CS 498: Bachelor's Thesis Project Evaluation: Phase-1

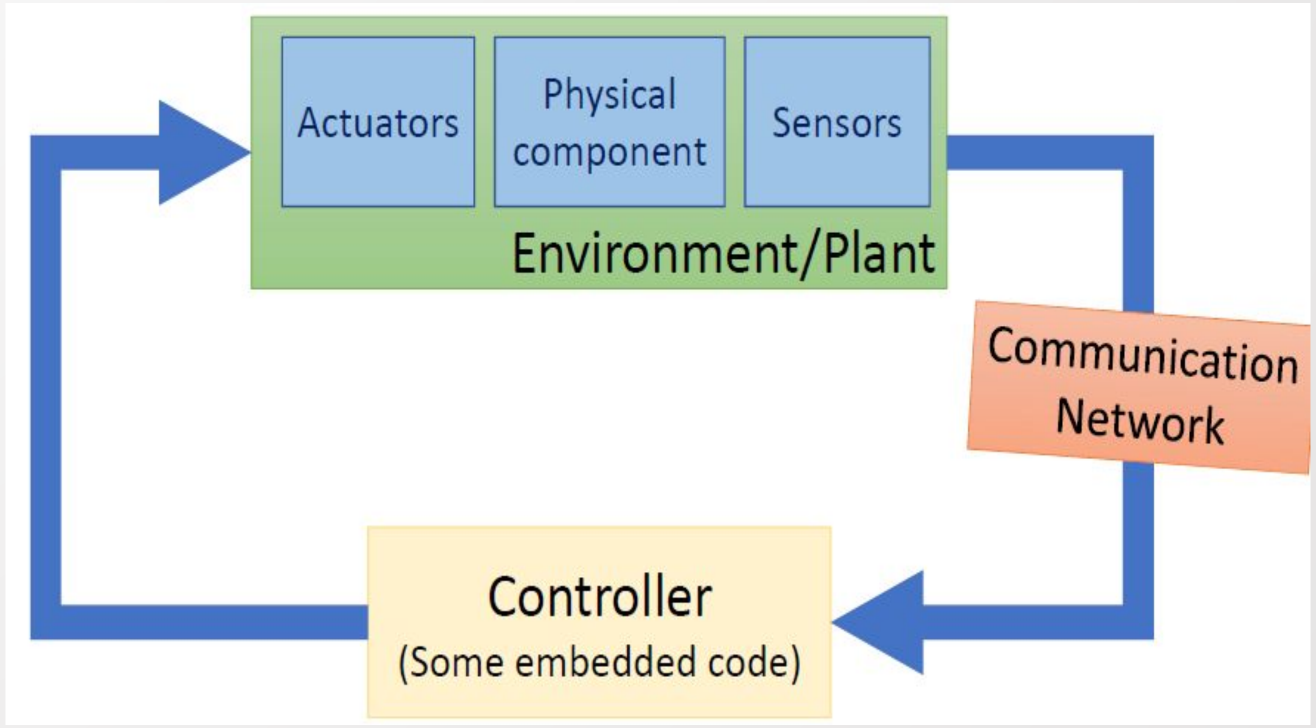
Shield Synthesis for Cyber-Physical Systems

Presented by:
Samay Varshney (180101097)
Siddhartha Jain (180101078)

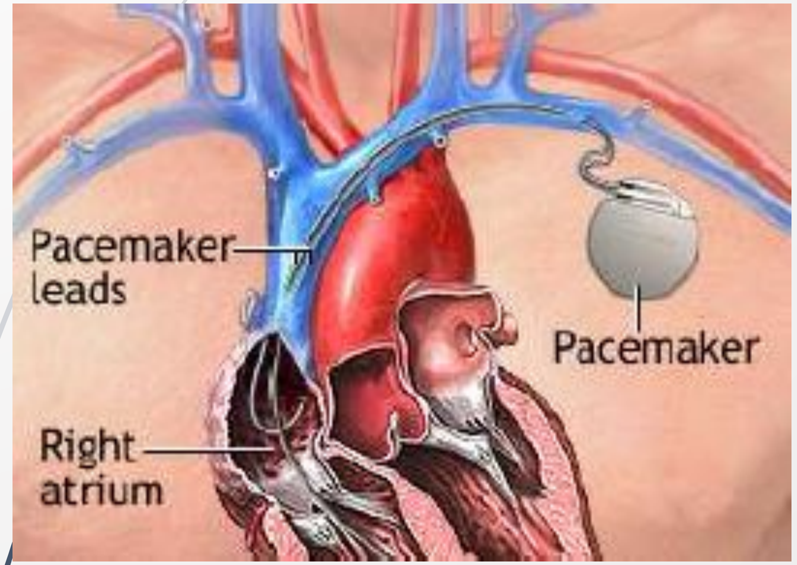
Cyber Physical Systems

What is a Cyber Physical System?

- A mechanism controlled or monitored by software algorithms.
- Typical architecture of cyber physical systems.
- Major 2 components: cyber and physical.

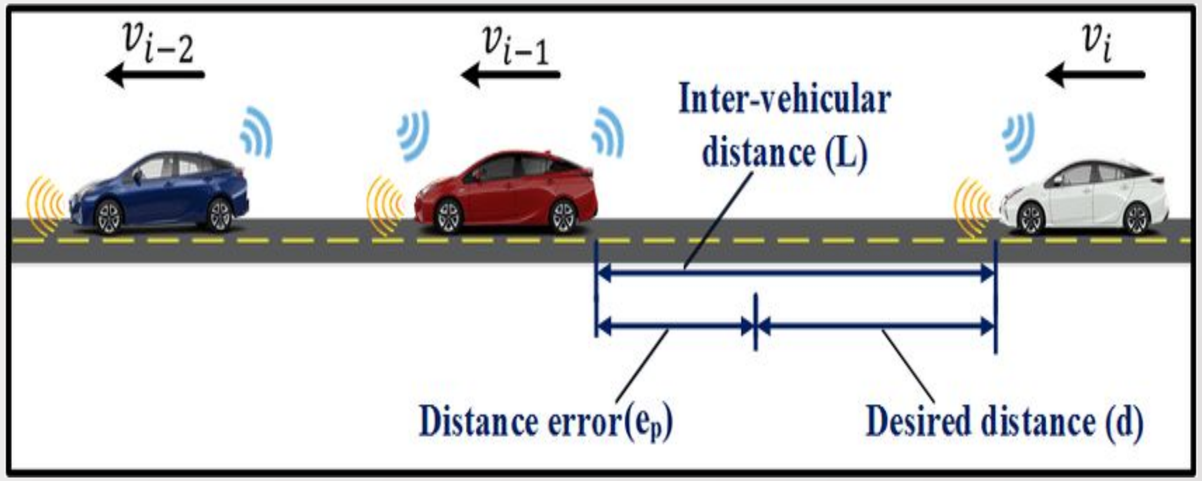


Examples of Cyber Physical Systems



Heart Pacemaker

Drone



Platooning of cars

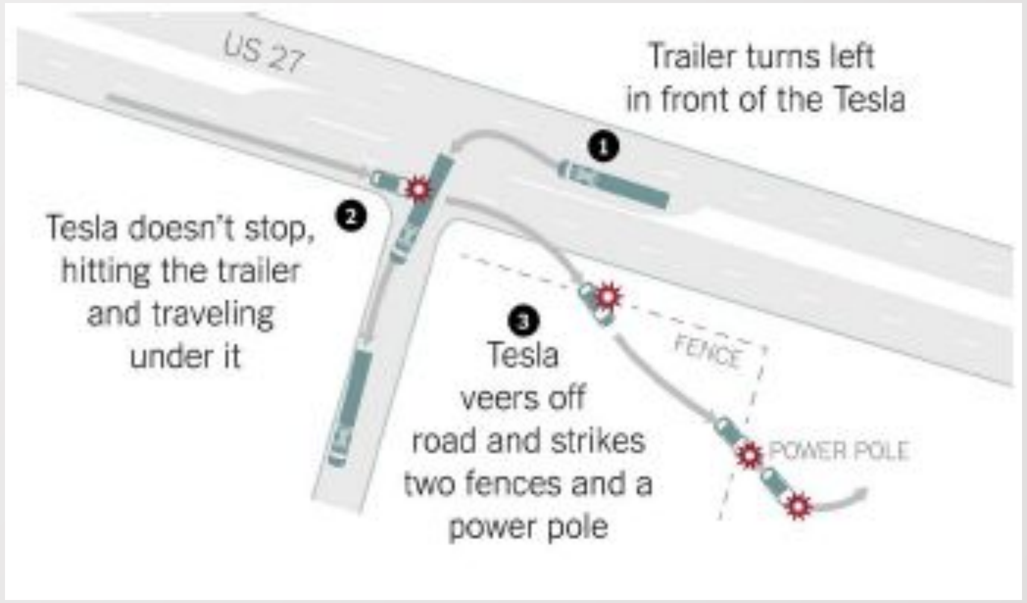
Need for Shield Synthesis (Runtime Enforcement)

Runtime Enforcement

- A technique to monitor and correct system execution at runtime.
- Enforces the system to satisfy some desired properties (a set of formal requirements).

Formal verification not realistic always:

- Too large or complex
- Models not available
  (eg. machine learning systems)


US 27
Trailer turns left in front of the Tesla
1
2
Tesla doesn't stop, hitting the trailer and traveling under it
3
Tesla veers off road and strikes two fences and a power pole
FENCE
POWER POLE

Solution: Shield Synthesis

- Implement a shield to enforce critical properties.
- Violations are not only detected but also overwritten.
- **Shield consider controller as a blackbox so verifying shield is easy.**
- Most of the real life examples are based on reactive systems.
- Hence, a need for enforcing the properties which needs to be followed at run time.
- For example, pacemakers are life-saving devices. But, these have caused serious harm including death of many patients.

Timed Properties

Timed Properties: Property with timed constraints.

Examples in Heart-Pacemaker CPS:

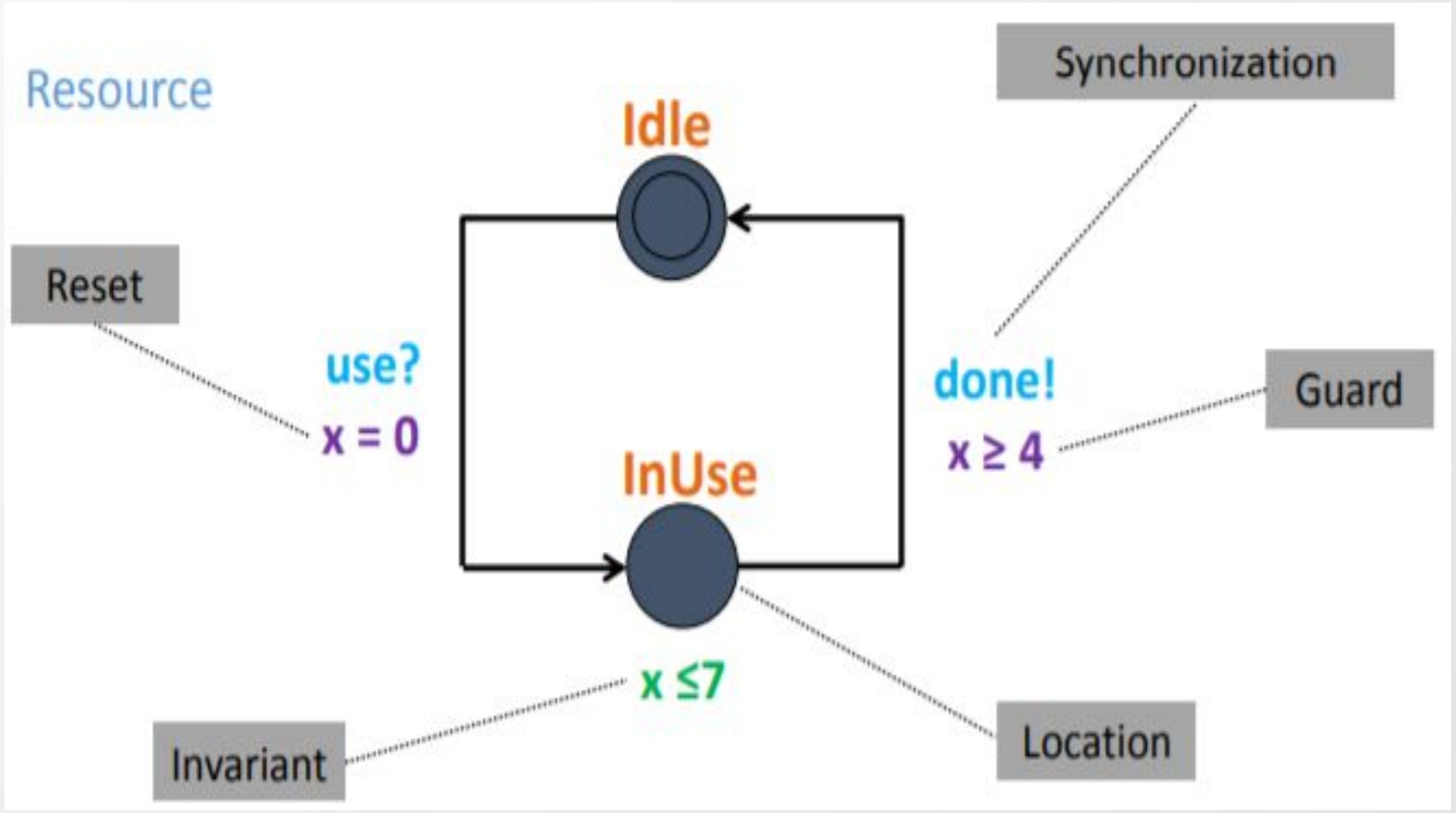
- Atrial Pace (AP) and Ventricular Pace (VP) cannot happen simultaneously.
- After a ventricle event, another ventricle event can happen only after URI.

Our focus in this work: Implementing shield for timed properties using Timed Automata.

Timed Automata

Timed Automata

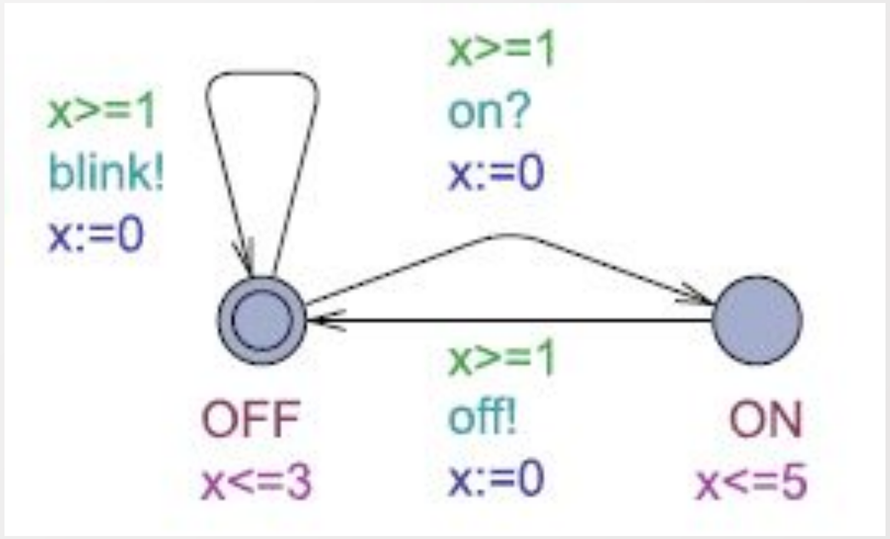
Used to represent timed properties



Example of Timed Automata

Timed Property:

- Whenever the light is switched ON, this setting has to be kept for 1 to 5 time units.
- Whenever the light is switched OFF, the light switch has to blink at least once every 3 time units.

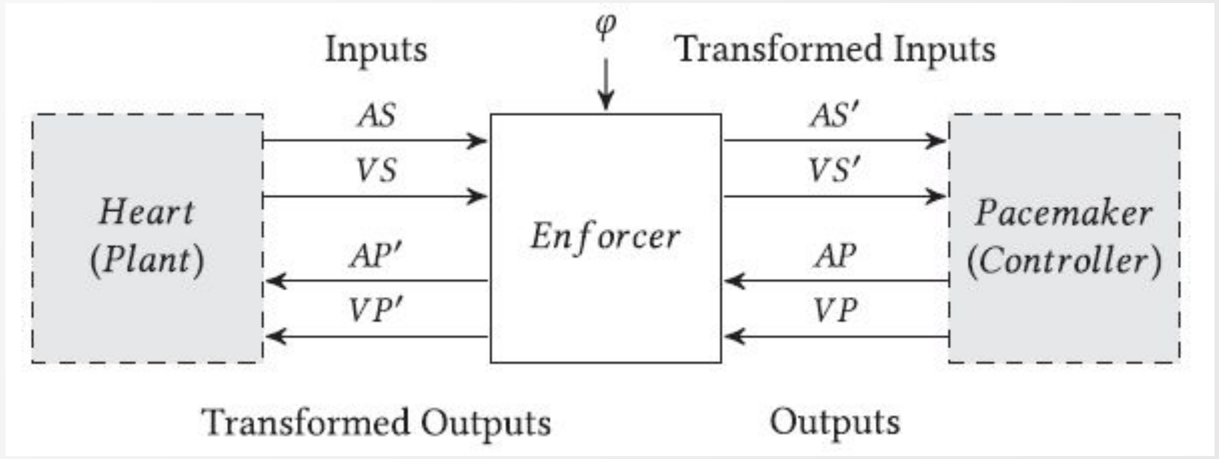


Timed Automata

<u>Existing Approach 1</u>

Runtime Enforcement of CPS

Bidirectional Runtime Enforcement



Bidirectional Runtime Enforcement

- Used in enforcing security policies.
- Enforcer suppresses malicious inputs from an attacker by modifying inputs.
- In Unmanned Aerial Systems, an attacker may feed-in bad inputs to take system control.

Example:

- Property: “when the brake and cruise inputs are simultaneously present, the brake should be given priority”.
- Enforcer will forward the brake input while toggling the cruise input.

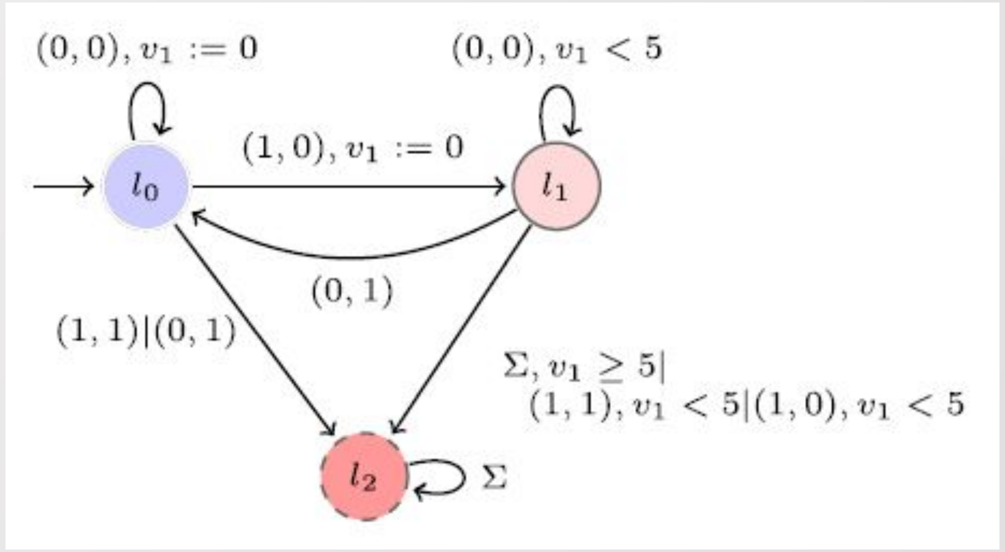
Discrete Timed Automata

- Timed automata with non-negative integer valued clocks
- Easier to analyse than timed automata
  - discrete notion of clock

<u>Timed Property</u>: A and B cannot happen simultaneously, A and B alternate starting with an A. B should be true with in 5 ticks after A occurs.

- Inputs (Σ_I) = {0,1}, Outputs (Σ_O) = {0,1}, $\Sigma = \Sigma_I \times \Sigma_O$
- Input trace/word σ = (0, 0) · (1, 0) ·(0, 0) · (0, 0) · (0, 1) ∈ Σ^*.

σ is accepted by DTA since the state reached upon run on σ is (l_0, v_1 = 3) which is an accepting state.

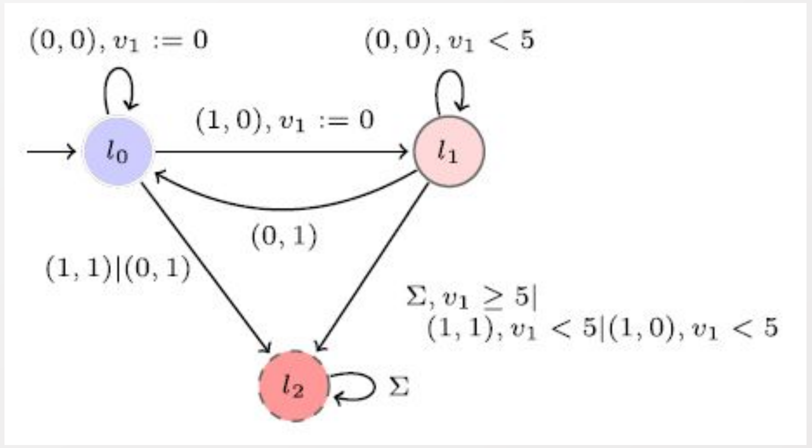


Language of a Property

- Language of a property is a set of all traces/words accepted by its DTA.
- Program (P) (can be a controller) satisfy a property (φ) iff L(P) ⊆ L(φ).

Input DTA

- Input DTA is obtained from Input-Output DTA.
- Ignore Outputs on the transitions.



$\longrightarrow$

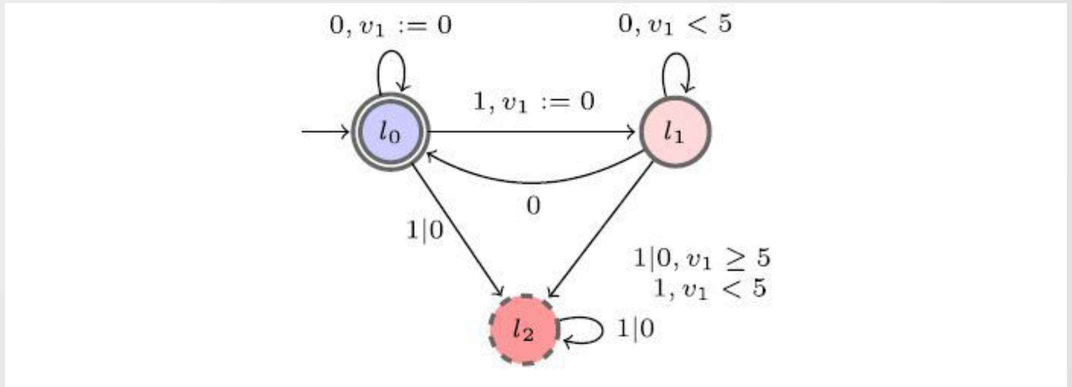


Fig. 5. Automaton obtained from $\mathcal{A}_{S_1}$ in Figure 3 by projecting on inputs.

Algorithm: Example Input - Output Behaviour

Table 1 shows input output behaviour of the algorithm.

The input output word read by the enforcer is
(0, 1) · (1, 1) · (1, 0) · (1, 0) · (0, 1)

The input output word released as output by the enforcer is (0, 0) · (1, 0) · (0, 0) · (0, 0) · (0, 1).

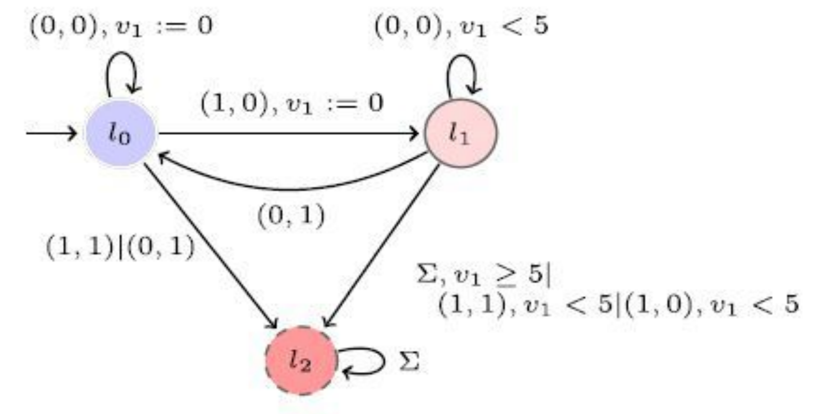


Fig. 3. Property S_1 defined as DTA $\mathcal{A}_{S_1}$.

Input-Output DTA

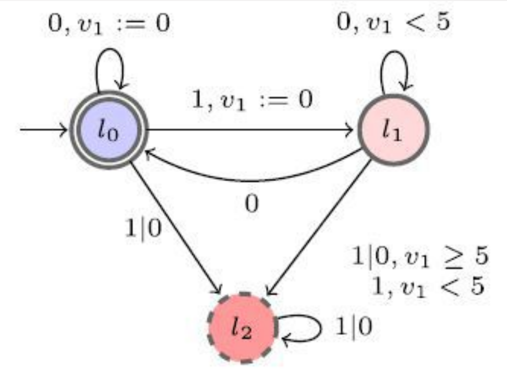
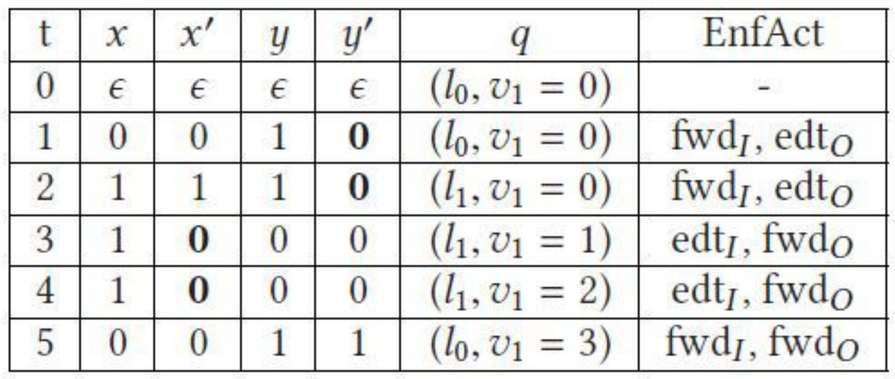


Fig. 5. Automaton obtained from $\mathcal{A}_{S_1}$ in Figure 3 by projecting on inputs.

Input DTA

Table 1. Example Illustrating Algorithm 1



t	x	x'	y	y'	q	EnfAct
0	ϵ	ϵ	ϵ	ϵ	$(l_0, v_1 = 0)$	-
1	0	0	1	**0**	$(l_0, v_1 = 0)$	fwd_I, edt_O
2	1	1	1	**0**	$(l_1, v_1 = 0)$	fwd_I, edt_O
3	1	**0**	0	0	$(l_1, v_1 = 1)$	edt_I, fwd_O
4	1	**0**	0	0	$(l_1, v_1 = 2)$	edt_I, fwd_O
5	0	0	1	1	$(l_0, v_1 = 3)$	fwd_I, fwd_O

Input-Output behaviour

Limitations of the Approach

- Using this approach, only enforceable properties can be enforced.
- Enforceable properties are just a subset of safety properties.

Property is enforceable if and only if an accepting state is reachable from every non-violating state in one or more steps.

Example: Let's take a non-enforceable property as defined using DTA in figure.

- Inputs (Σ_I) = {0,1}, Outputs (Σ_O) = {0,1}, $\Sigma = \Sigma_I \times \Sigma_O$
- Input-output sequence $\sigma = (1, 1) \cdot (1, 0) \in \Sigma^*$.

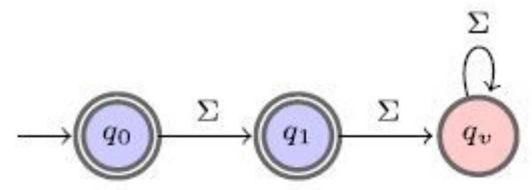
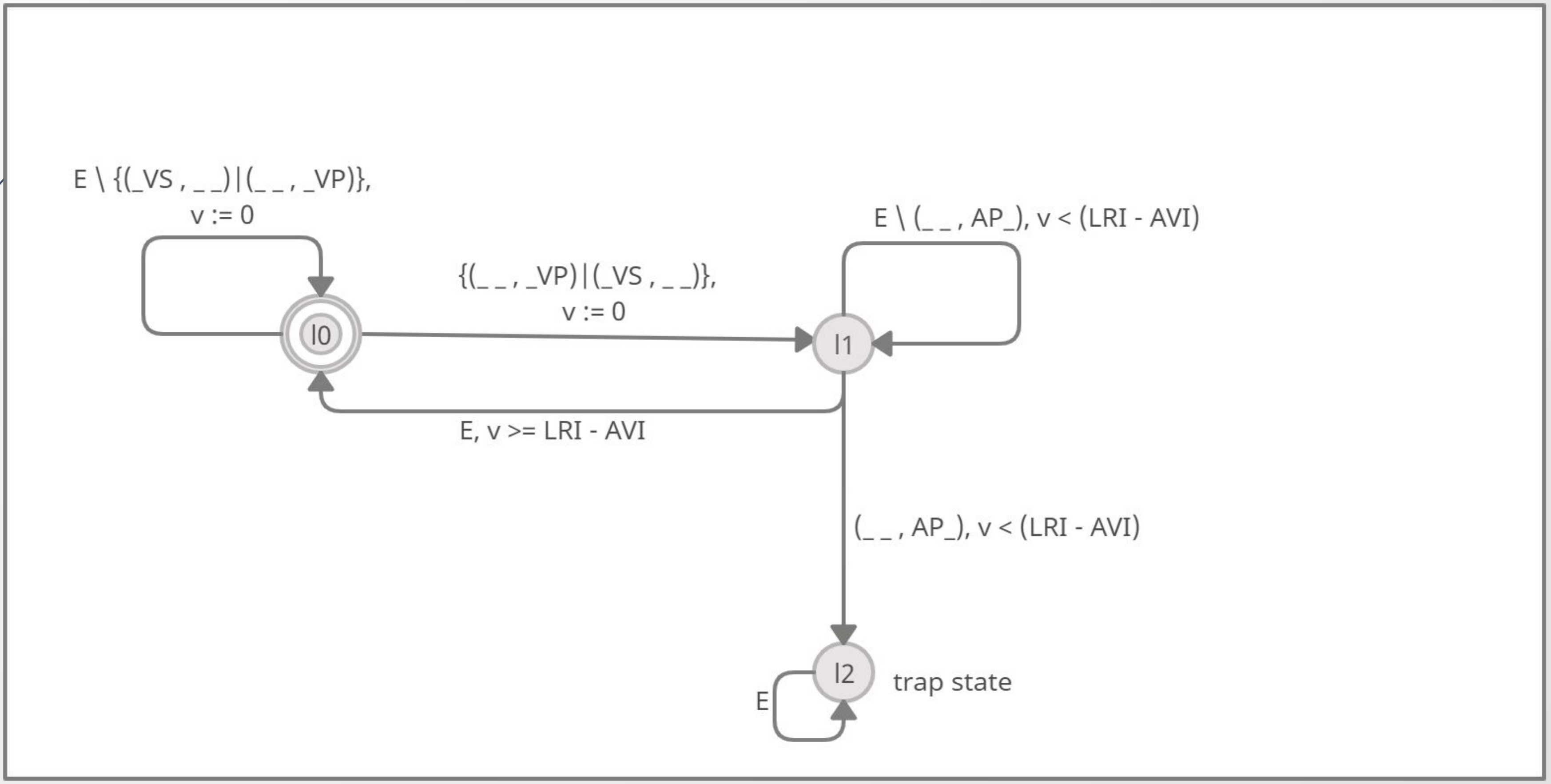


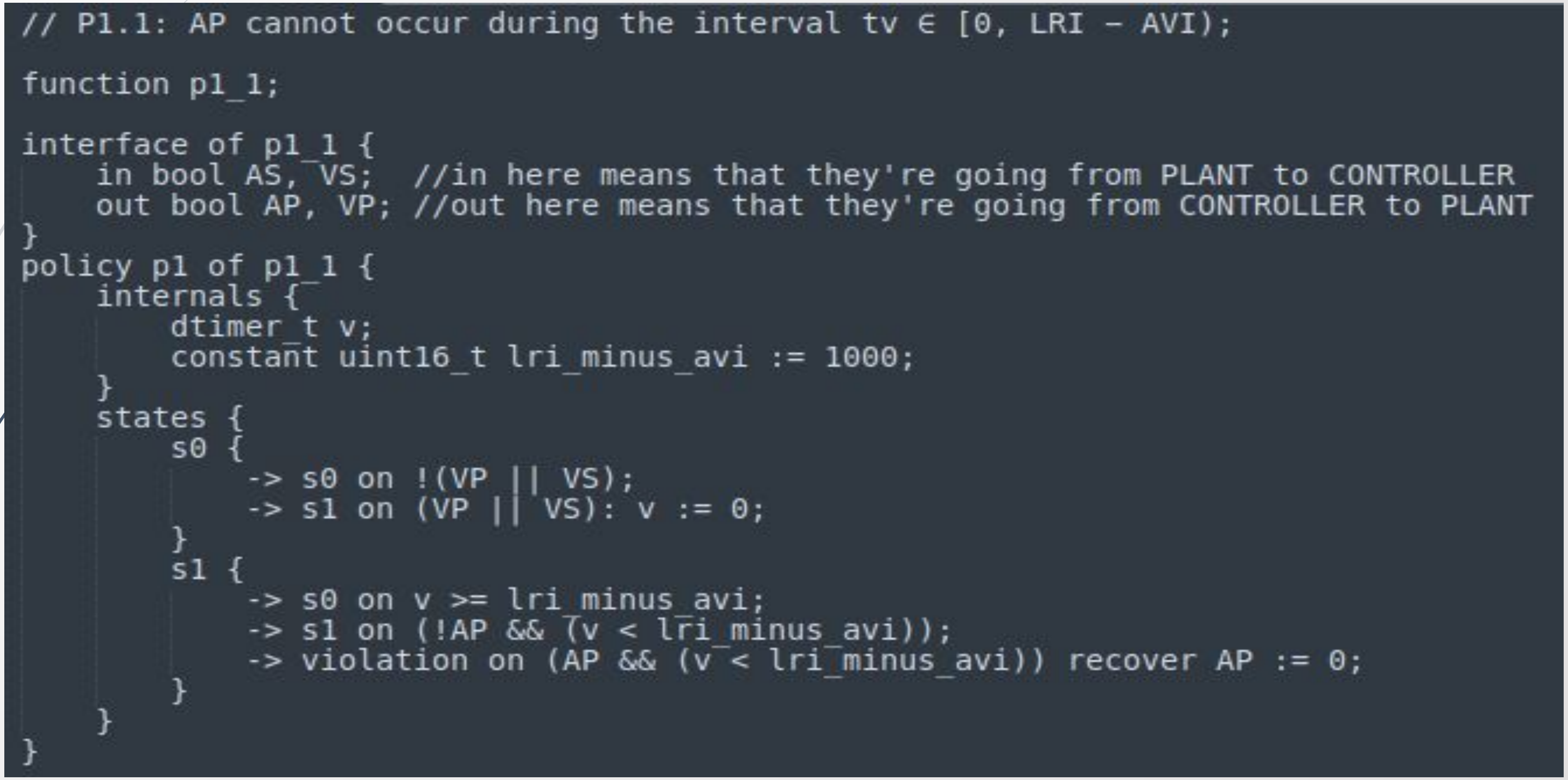
Fig. 6. A non-enforceable property.

Implemented Examples on Tool by us (P1.1)

- AP cannot occur during the interval $t_v \in [0, LRI - AVI)$ i.e.
- AP cannot occur within the interval [0, LRI - AVI) after any ventricular event (VS or VP) happens.

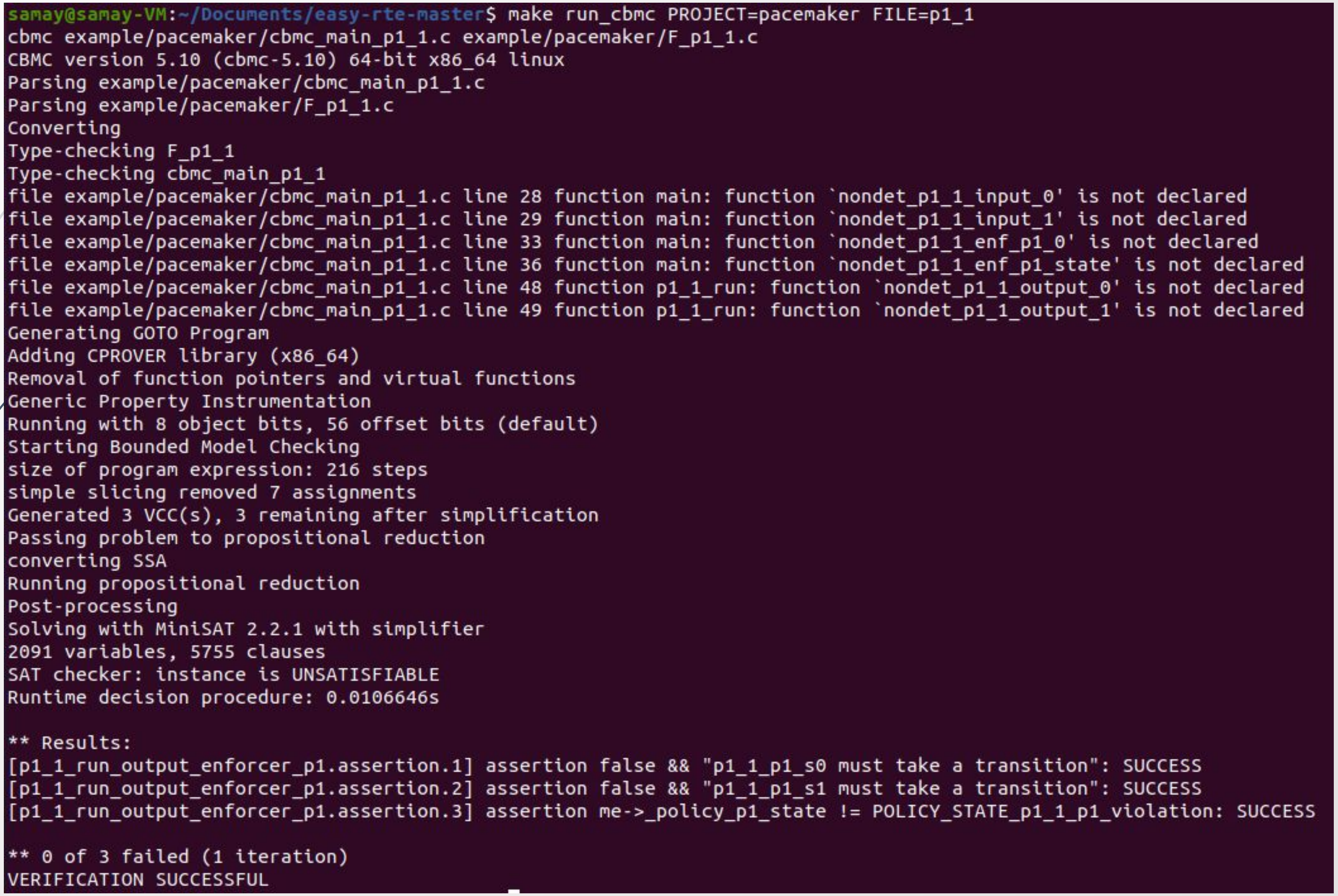

E \ {(_VS , _ _) | (_ _ , _VP)},
v := 0
{(_ _ , _VP) | (_VS , _ _)},
v := 0
E \ (_ _ , AP_), v < (LRI - AVI)
l0
l1
E, v >= LRI - AVI
(_ _ , AP_), v < (LRI - AVI)
l2
trap state
E

Implemented Examples on Tool by us (P1.1)



```
// P1.1: AP cannot occur during the interval tv ∈ [0, LRI − AVI);

function p1_1;

interface of p1_1 {
    in bool AS, VS;  //in here means that they're going from PLANT to CONTROLLER
    out bool AP, VP; //out here means that they're going from CONTROLLER to PLANT
}
policy p1 of p1_1 {
    internals {
        dtimer_t v;
        constant uint16_t lri_minus_avi := 1000;
    }
    states {
        s0 {
            -> s0 on !(VP || VS);
            -> s1 on (VP || VS): v := 0;
        }
        s1 {
            -> s0 on v >= lri_minus_avi;
            -> s1 on (!AP && (v < lri_minus_avi));
            -> violation on (AP && (v < lri_minus_avi)) recover AP := 0;
        }
    }
}
```

Implemented Examples on Tool by us (P1.1)

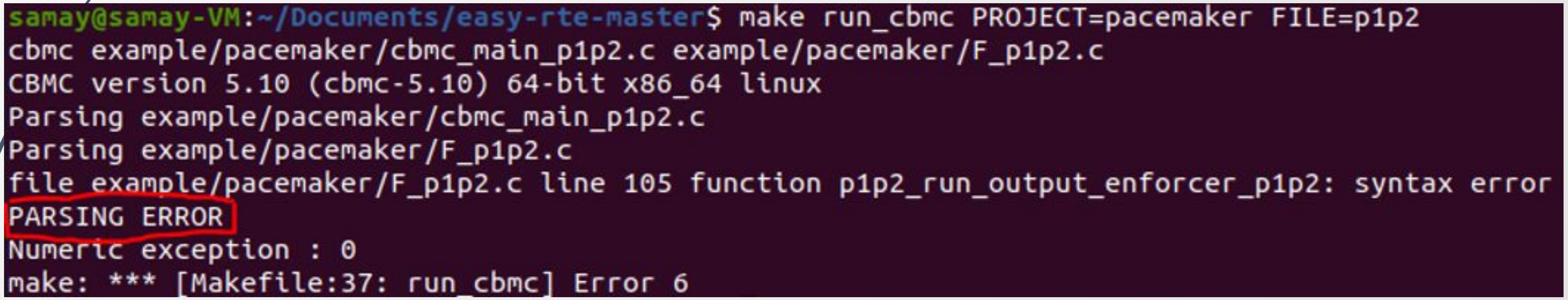


```
samay@samay-VM:~/Documents/easy-rte-master$ make run_cbmc PROJECT=pacemaker FILE=p1_1
cbmc example/pacemaker/cbmc_main_p1_1.c example/pacemaker/F_p1_1.c
CBMC version 5.10 (cbmc-5.10) 64-bit x86_64 linux
Parsing example/pacemaker/cbmc_main_p1_1.c
Parsing example/pacemaker/F_p1_1.c
Converting
Type-checking F_p1_1
Type-checking cbmc_main_p1_1
file example/pacemaker/cbmc_main_p1_1.c line 28 function main: function `nondet_p1_1_input_0' is not declared
file example/pacemaker/cbmc_main_p1_1.c line 29 function main: function `nondet_p1_1_input_1' is not declared
file example/pacemaker/cbmc_main_p1_1.c line 33 function main: function `nondet_p1_1_enf_p1_0' is not declared
file example/pacemaker/cbmc_main_p1_1.c line 36 function main: function `nondet_p1_1_enf_p1_state' is not declared
file example/pacemaker/cbmc_main_p1_1.c line 48 function p1_1_run: function `nondet_p1_1_output_0' is not declared
file example/pacemaker/cbmc_main_p1_1.c line 49 function p1_1_run: function `nondet_p1_1_output_1' is not declared
Generating GOTO Program
Adding CPROVER library (x86_64)
Removal of function pointers and virtual functions
Generic Property Instrumentation
Running with 8 object bits, 56 offset bits (default)
Starting Bounded Model Checking
size of program expression: 216 steps
simple slicing removed 7 assignments
Generated 3 VCC(s), 3 remaining after simplification
Passing problem to propositional reduction
converting SSA
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.1 with simplifier
2091 variables, 5755 clauses
SAT checker: instance is UNSATISFIABLE
Runtime decision procedure: 0.0106646s

** Results:
[p1_1_run_output_enforcer_p1.assertion.1] assertion false && "p1_1_p1_s0 must take a transition": SUCCESS
[p1_1_run_output_enforcer_p1.assertion.2] assertion false && "p1_1_p1_s1 must take a transition": SUCCESS
[p1_1_run_output_enforcer_p1.assertion.3] assertion me->_policy_p1_state != POLICY_STATE_p1_1_p1_violation: SUCCESS

** 0 of 3 failed (1 iteration)
VERIFICATION SUCCESSFUL
```

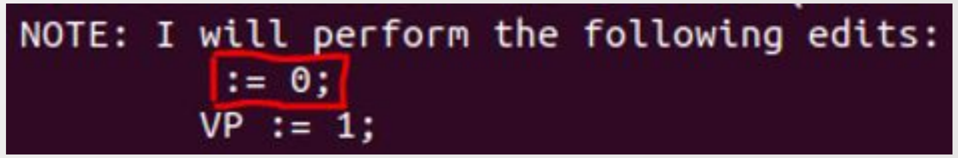
Problems Faced in usage of Tool

Explored the runtime enforcement tool (https://github.com/PRETgroup/easy-rte).

- In pacemaker example, on removing manual recover statements, in some cases it was giving wrong outputs.
- In some cases it says "no solution is found" although solution exists.
- In other, it was editing adding unnecessary statements like ":=0" which does not make sense.



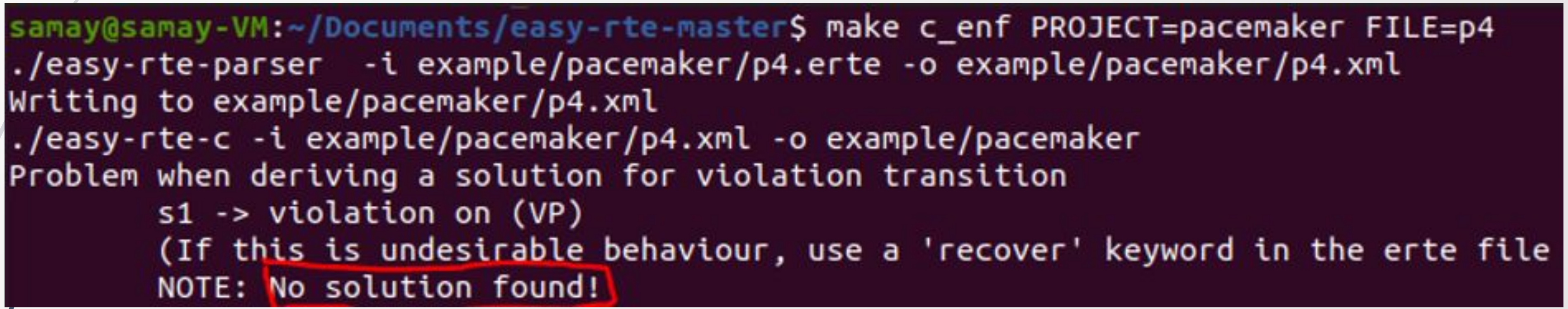
```
samay@samay-VM:~/Documents/easy-rte-master$ make run_cbmc PROJECT=pacemaker FILE=p1p2
cbmc example/pacemaker/cbmc_main_p1p2.c example/pacemaker/F_p1p2.c
CBMC version 5.10 (cbmc-5.10) 64-bit x86_64 linux
Parsing example/pacemaker/cbmc_main_p1p2.c
Parsing example/pacemaker/F_p1p2.c
file example/pacemaker/F_p1p2.c line 105 function p1p2_run_output_enforcer_p1p2: syntax error
PARSING ERROR
Numeric exception : 0
make: *** [Makefile:37: run_cbmc] Error 6
```



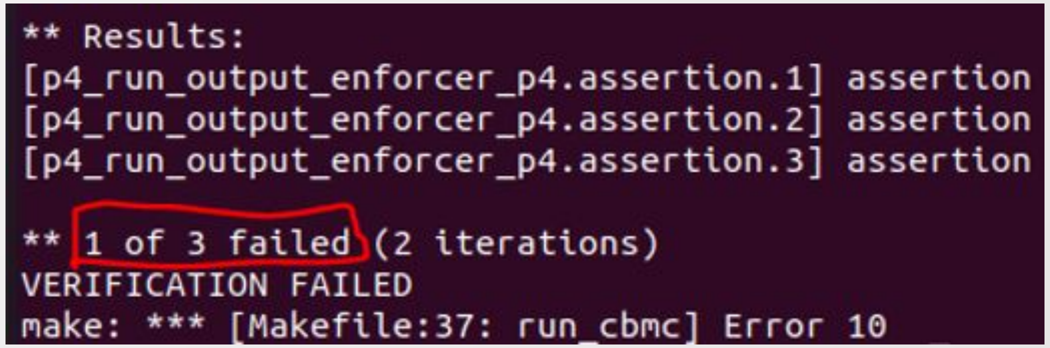
```
NOTE: I will perform the following edits:
         := 0;
        VP  := 1;
```

Problems Faced in usage of Tool (contd.)

- In our policies, we need to add manual recover statements
- otherwise it is saying "no solution is found"
- Making the whole tool useless if we have to correct violation manually



```
samay@samay-VM:~/Documents/easy-rte-master$ make c_enf PROJECT=pacemaker FILE=p4
./easy-rte-parser  -i example/pacemaker/p4.erte -o example/pacemaker/p4.xml
Writing to example/pacemaker/p4.xml
./easy-rte-c -i example/pacemaker/p4.xml -o example/pacemaker
Problem when deriving a solution for violation transition
        s1 -> violation on (VP)
        (If this is undesirable behaviour, use a 'recover' keyword in the erte file
        NOTE: No solution found!
```



```
** Results:
[p4_run_output_enforcer_p4.assertion.1] assertion
[p4_run_output_enforcer_p4.assertion.2] assertion
[p4_run_output_enforcer_p4.assertion.3] assertion

** 1 of 3 failed (2 iterations)
VERIFICATION FAILED
make: *** [Makefile:37: run_cbmc] Error 10
```

Existing Approach 2

Shield Synthesis for Timed Properties

Timed Shield

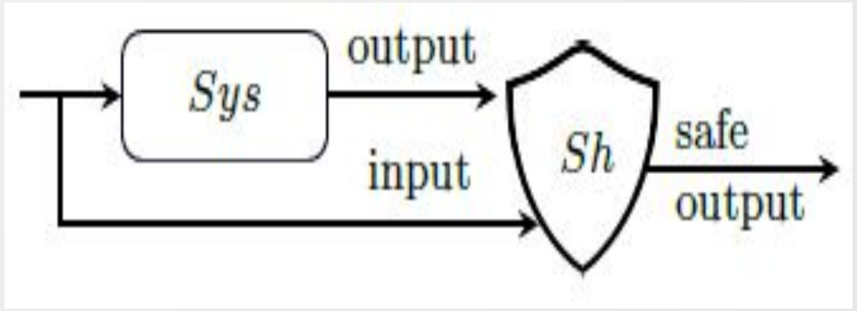
- A timed shield is attached after the controller,
- Monitors its inputs and outputs,
- Corrects and forwards the correct output to the environment/plant.

Desired Properties of the shield:

- Correctness: Shield ensures correctness if and only if a shielded system refines specification.

(System | Shield) ≤ Specification.

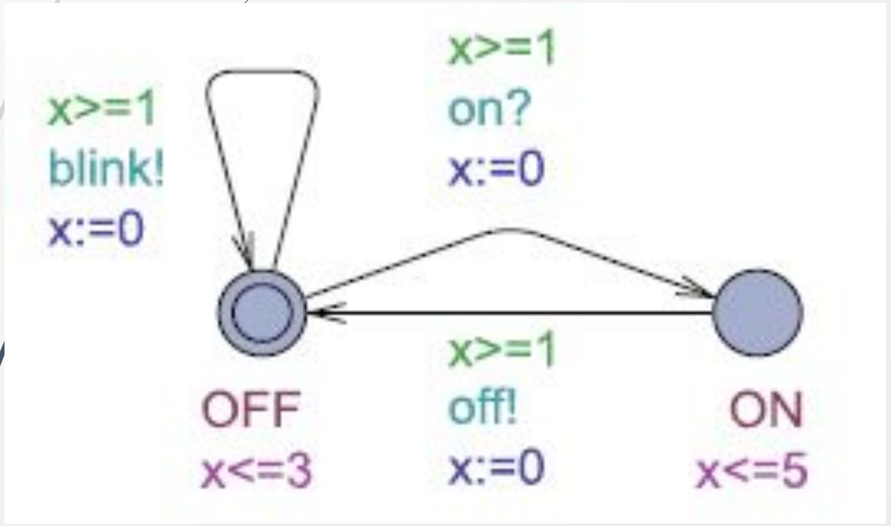
- Refinement (≤): containment of timed behaviour. i.e.
- Behaviour of shielded system is a subset of behaviour of specification.



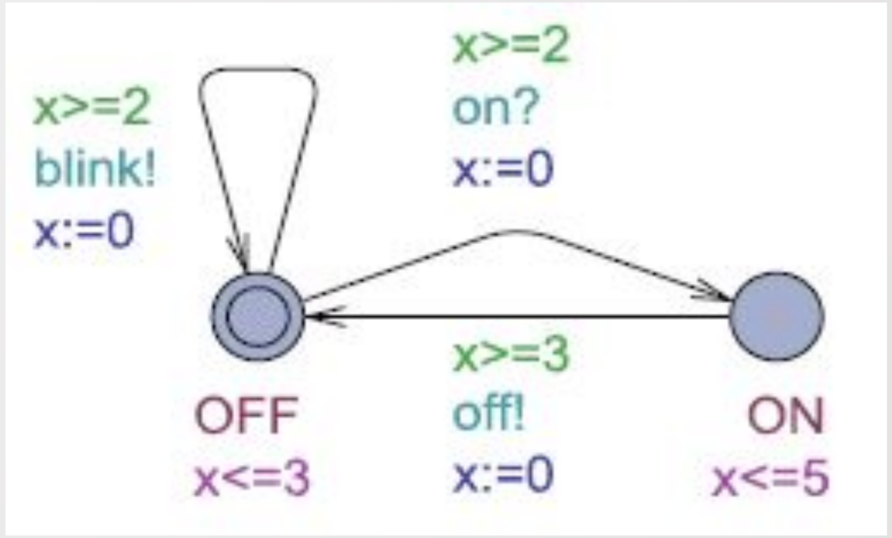
Refinement of Timed Automata (≤)

- Refinement is denoted by ≤.

<u>Specification:</u> Whenever the light is switched ON, this setting has to be kept for 1 to 5 time units. Whenever the light is switched OFF, the light switch has to blink at least once every 3 time units.



$\longrightarrow$



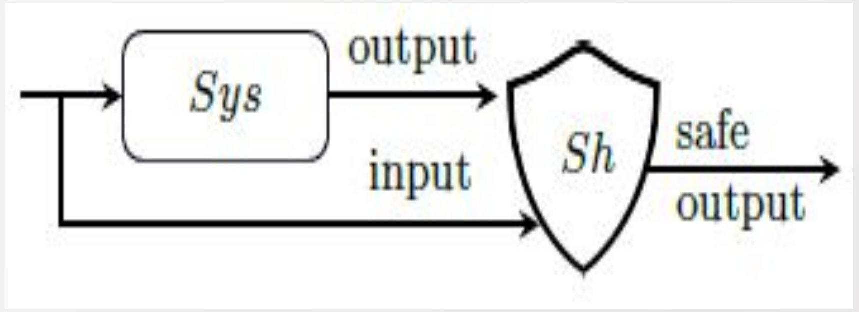
A B

- Timed Automata B refines Timed Automata A (B ≤ A) means L(B) ⊆ L(A)

Timed Shield

Desired Properties of the shield:

- <u>No Unnecessary Deviation:</u> Shield does not deviate from System unnecessarily. Why?



Timed Shield Synthesis

- Timed shield can be synthesized by solving a timed game.
- Played by 2 players: System and Environment on Timed game automata (TGA).

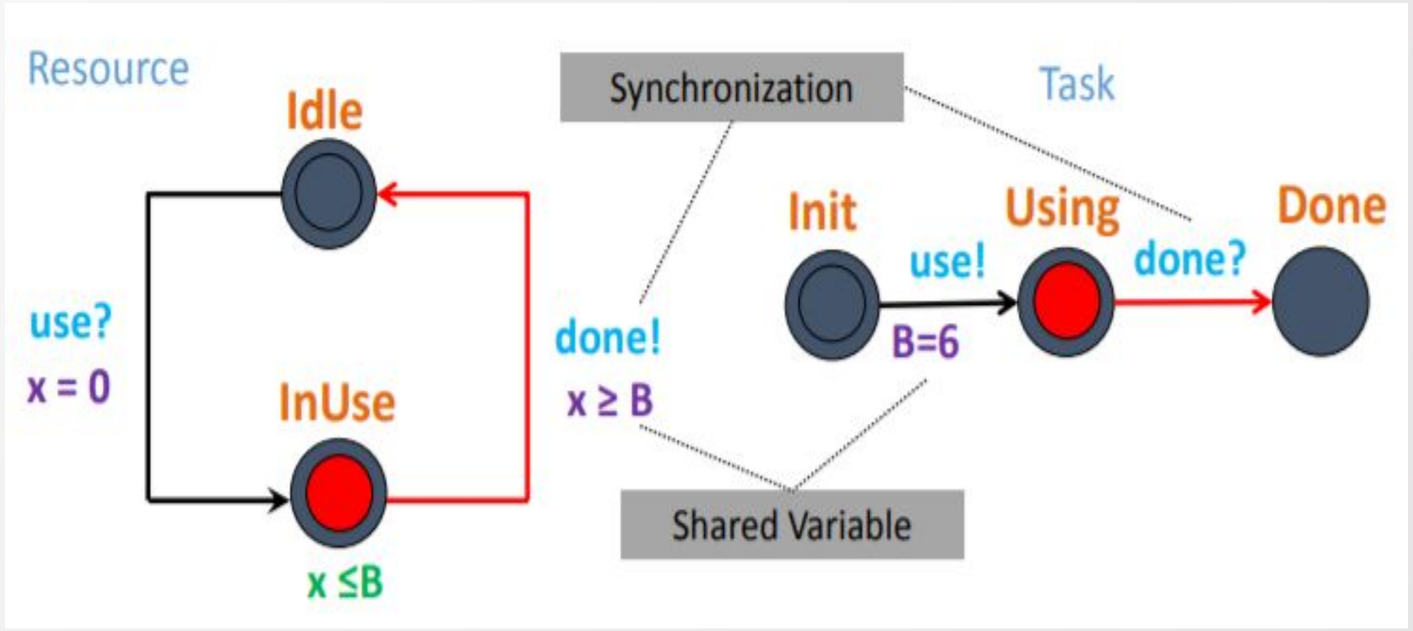
- Timed Game Automata is timed automata in which set of output actions partitioned into controllable actions and uncontrollable actions.
- System can choose controllable actions.
- Environment chooses uncontrollable actions.

- System need to satisfy the safety objective. How?
  - **Strategy:** A function over the states of TGA to the set of controllable actions or a special nothing symbol.
  - **Winning**: Safety objective is never violated no matter what environment does.
  - **Winning strategy**: A strategy which is always winning no matter what strategy environment chooses.

A **Timed Shield** is the network of Timed Automata obtained by composing Timed Game Automata (TGA) with the winning strategy.

Network of Timed Automata (TA composition)

- Network of Timed Automata is denoted by '|'



! is a output action and ? is a input action. They are used for synchronization.
Communication between automata through channels and shared variables.

Example: Timed Shield

<u>Example:</u> Say we have a Timed Game (represented using TGA) and the following safety objective:

control: A□ distance between cars > 5

- Solving the timed game automata w.r.t to this safety objective produces a winning strategy
- Timed shield: Winning strategy composed with the timed game automata

Future Work

- Explore UPPAAL, understand its working (Uppaal is an integrated tool for modeling, validation and verification of real-time systems modeled as networks of timed automata).
- Implement some real life examples like platooning of cars, pacemakers etc.
- Explore different UPPAAL branches and understand which is best for shield synthesis.
- Use the selected UPPAAL branch to <u>implement algorithms for solving games based on timed game automata with safety properties</u> for different CPS.
- To identify the shortcomings/problems with the UPPAAL tool.

Questions?