When the quality is insufficient, clean the pasteurizer.**
- **Pointcut : we pasteurize, between `pastOn` and `pastOff`.**



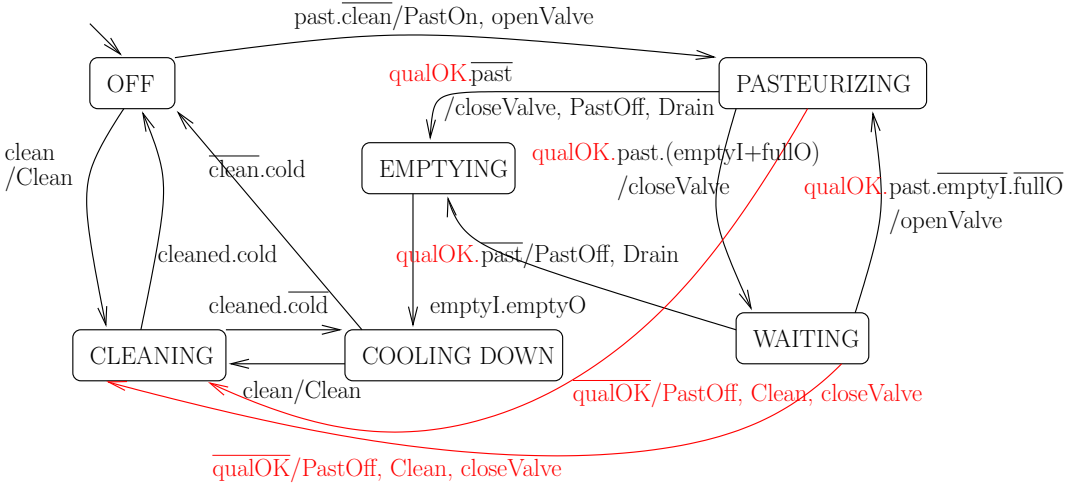
PC automaton

- **Advice : switch off the pasteurizer, close the valve, start cleaning, and go to the state that corresponds to cleaning.**
  - **emit PastOff, Clean and closeValve**
  - **trace = clean.$\overline{\text{past}}$**

Pasteurizer – an Implementation



An Implementation – Aspect



Proposition : Recovery Aspects

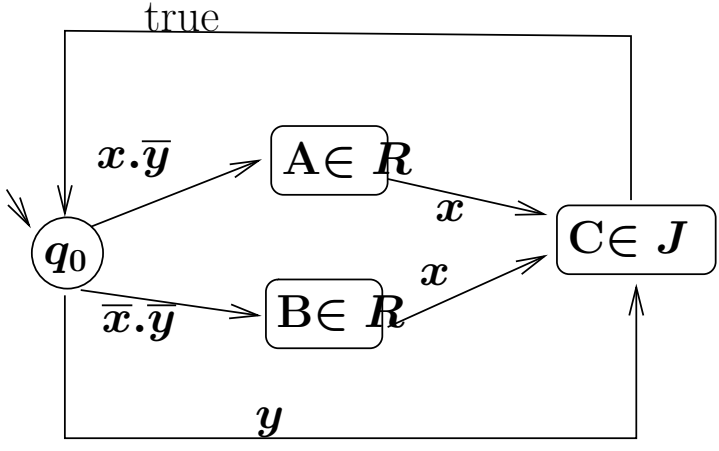
- **Jump backwards.**
- **we define certain states as “recovery states”.**
  - **specification similar to the pointcut’s.**
- **When α is true, go back to the last recovery state passed.**

Recovery Aspect Weaving

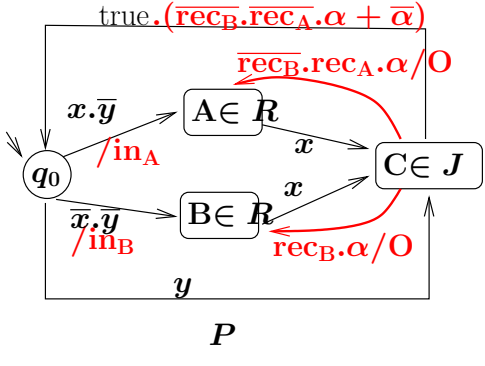
- **Automaton $\mathcal{M}$ memorizes which recovery was passed last, executes in parallel.**
- **Signals $\mathcal{I}$n : tell $\mathcal{M}$ when the program enters a recovery state.**
- **Signals $\mathcal{R}$ec : tell the program which recovery state was passed last.**

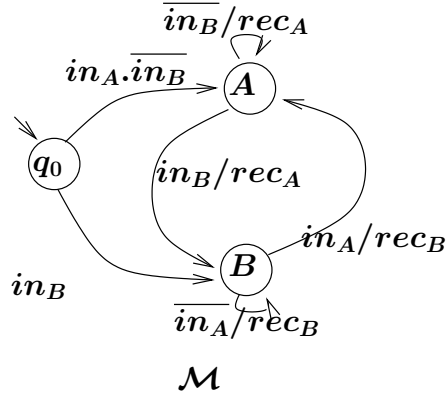
Example – Recovery Weaving

– A,B : recovery states (R)
– C : join point(J)



Example – Recovery Weaving



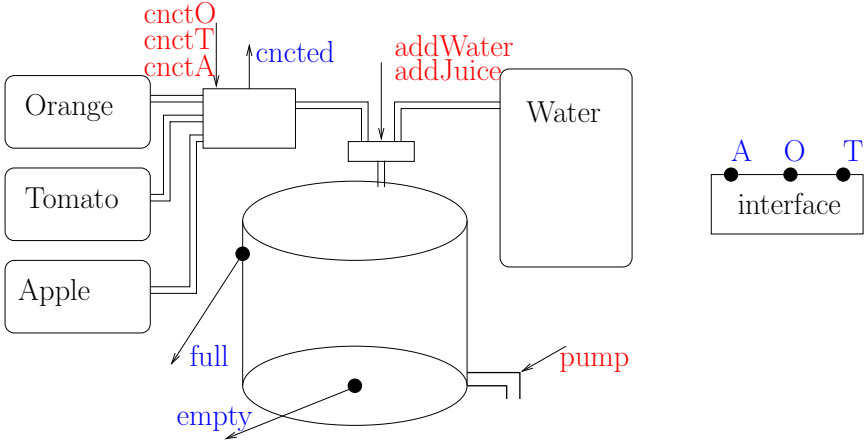


– **Program after weaving :**

$$P \| \mathcal{M} \setminus \{in_A, in_B, rec_A, rec_B\}$$

Example : Blender

- **Blends water with three juice concentrates to produce fruit juice.**
- **3 juices with different water/concentrate relations.**

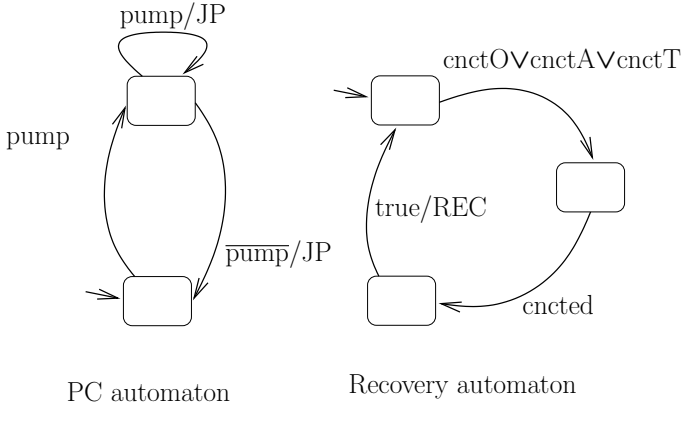


Blender – the Aspect

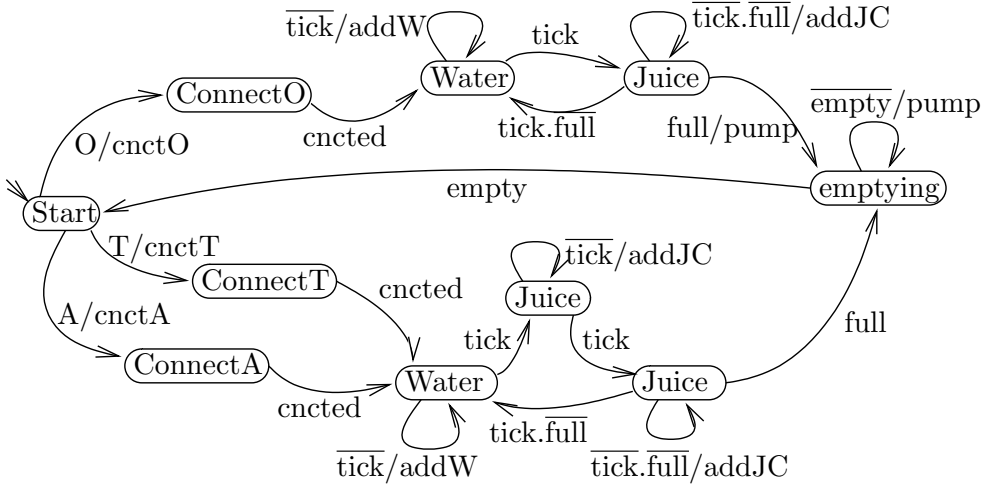
- When the tank has been emptied, the user must specify which juice to produce next.
- The concentrate tube is then reconnected (expensive).
- Aspect : new input `restart`
  - activation signal : `restart.empty`.
  - applies when the tank has just been emptied.
  - the tube is not reconnected.
  - continue to produce the current juice.

Blender – the Aspect

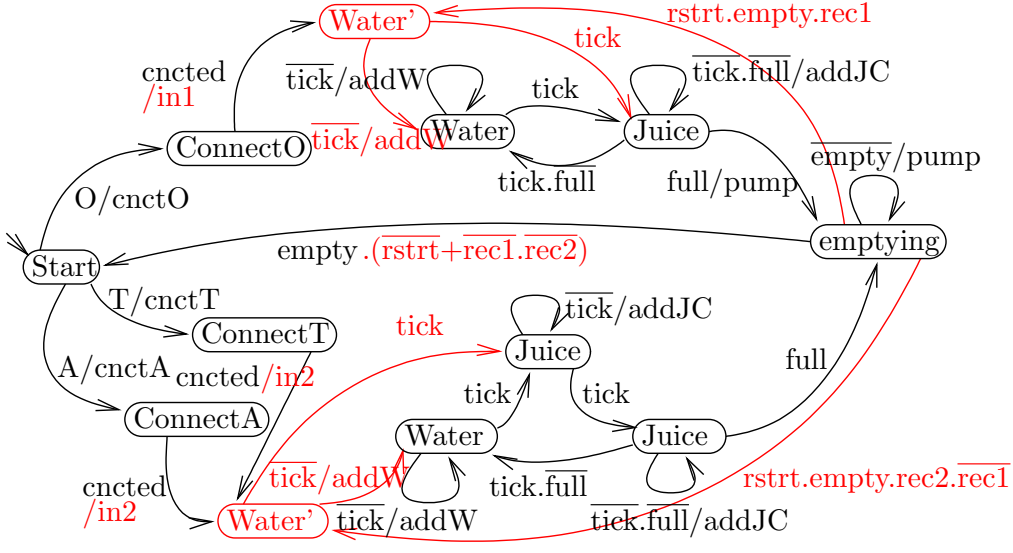
- **Pointcut : when emptying, i.e. when `pump` is true.**
- **Recovery : Just after the reconnection of the pipe, i.e. when `cncted` is true after `cnctX` has been emitted.**



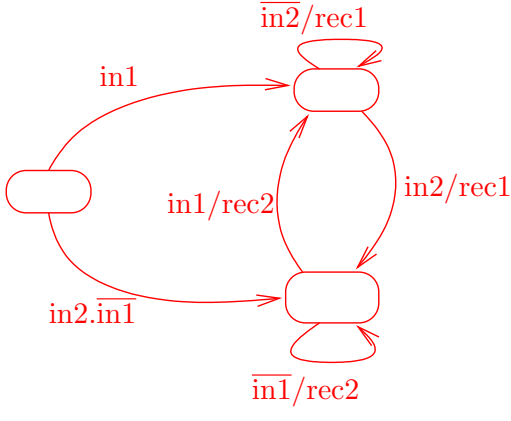
Blender – an Implementation



An Implementation – Aspect



An Implementation – Aspect

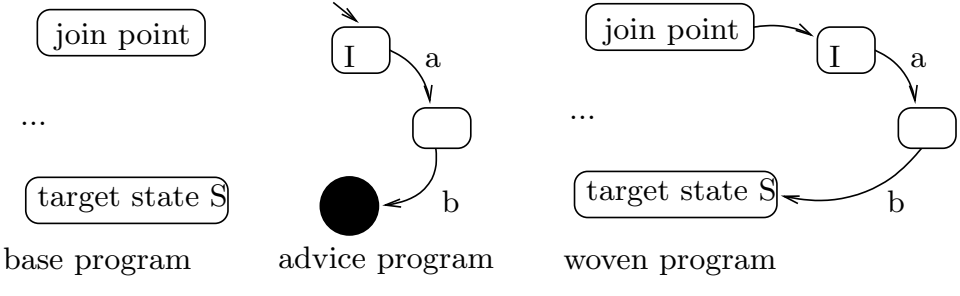


Pasteurizer – Extension

- Extension : cleaner also cleans tanks.
- New requirement : tanks must be emptied before cleaning.
- Aspect must empty tanks :
  - emit Drain to empty tanks.
  - wait for tanks to be empty, and clean.
- Cannot be done with existing aspects !

Extension – Advice Program

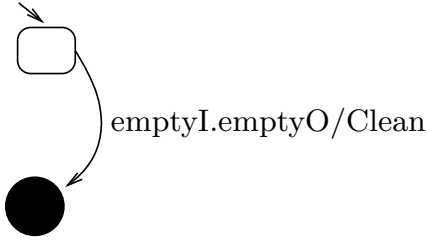
- **Idea : include automaton on advice transitions.**
- **Specify advice program : automaton with special final state.**



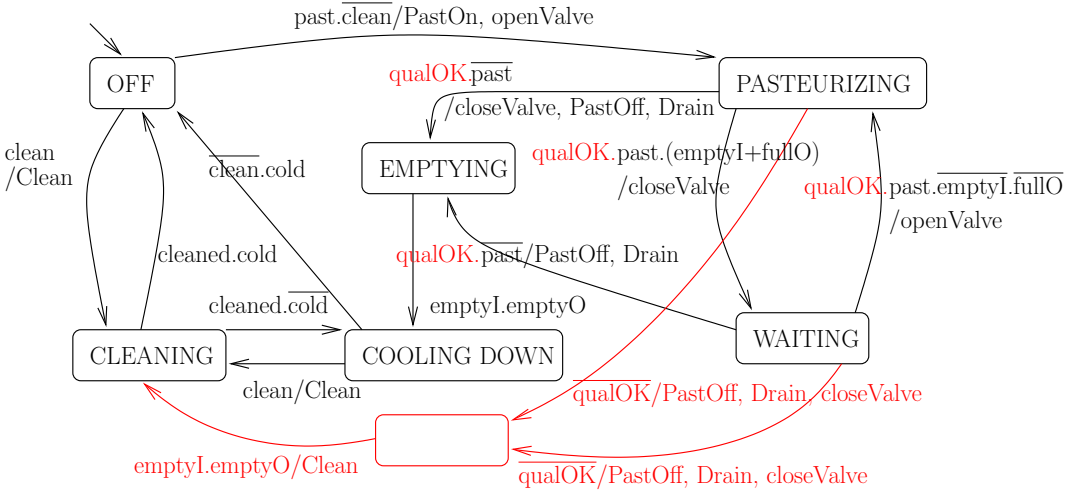
base program advice program woven program

Pasteurizer – Advice Program

- **Poincut, target state and activation condition unchanged.**
- **Outputs : Drain, PastOff, closeValve**



Pasteurizer – Aspect Application



Conclusion

- Aspect language for Argos.
- New operator for Argos.
- Preserves trace equivalence, determinism, reactivity.
- Implementation exists.
- Perspectives :
  - Study other kinds of advice.
  - What is the effect of an aspect on a program ?
  - Do we need aspects ?