PHAVer

Dott. Luigi Di Guglielmo

Prof. Tiziano Villa

University of Verona
Dep. Computer Science
Italy

Outline

- Introduction
- Hybrid Automata
- Modeling in PHAVer
- Verification in PHAVer
- Examples
- How to use PHAVer

Introduction (I)

- PHAVer (*P*olyhedral *H*ybrid *A*utomaton *V*erifier) is a tool for verifying affine hybrid automata
- It has been developed by Goran Frehse, member of the Hybrid Systems Group at the University of Grenoble (France)

Introduction (II)

- Systems with discrete as well as continuous dynamic, i.e., hybrid systems, are complex to analyze
- The verification of hybrid systems is a challenging problem from both the theoretical and experimental sides

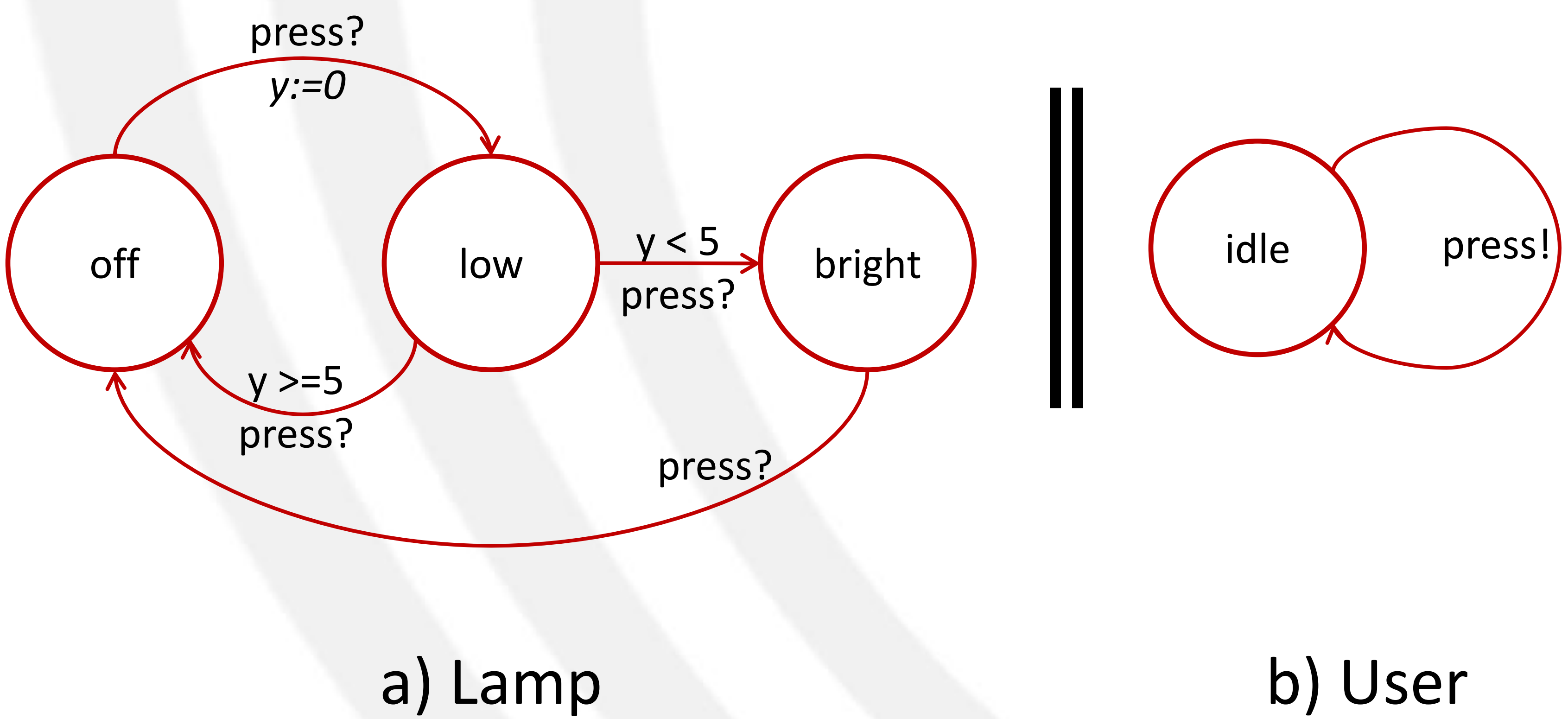
Hybrid Automata (I)

- From the theoretical side, *Hybrid Automata* have been proposed as a formal model for the design and verification of hybrid systems
- The formal model allows to
  - Model temporal and functional aspects of both discrete and continuous components
  - Compose the different automata generating a whole system that can be formally verified

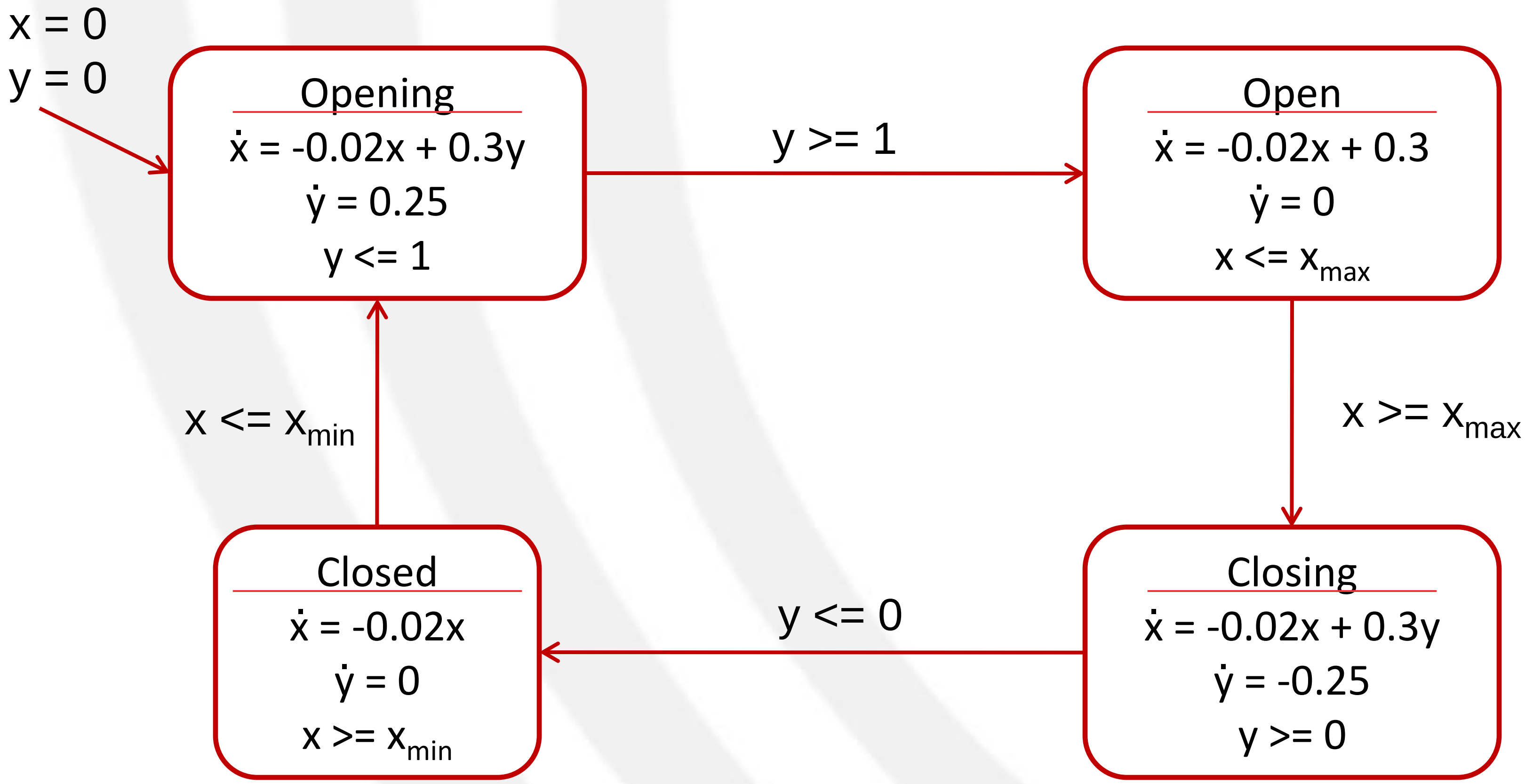
Hybrid Automata (II)

- Different classes of Hybrid Automata
  - Timed Automata
    - Continuous variables
      - Continuous dynamics are stated by the constant "1"
        - » dot(x) = 1
      - vars can be updated (i.e., reset) only to zero
  - Linear Hybrid Automata (LHA)
    - Continuous variables
      - Continuous dynamics must follow a rectangular predicate
        - » dot(x)∊ [`l`,`u`], `l` and `u` are rational constants
      - vars can be updated non deterministically to any value satisfying a rectangular predicate
  - Affine Hybrid Automata (AHA)
    - Continuous variable
      - Continuous dynamics are stated by linear differential equations
        - » dot(x) = A x + b
      - vars can be updated with a value given by a linear function
  - Hybrid automata
    - Continuous variable
      - Continuous dynamics can be modeled by any type of differential equation
      - Complex update relation

Timed Automata Example



Affine Hybrid Automaton Example



PHAVer

- PHAVer computes the set of reachable states of a network of affine hybrid automata
- The identified set of reachable states allows to verify whether the set of reachable states intersects a target region (reachability) or intersects a bad region (safety)

MODELING IN PHAVER

Modeling in PHAVer (I)

- PHAVer does not provide a graphical user interface
- The network of hybrid automata must be described into a text file given as input to PHAVer
- The syntax is simple and intuitive

Modeling in PHAVer (II)

- Comments
  - Are preceded by either **//**, **--**, or enclosed in **/* */**
- Identifier
  - Is a letter plus any combination of letters, digits and _ (underscore)
- Number
  - Can be given in floating point format
    - E.g., 3.14 or 6.28e-32
  - Or as a fraction
    - E.g., 9/5
- Constants
  - Are defined in the form
    `identifier := expression;`

Modeling in PHAVer (III)

- Linear expression
  - Is defined over a set of variables, numbers, constants that can be combined using +,-,/,*,(,)
  - It is not allowed to multiply two variables or divide by a variable
- Linear constraint
  - Is the combination of two linear expressions with one of the signs <, >, <=, >=, ==
- Convex linear formula
  - Is given as a conjunction of linear constraints that are joined by & (ampersand)
- Linear formula
  - Is a disjunction of convex linear formulas joined by | (pipe)

Modeling in PHAVer (IV)

- Symbolic state
  - Is a combination of a location name and a linear formula joined by & (ampersand)
    - E.g., (opening) & (x >= 0 & y <= 1)
- Set of symbolic states
  - Is a list of symbolic states joined by , (comma)
- Wildcards
  - *identifier* = *aut* `.{$ & true};`

Modeling in PHAVer (V)

```
automaton aut
   contr_var:  var_ident, var_ident, ... ;
   input_var:  var_ident, var_ident, ... ;

   synclabs:  label_ident, label_ident ... ;
   loc loc_ident: while invariant wait { derivative };
        when guard sync label_ident do {trans_rel} goto loc_ident;
        when guard sync label_ident goto loc_ident;
        when ...
   loc loc_ident:   while ...
   initially:  initial_states;
end
```

Modeling in PHAVer (VI)

- `contr_var`
  - State variables whose dynamics are specified by the automaton
- `input_var`
  - Variables whose dynamics are not specified by the automaton
- `synclabs`
  - Synchronization labels

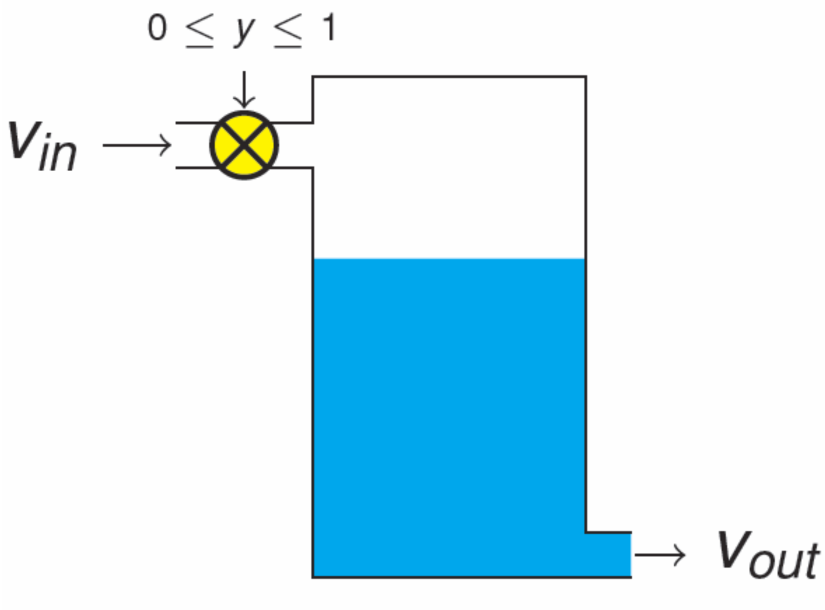
Modeling in PHAVer (VII)

- *invariant* and *guard*
  - Linear formulas over the state variables (i.e., `contr_var`), input variables and parameters
- *derivative*
  - For <u>piece-wise constant dynamics</u>, it is a convex linear formula over the state variables derivatives
    - E.g., 0 <= x' & x' < 1   stands for $\dot{x} \in [0,1)$
  - For <u>affine dynamics</u>, it is a convex linear formula over the variables and their derivatives
    - E.g., x' == -2 * x   stands for $\dot{x} = -2x$

Modeling in PHAVer (VIII)

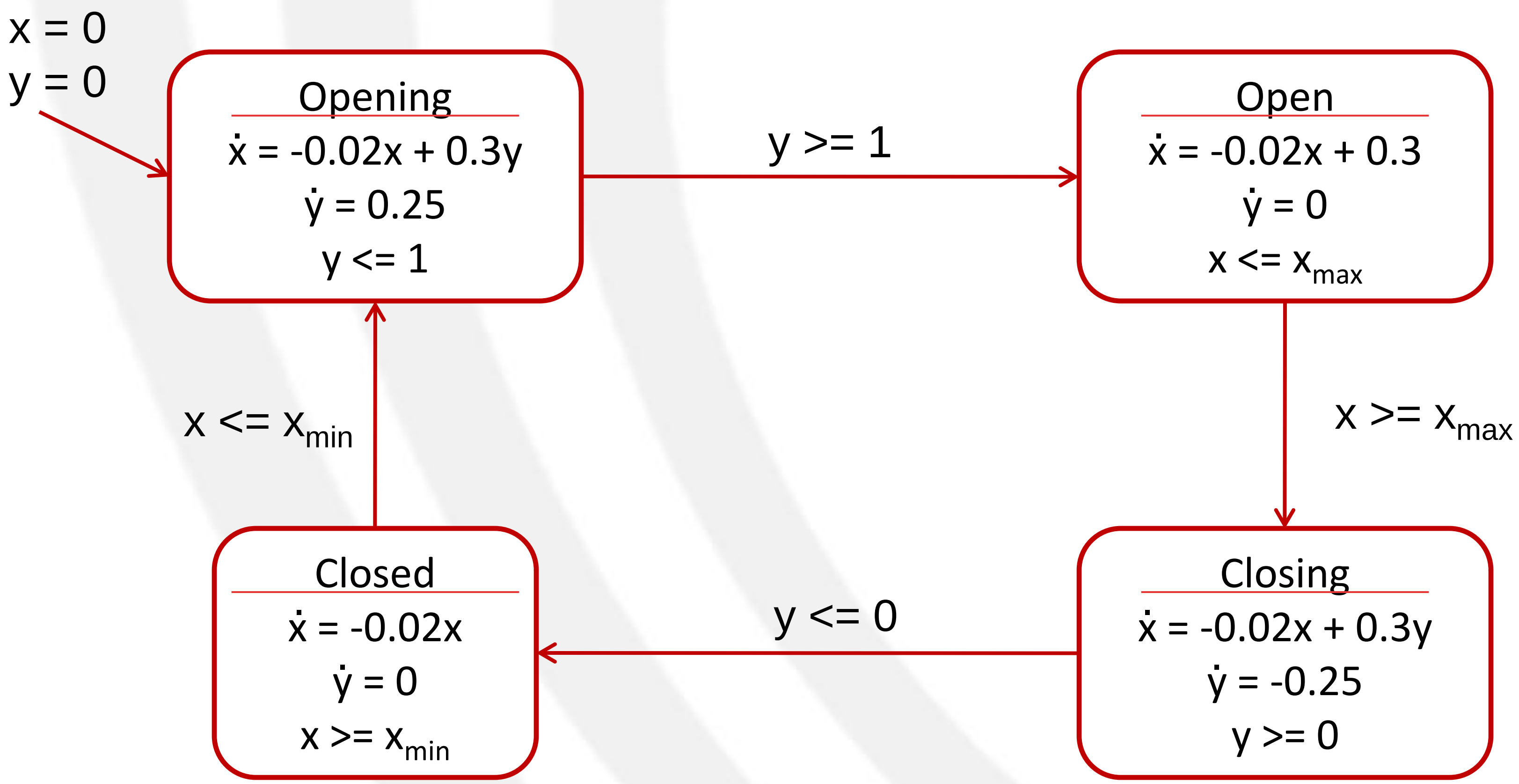
- Transitions are specified by `when .. goto` statements
  - Synchronization label must be specified
  - Resets may be specified by `do` *trans_rel*
- *trans_rel* is a linear formula
  - Post-transition value of a state variable is indicated by ‘ (single quote)
    - E.g, z’ == 0
  - State variables not changed by the transition must be specified explicitely
    - E.g., x’ == x & y’ == y
- *initial_states* is a set of symbolic states

Example: the Water-tank System (I)



- `y` specifies the valve aperture
  - It ranges within 0 and 1
- `x` describes the liquid level
- The water flows from the tank as $v_{out} = -0.02x$
- The water arrives into the tank as $v_{in} = 0.3y$
- At the beginning the tank is empty and the valve is opening

Example: the Water-tank System (II)



Consider $x_{min} = 5.5$ and $x_{max} = 8.5$

Example: the Water-tank System (III)

```
Xmin :=5.5;
Xmax :=8.5;
// automaton definition
//
automaton tank
contr_var: y, x; //y valve aperture
                 //x water level
synclabs: jump;
loc opening: while y<=1 wait
    {x'==-0.02*x + 0.3*y & y'==0.25}
  when y>=1 sync jump do {x'== x & y'==y} goto open;

loc open: while x<=Xmax wait
    {x'==-0.02*x + 0.3 & y'==0}
  when x>=Xmax sync jump do {x'==x & y'==y } goto closing;

loc closing: while y>=0 wait
    {x'==-0.02*x + 0.3*y & y'==-0.25}
  when y<=0 sync jump do {x'==x & y'==y} goto close;

loc close: while x>=Xmin wait
    {x'==-0.02*x & y'== 0}
  when x<= Xmin sync jump do {x'==x & y'==y}
goto opening;

initially: opening & x==0 & y==0;
end
```

COMMANDS

Reachability Analysis (I)

- *state_ident* = *aut_ident*.**`reachable`**`;`
  - Returns the set of states reachable in the automaton *aut_ident* from its initial states
  - This set of reachable states is stored in *state_ident*
- *state_ident1* = *aut_ident*.**`reachable`**`(`*state_ident2*`)`;
  - Returns the set of states reachable in the automaton *aut_ident* from the states in *state_ident2*
  - This set of reachable states is stored in *state_ident1*

General

- *Identifier*.**`print`**([“*file_name*”][, method]);
  - If *file_name* is specified, writes a representation of *identifier* to the file file_name, otherwise to the standard output
  - Method is a number that specifies the output format
    - 0 (default): textual description of location names and linear formulas readable by PHAVer
    - 2: format compatible with plotting tools like *gnuplot*

Example: the Water-tank System (III)

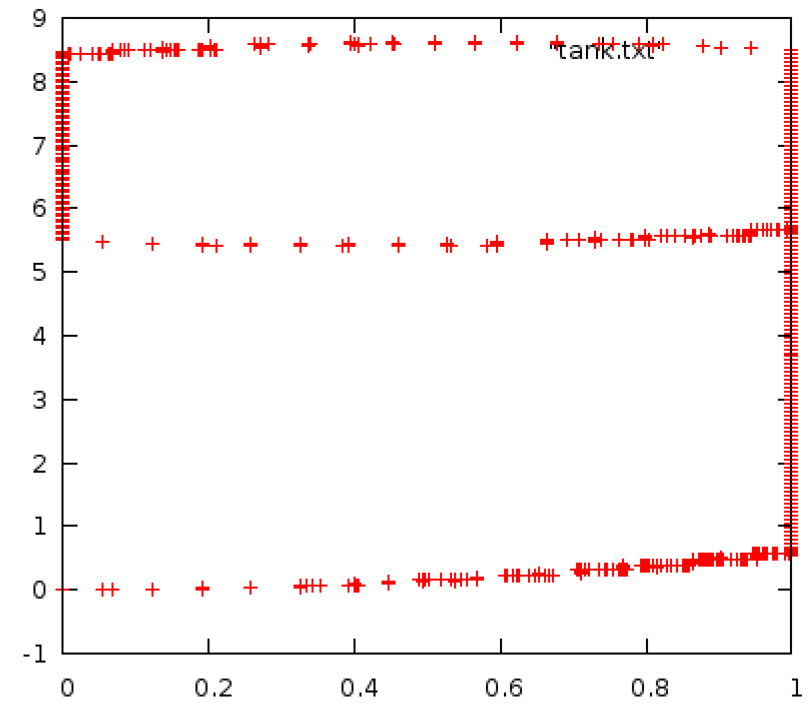
```
// grid definition
//
tank.add_label(tau);
tank.set_refine_constraints
     ((x, 0.08),(y,0.08),tau);
//
//
//
region=tank.reachable;
//
// output file
//
region.print("tank.txt",2);
```

How to use PHAVer

- PHAVer can be executed by typing in
```
#> ./phaver tank.pha
```
- The generated .txt file can be viewed with *gnuplot*
```
#> gnuplot
gnuplot> plot “tank.txt”
```

VERIFICATION IN PHAVER

Reachable Regions



- x (i.e., the water level)
- y (i.e., the valve aperture)

Safety Problem (I)

- The safety problem in PHAVer can be seen as a reachability problem over a set of bad states
- Once the set of bad states is defined, it is possible to ask PHAVer to compute if such a set of states is reachable from the set of initial states

Safety Problem (II)

- *state_ident1* = *aut_ident*.**`is_reachable`**`(`*state_ident2*`);`
  - Computes the set of reachable states, but stops as soon as a state in *state_ident2* is found.
  - Returns the states that were found to be reachable at the time of termination

Safety Problem (II)

- *identifier1*`.intersection_assign(`*identifier2*`);`
  - Intersects *identifier2* with *identifier1* and puts the result into *identifier1*
- *identifier*`.is_empty;`
  - Writes whether the object *identifier* is empty to the standard output

Safety Problem (III)

```
// safety verification
// defining bad states

forbidden = tank.{$ & x >= 8.7};

safe_region=tank.is_reachable(forbidden);

safe_region.intersection_assign(forbidden);

echo "the intersection between the reachable region and the bad reagion is:";
safe_region.is_empty;
```

Safety Problem (IV)

```
(disk)dgglgu08@delta010:~/sistemi/lesson03$ phaver tank.pha

  Parsing file tank.pha.
  ----------------------
      ...............................................
      ...............................................
      ...............................................
      ...............................................
      ..........

  Printing symb. states in 405 locations in generator floating point raw format.
  ------------------------------------------------------------------------------
      ...............................................
      ...............................................
      ...............................................
      ...............................................
      ..........
forbidden not reachable.
the intersection between the reachable region and the bad reagion is:
empty
Finished. Exiting.
```