CS 498: Bachelor's Thesis Project
End Semester Evaluation

Shield Synthesis for Cyber-Physical Systems

Supervisor: Dr. Purandar Bhaduri

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

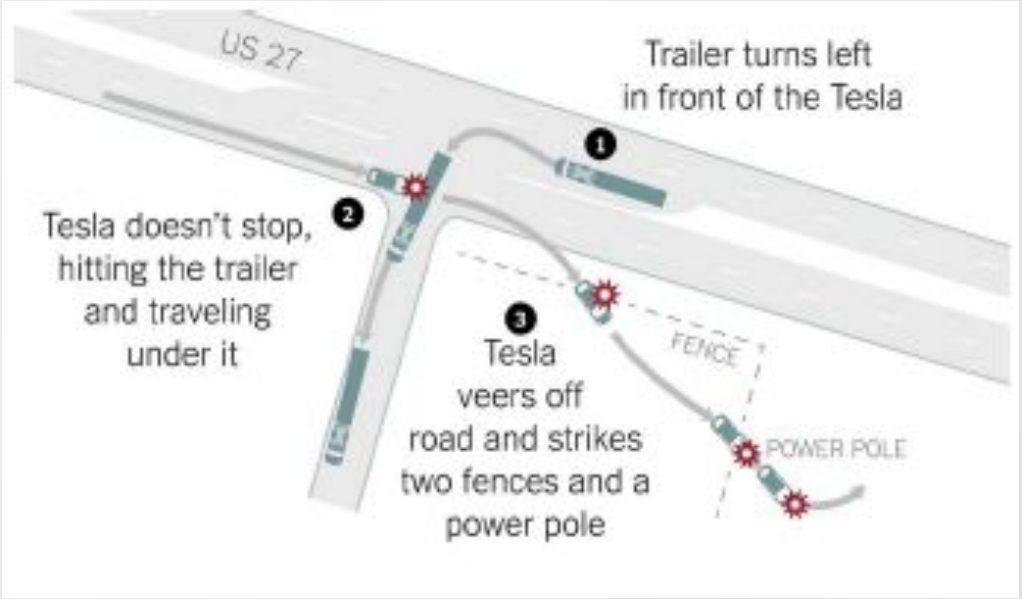
Need for Shield Synthesis (Runtime Enforcement)

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)



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.

Existing Approach 1

Runtime Enforcement of CPS

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 $\sigma = (1, 1) \cdot (1, 0) \in \Sigma^*$.

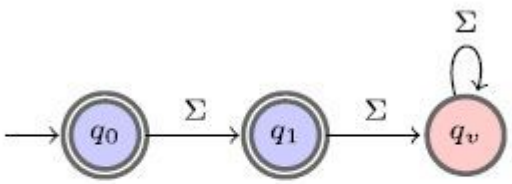


Fig. 6. A non-enforceable property.

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, 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!
```

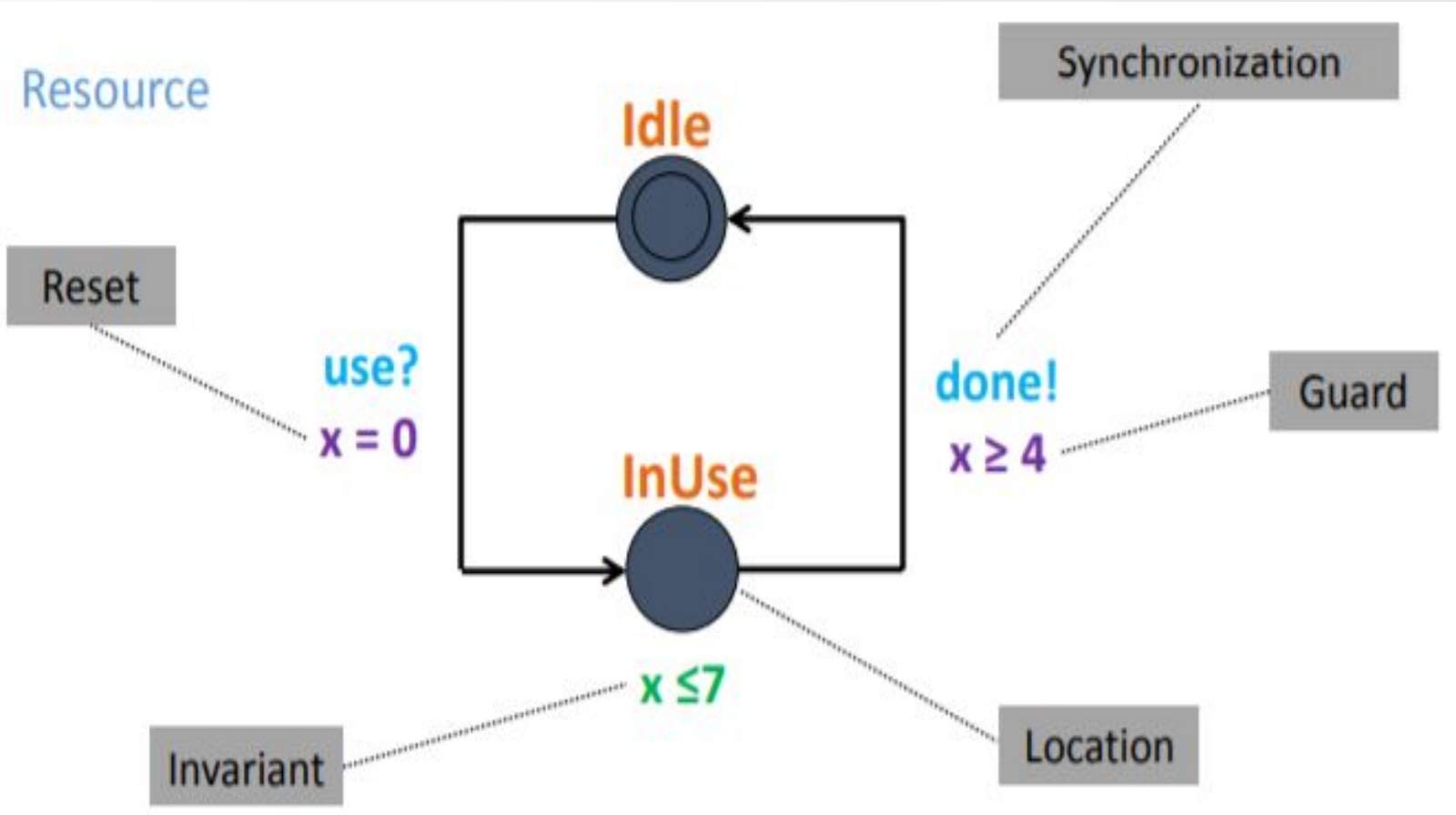
```
** Results:
[p4_run_output_enforcer_p4.assertion.1] assertion
[p4_run_output_enforcer_p4.assertion.2] assertion
[p4_run_output_enforcer_p4.assertion.3] assertion

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

Shield Synthesis for Timed Properties

Timed Automata

- Used to represent timed properties



Timed Shield

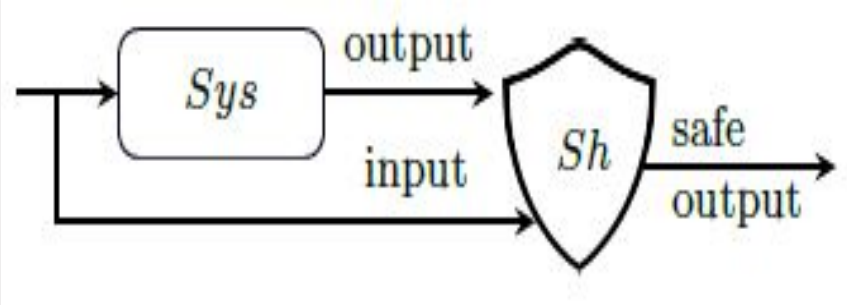
- Attached after the system,
- Monitors its inputs and outputs,
- Corrects and forwards the correct output to the environment/plant.

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.
- Timed behaviour of shielded system is a subset of timed behaviour of specification.

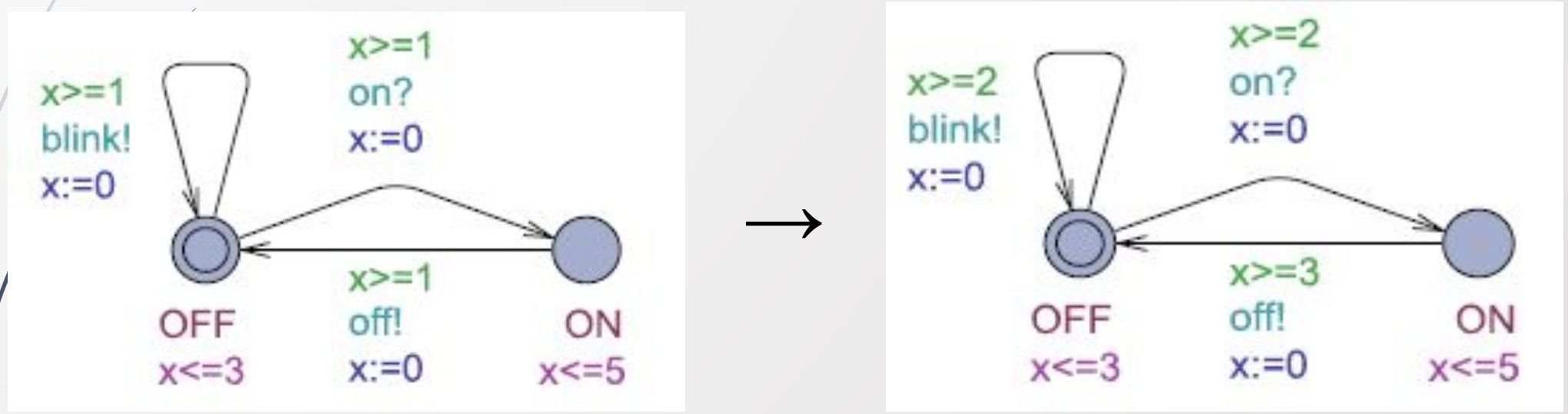


Sys: System
Sh: Shield

Refinement of Timed Automata (≤)

- Denoted by ≤.

Specification: 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.



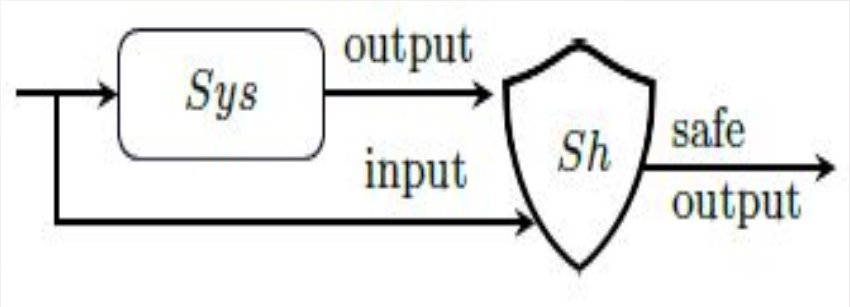
A B

- B refines A (B ≤ A) if set of timed traces of B is contained in set of timed traces of A.

Timed Shield

Desired Properties of the shield:

- No Unnecessary Deviation: Shield does not deviate from System unnecessarily. Why?



Timed Shield Synthesis

- Solving a timed game.
- 2 Players: Player 0 (System) and Player 1 (Environment).

- Timed Game Automata: Timed Automata
  - Output Actions - Controllable Actions (System) and UnControllable Actions (Environment).

- System need to satisfy the safety objective. How?

  - **Strategy:** A function over the states of TGA to the set of controllable actions.

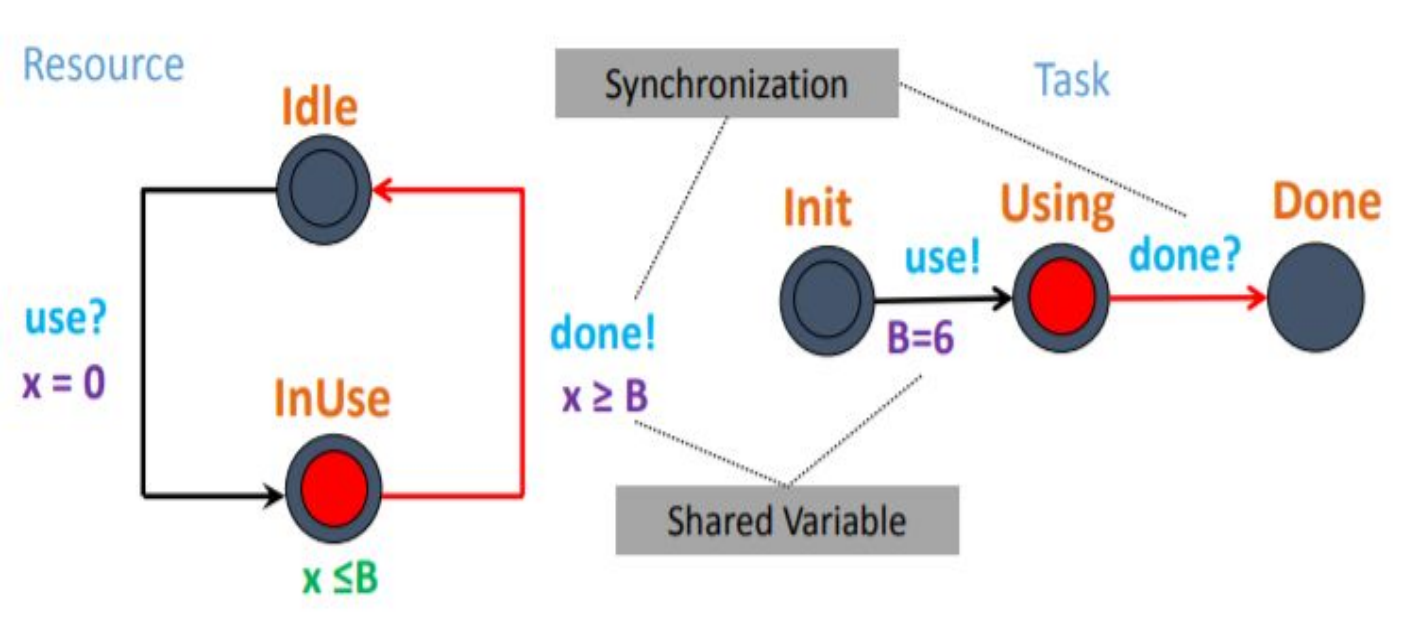
  - **Winning**: Safety objective is never violated no matter what environment does.

  - **Winning strategy**: Always winning no matter what strategy environment chooses.

Timed Shield: 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 '|'



! : Output Action
?: Input Action

! and ? are used for synchronization.
Communication occurs through channels and shared variables.

Algorithm: Timed Shield

Algorithmic Flow:

- **System / Environment** (Sys / Env) → black-box
- **Specification Monitor** (mSpec) → observes system's output
- **Shield** (Sh) → On composing with winning strategy corrects the system's output
- **Specification Monitor for the shield** (mSpec') → observes shield's output

Objective:

- **Shield is always correct:** Specification monitor for the shield never enters the error state.

Example of Timed Shield

Car Platooning System

Example: Car Platooning System

Two cars:

- Front (Environment)
- Ego (System)

Safety Property: Distance between the cars should be between 5 and 15 meters.

Objective:

- Implement timed shield
- Winning strategy using the UPPAAL STRATEGO tool
- Verification of safety property

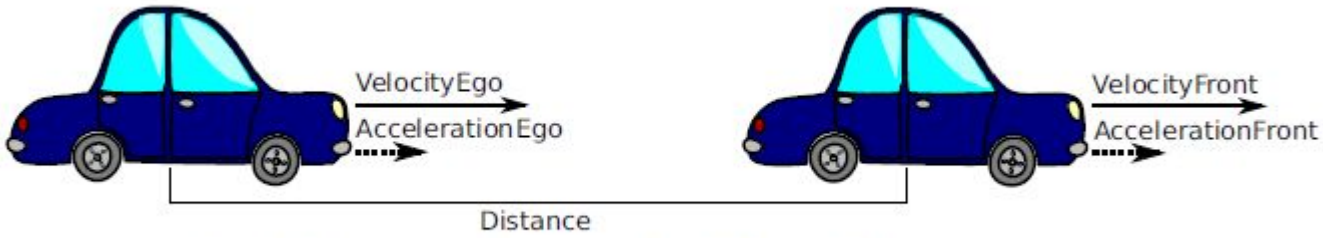
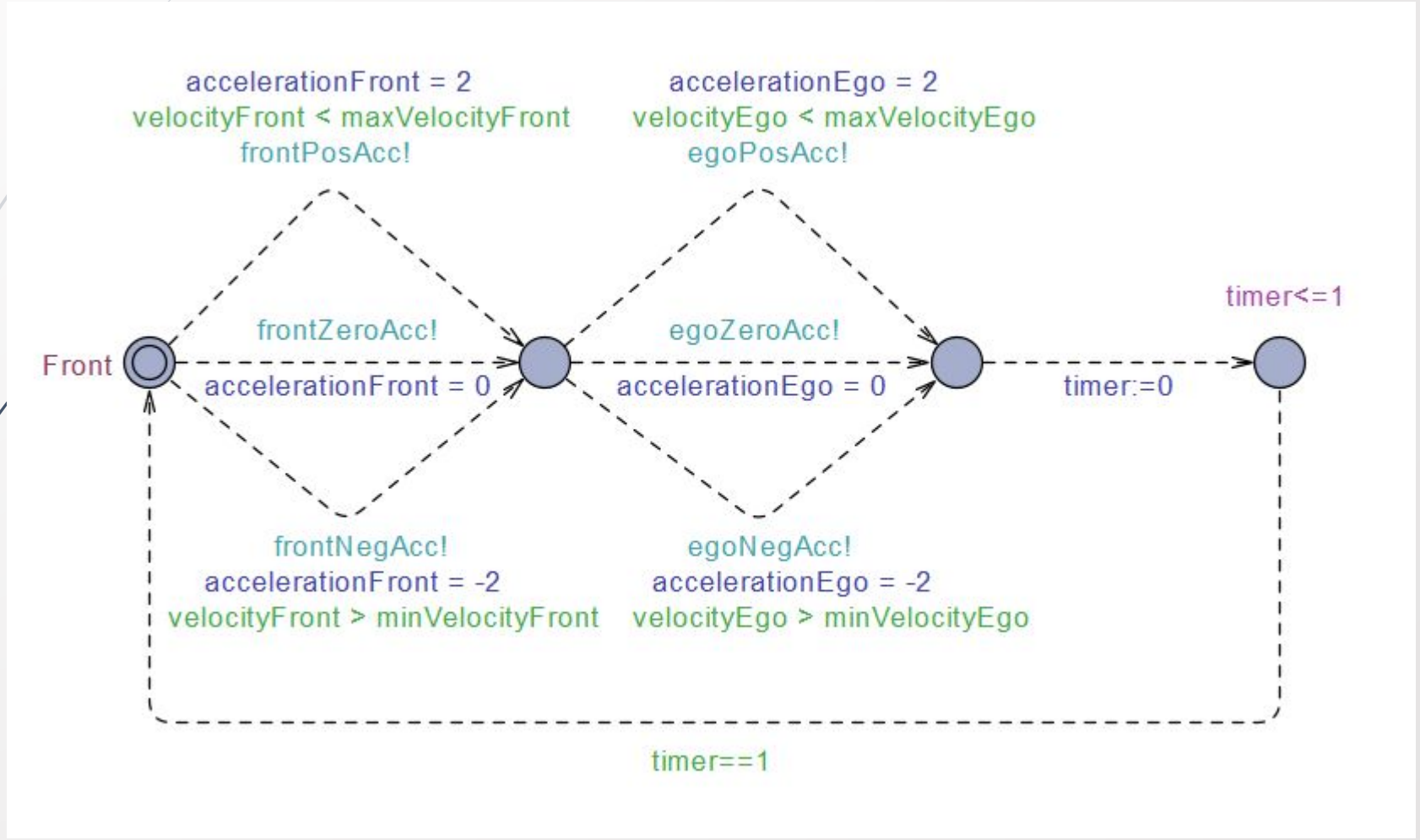
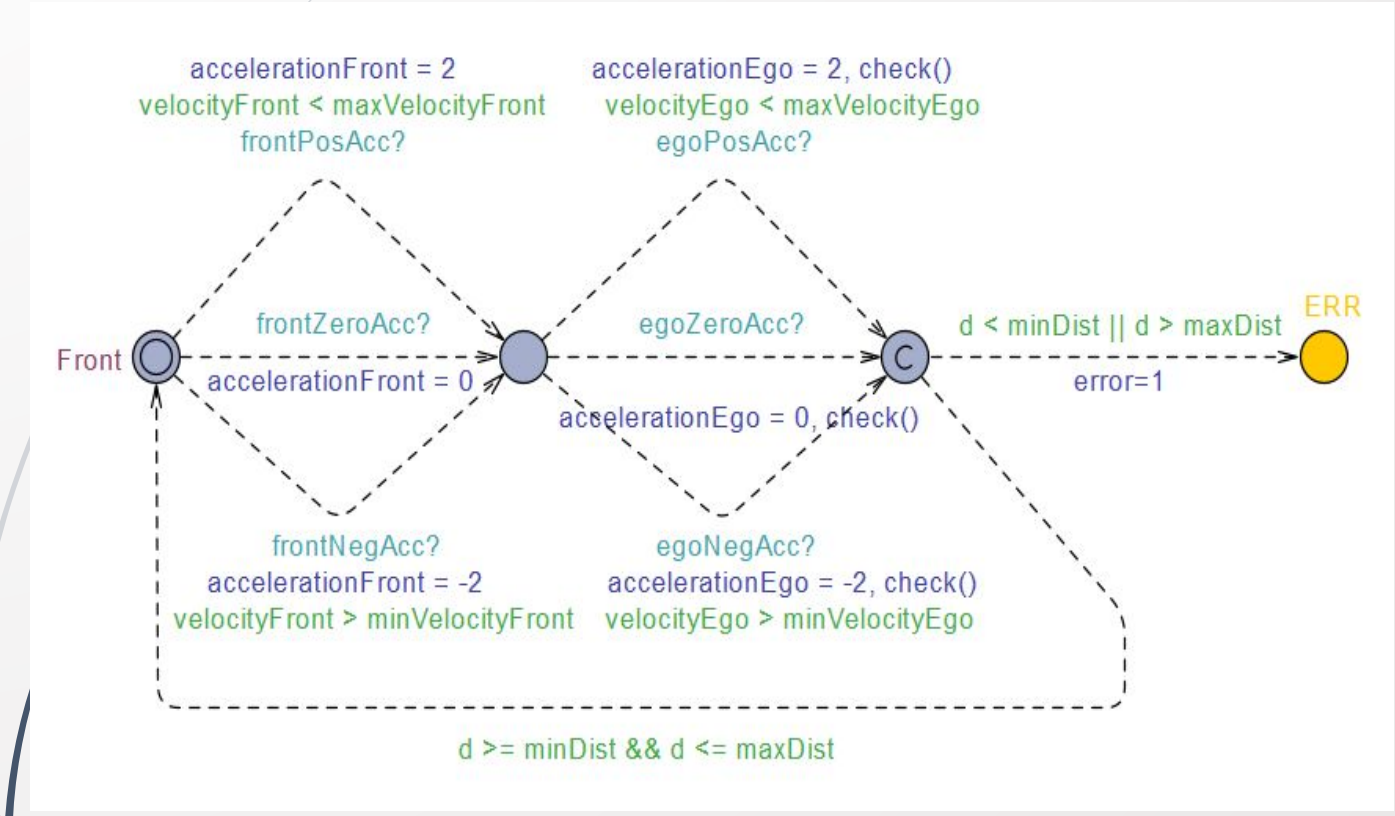


Fig. 1. Distance, velocity and acceleration between two cars.

Example: Car Platooning System (System)



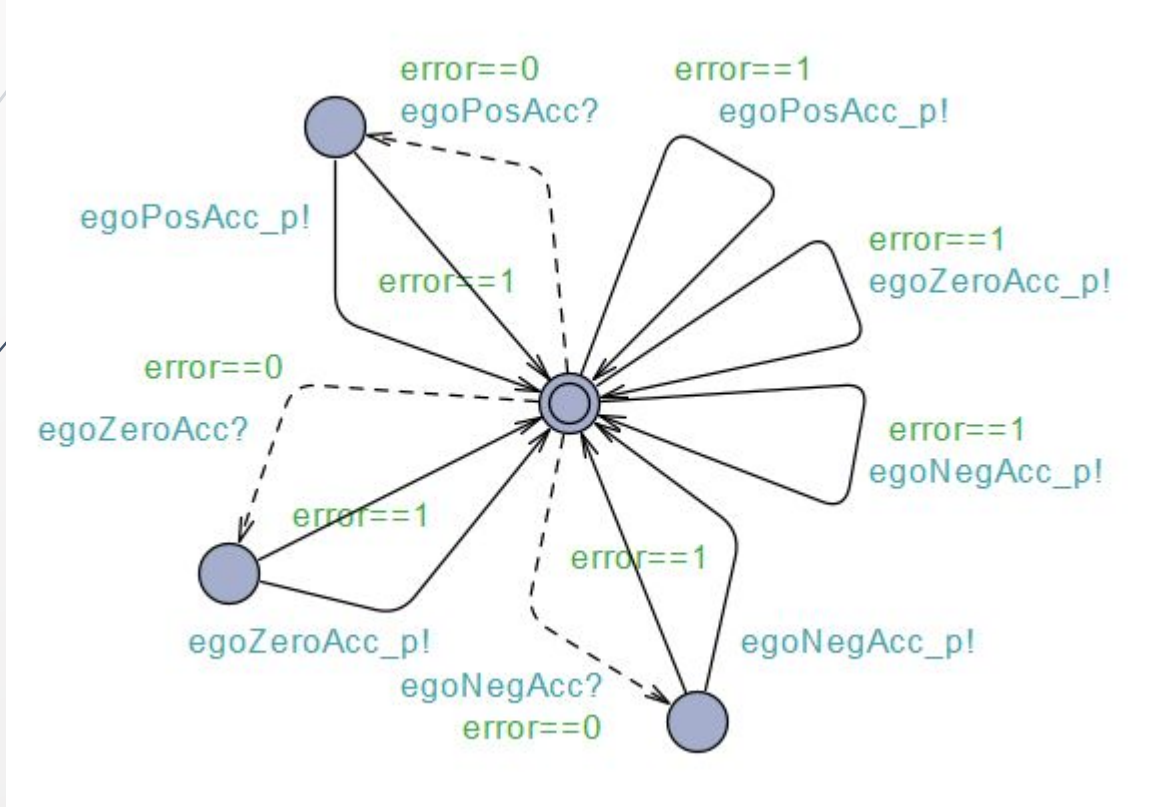
Example: Car Platooning System (Specification Monitor)



check():
function computes distance between the cars after 1 second.

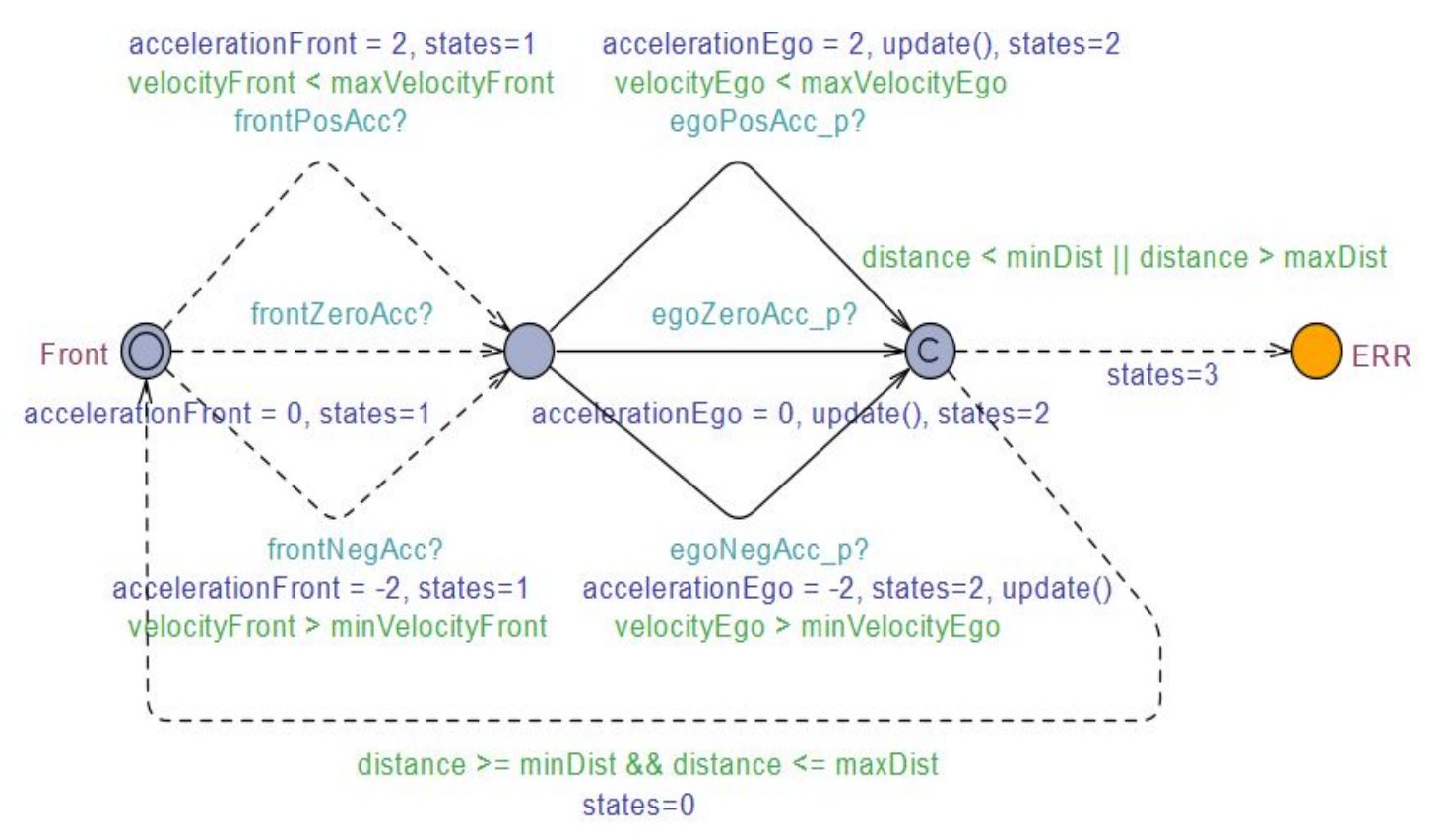
- Specification monitor observes the outputs of the system and check if it goes into the error state.

Example: Car Platooning System (Shield)



- Acceleration of ego (positive, zero, negative) is controlled by the shield.
- Shield has 3 parts:
  - Pre-fault actions
  - Post-fault actions
  - Last-chance actions
- error → tells if system goes into the error state or not.

Example: Car Platooning System (Specification Monitor for the Shield)



update():
function updates the velocity and distance between the cars w.r.t. to the shield output.

- It observes the output of the shield and check if it produces the correct output or not (goes into the error state).

Example: Car Platooning System (Verification)

```
strategy safe1 = control: A[] distance >= 9 && distance <= 12
saveStrategy("car_distance.txt", safe1)
strategy safe = control: A[] not mSpec_p.ERR
saveStrategy("car.txt", safe)
A[] distance >= 5 && distance <= 15 under safe
```

strategy safe1 = control: A[] distance >= 9 && distance <= 12
Verification/kernel/elapsed time used: 1.953s / 0.015s / 2.003s.
Resident/virtual memory usage peaks: 64,892KB / 142,832KB.
Property is satisfied.

A[] distance >= 5 && distance <= 15 under safe1
Verification/kernel/elapsed time used: 0.593s / 0s / 0.589s.
Resident/virtual memory usage peaks: 282,692KB / 584,528KB.
Property is satisfied.

strategy safe = control: A[] not mSpec_p.ERR
Verification/kernel/elapsed time used: 6.656s / 0.188s / 6.908s.
Resident/virtual memory usage peaks: 236,092KB / 514,788KB.
Property is satisfied.
saveStrategy("car.txt", safe)
Verification/kernel/elapsed time used: 4.11s / 2.39s / 6.528s.
Resident/virtual memory usage peaks: 236,092KB / 514,788KB.
Property is satisfied.

Example: Car Platooning System (Results)

Objectives:

- Implement the timed shield. ✅
- Find winning strategy: ✅
  - First query creates a winning strategy for the safety property.
  - **safe1** contains that winning strategy.
  - Strategy can also be saved in a .txt file.
- Verification of safety property: ✅
  - Last query checks if the shielded system satisfy the safety property or not.
  - Keyword **under** is used to compose the Timed Game Automata with the winning strategy **safe1**.

Algorithm: Time Shield with Recovery Guarantees

Algorithmic Flow:

- **System / Environment** (Sys / Env) → black-box
- **Specification Monitor** (mSpec) → observes system's output
- **Fault Models** → absorb transient fault
- **Recovery Model** → observe fault status
- **Shield** (Sh) → On composing with winning strategy corrects the system's output
- **Specification Monitor for the shield** (mSpec') → observes shield's output

Objective:

- **Shield is always correct:** Specification monitor for the shield never enters the error state.
- Recovery happens within the time.

Example of Timed Shield with Recovery Guarantees

Heart-Pacemaker System

Example: Heart-Pacemaker System

Consider the heart-pacemaker system as a **black-box**.

- Shield is independent of the system.

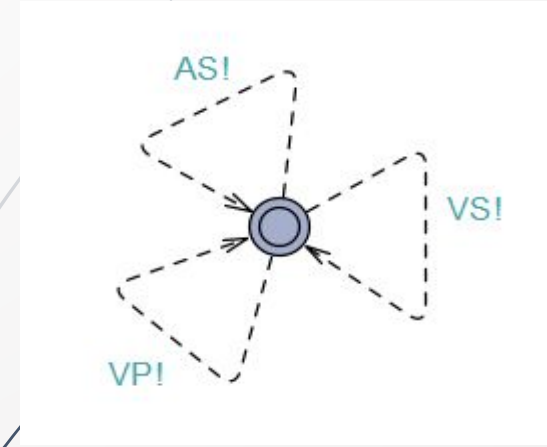
Combination of 2 safety properties:

- If Atrial Sense (AS) does not occur within interval $t_v \in [0, LRI - AVI)$, then Atrial Pace (AP) should occur at $t_v = LRI - AVI$.
- Atrial Pace (AP) cannot occur during the interval $t_v \in [0, LRI - AVI)$.

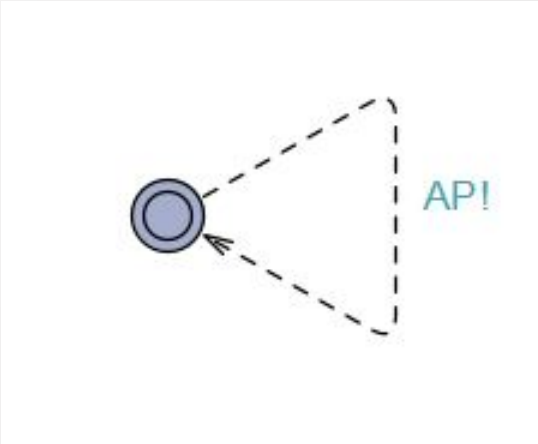
Objectives:

- Implement timed shield as a network of automata.
- Implement fault models to handle transient errors.
  - **Transient Error**: Happens only once, and has correct pre and post error behaviour.
- Find winning strategy using the UPPAAL STRATEGO tool.
- Verification of safety properties.

Example: Heart-Pacemaker System



Environment



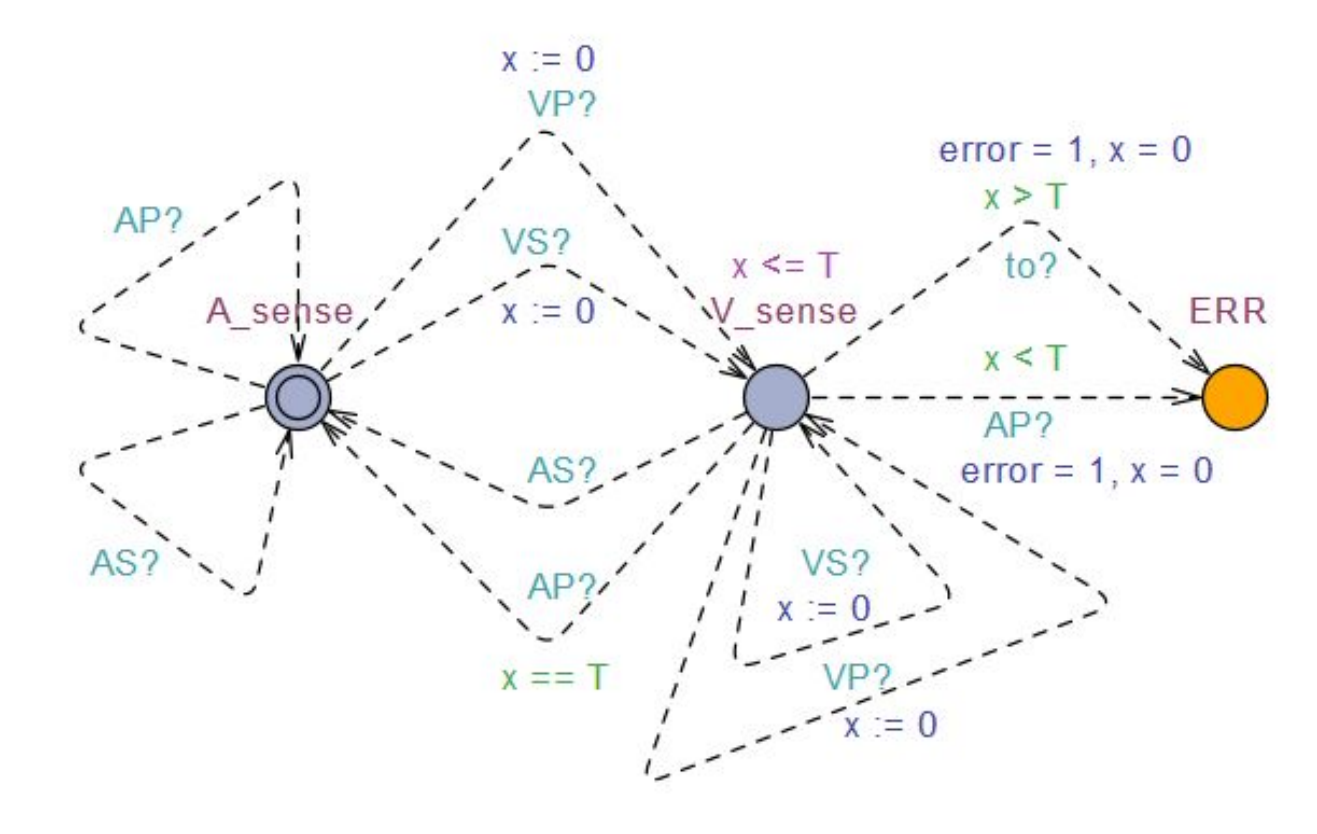
System

System can give 2 possible outputs:

- AP
- Wait

- AS, VS, and VP are controlled by the environment.
- AP is controlled by the system.
- As shield is independent, outputs are taken at random at each time step.

Example: Heart-Pacemaker System (Specification Monitor)

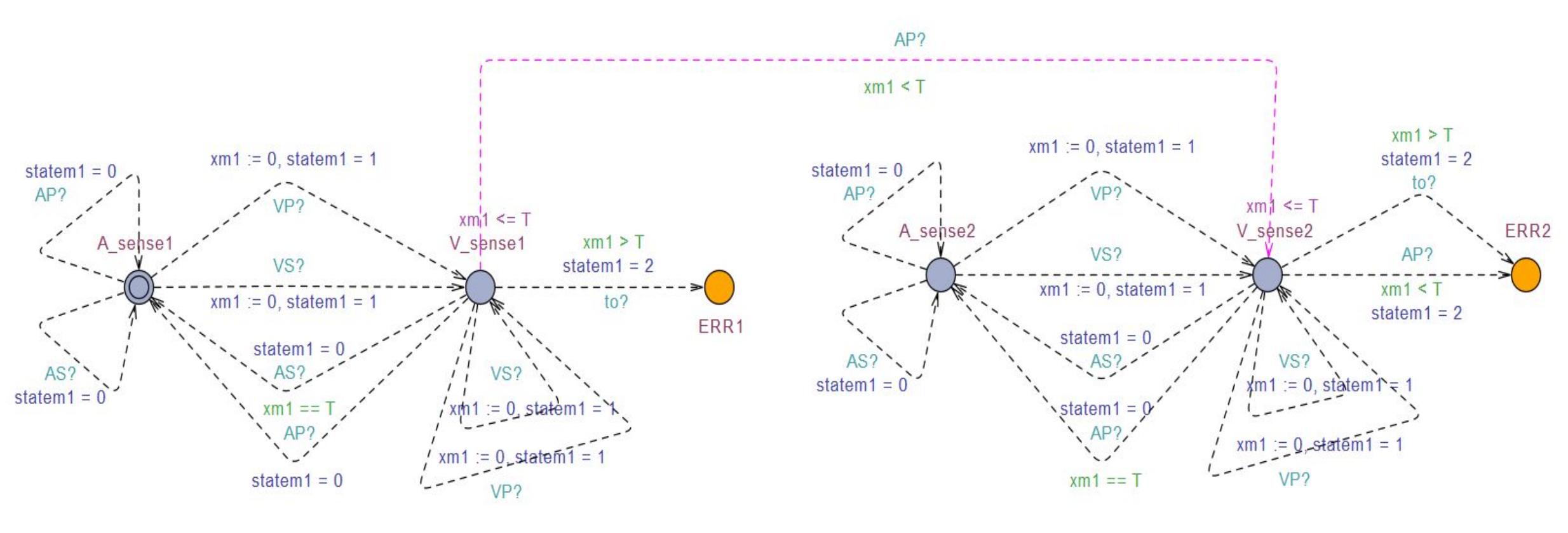


- Observes the system outputs and check if it goes into the error state.
- All the transitions here are uncontrollable.
  - As it takes outputs of system/ environment as inputs which are not in our control.

Example: Heart-Pacemaker (Fault Models)

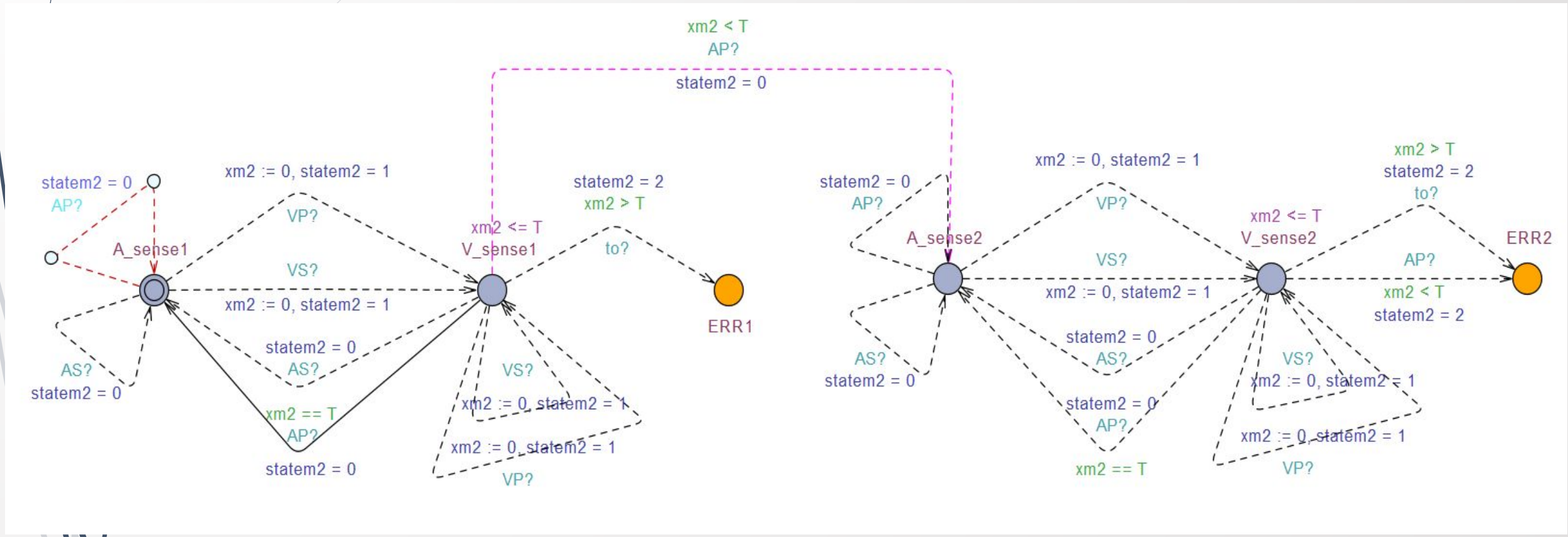
- As there can be multiple faults (leading to error state) in the system.
- Each fault will require multiple fault models for detection.
- Here, we consider the following fault and create the fault models for it:
  - We are in location V_sense and encountered an AP input leading to error state.
- We can observe through shield that when we encountered a fault i.e. error == 1, we have 2 choices to make:
  - wait in the current location
  - move to location A_sense (when clock value <= T)

Example: Heart-Pacemaker (Fault Model 1)



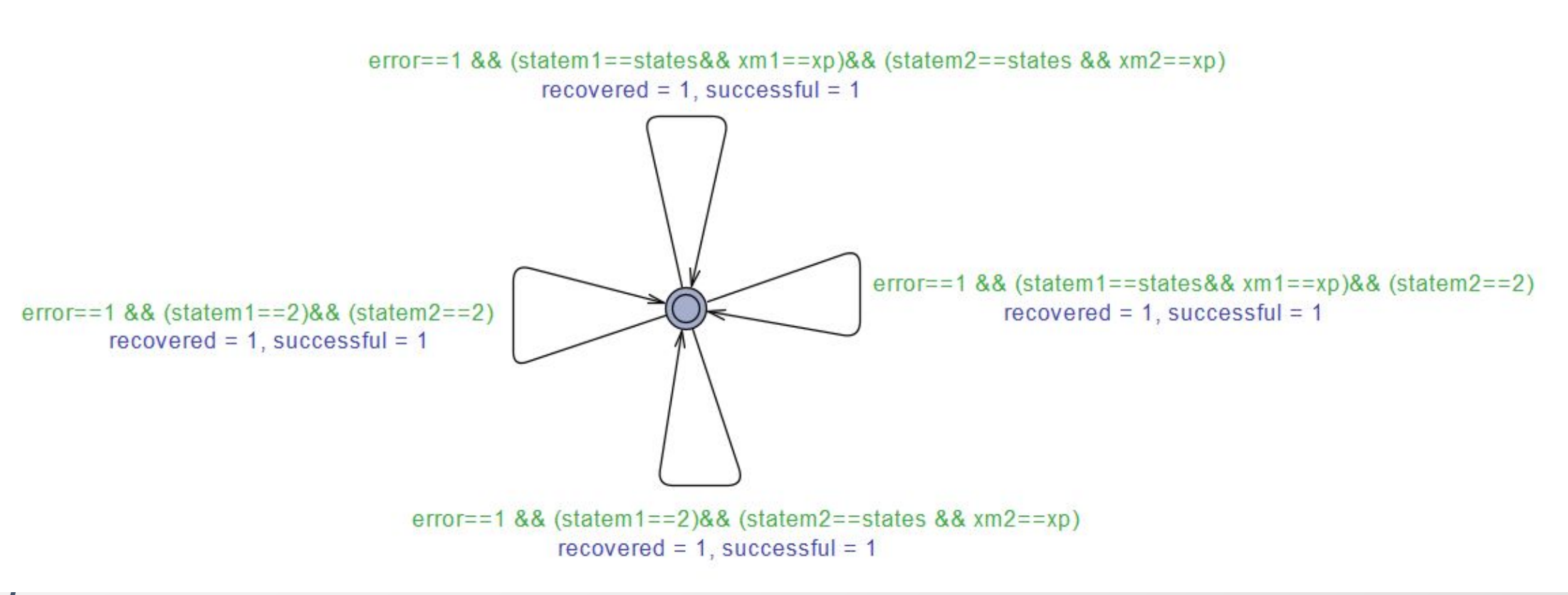
- It is a fault model that handle “wait into the current state” choice by taking the transition to V_sense1 on encountering the fault.

Example: Heart-Pacemaker (Fault Model 2)



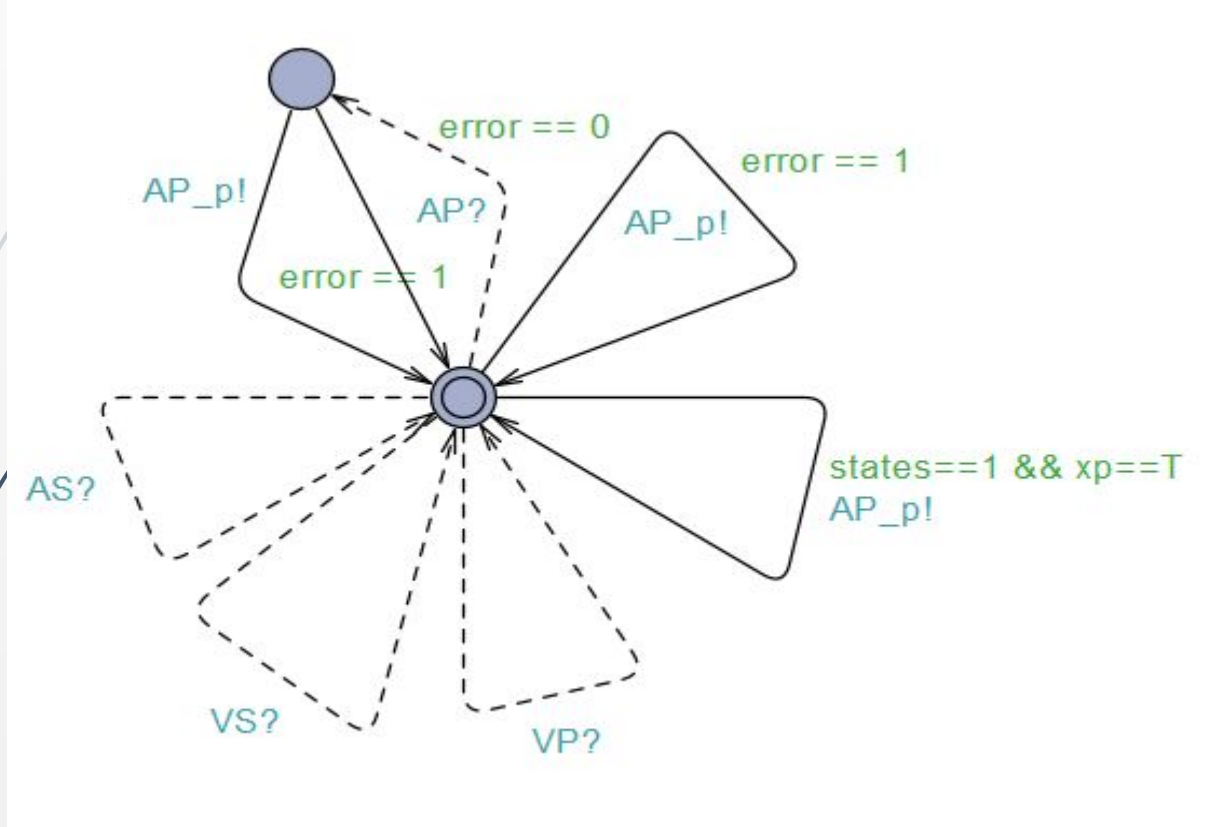
- It is a fault model that handles that “move to location A_sense” choice by taking a transition to A_sense1 on encountering the fault.

Example: Heart-Pacemaker (Recovery Model)



- Used to check whether the fault models are aligning with the system state or fault model has gone into error state (i.e. it is aligning with the specification monitor or not).
- If one of the fault models are aligning, then the model has recovered successfully, with the fault being the one depicted by that fault model.

Example: Heart-Pacemaker System (Shield)

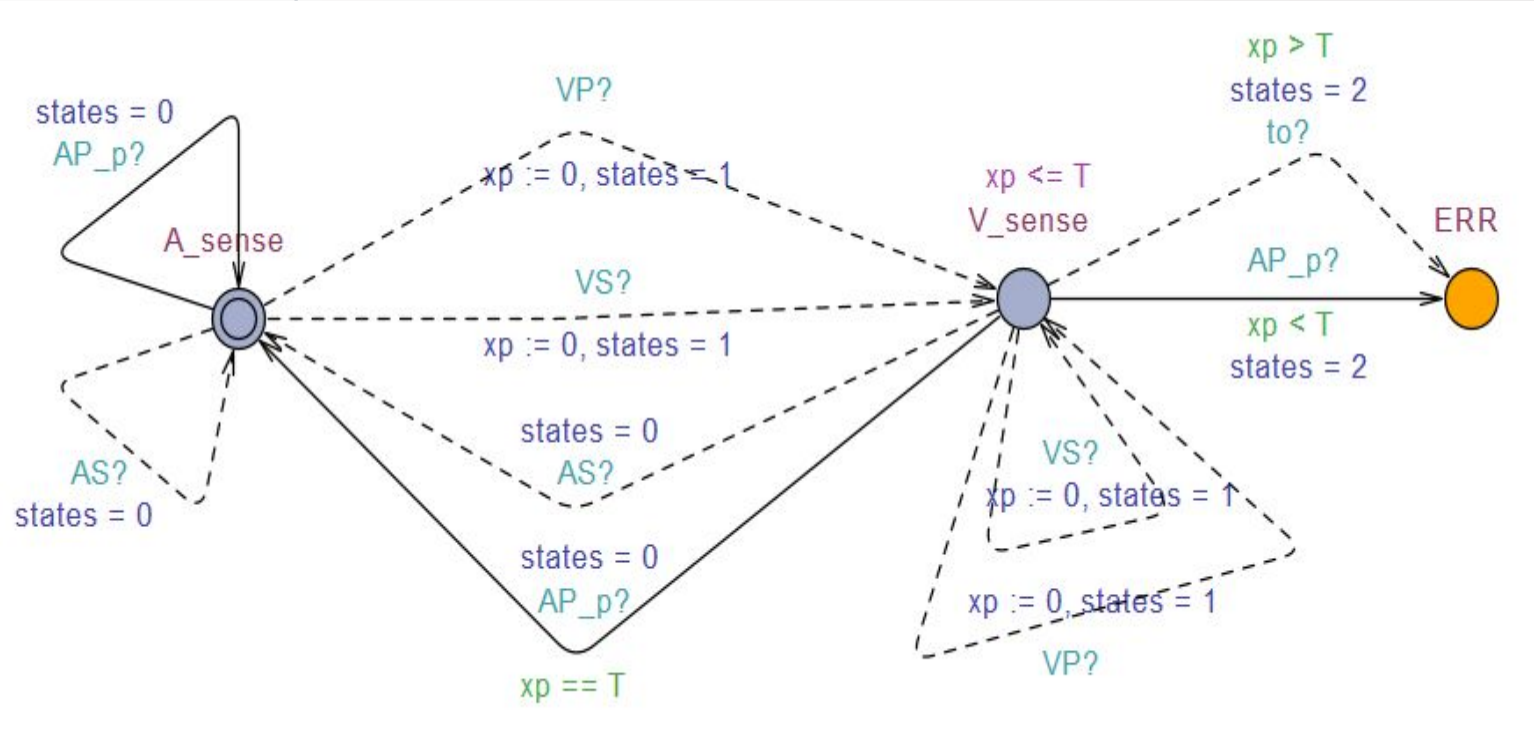


- AS, VS and VP are uncontrollable.
- AP (output of system) is controlled by the shield.

- Shield has 3 parts:
  - Pre-fault actions
  - Post-fault actions
  - Last-chance actions

- error variable indicates whether the system goes into the error state or not.

Example: Heart-Pacemaker System (Specification Monitor for the Shield)



- It observes the output of the shield and check if it produces the correct output or not (goes into the error state).
- All the transitions which are outputs of system and environment are uncontrollable in nature (dotted lines).
- Controllable transitions are the ones which are the outputs of the shield (solid lines).

Example: Heart-Pacemaker (Verification)

```
strategy safe = control: A[] not mSpec_p.ERR
A[] mSpec_p.V_sense imply xp <= T under safe
strategy REC = control: A[] not mSpec_p.ERR and ((not mSpec.ERR) or x <= T1 or recovered)
E<> successful ==1 under REC
saveStrategy("pacemaker_fault_model.txt", REC)
```

strategy safe = control: A[] not mSpec_p.ERR
Verification/kernel/elapsed time used: 0.016s / 0s / 0.017s.
Resident/virtual memory usage peaks: 14,020KB / 39,888KB.
Property is satisfied.
A[] mSpec_p.V_sense imply xp <= T under safe
Verification/kernel/elapsed time used: 0s / 0s / 0.012s.
Resident/virtual memory usage peaks: 13,996KB / 40,680KB.
Property is satisfied.

strategy REC = control: A[] not mSpec_p.ERR and ((not mSpec.ERR) or x <= T1 or recovered)
Verification/kernel/elapsed time used: 0.016s / 0s / 0.019s.
Resident/virtual memory usage peaks: 14,464KB / 40,764KB.
Property is satisfied.

E<> successful ==1 under REC
Verification/kernel/elapsed time used: 0s / 0s / 0.001s.
Resident/virtual memory usage peaks: 14,164KB / 41,100KB.
Property is satisfied.

Example: Heart-Pacemaker (Results)

Objectives:

- Implement the timed shield. ✅
- Implement fault models. ✅
  - We implement fault models with recovery mechanism for one scenario in the UPPAAL tool.
- Find winning strategy: ✅
  - First query creates a winning strategy for the safety property.
  - Third query creates a winning strategy with guaranteed time bounded recovery.
  - Strategy can also be saved in a .txt file using keyword **strategy** for further usage.
- Verification of safety property: ✅
  - The fourth query shows that the shielded system satisfy the safety property with guaranteed time bounded recovery.
  - Keyword **under** is used to compose the time game automata with the winning strategy **REC**.

Limitations of the Algorithm

- Recovers the system in case of **transient faults** only.
- In case another fault is encountered afterwards, then shield will not be able to recover i.e. cannot go in sync with the system.
- To capture all transient faults in a system, a large number of fault models is needed.
- It may not be feasible to synthesize shields with guaranteed recovery for large specifications considering a large number of fault models.

Future Work

- We could apply the timed shield algorithm and the work to other real life applications as well. We could think of more such case study.
- We could explore other real life applications as cyber physical system with real values as input and output.
- All the work done by us can be seen here: https://github.com/BTP-CPS/Phase-1

Questions?