CS 499: 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)

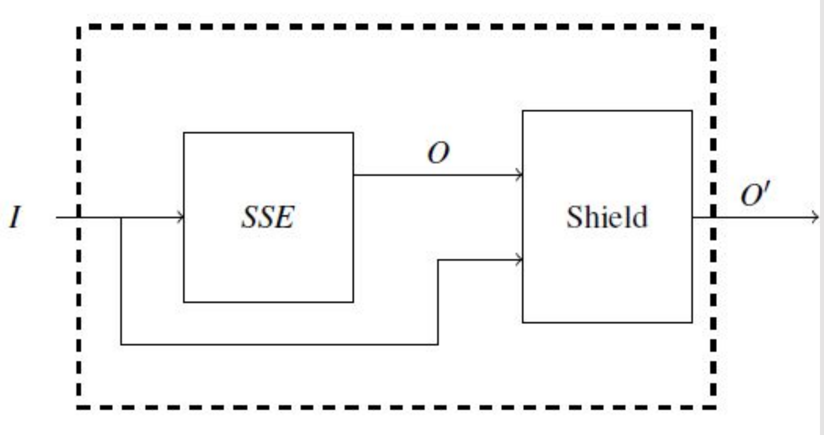
Solution: Shield Synthesis

- 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.

<u>Existing Approach 1</u>

Run-time Enforcement Shield using QDCC

Introduction



Shield:

- Attached after the system,
- Monitors its inputs I and outputs O,
- Corrects and forwards the corrected output O'.

Desired Properties of the shield:

- Correctness
- Deviation: Shield must deviate as little as possible from the system.

Terms:

- SSE: system with sporadic errors
- REQ(I,O): correctness requirement
- HDC: hard deviation constraint
  - Hard requirement satisfied to ensure deviation property of the shield
  - ex: k-stabilizing shield

Soft Deviation Constraint

- Further optimize the deviation
- Soft requirement formulas using weights
- Maximizes the expected value of cumulative weight over next r-steps

$$Hamming(O, O') = \langle (true \hat{} \langle o_1 = o'_1 \rangle) : 1, \ \ldots \ , (true \hat{} \langle o_r = o'_r \rangle) : 1 \rangle$$

- Non-deviation of any output variable at any position gives a reward of 1.
- Summed over to give weight of soft requirement.
- Horizon value (H).
- H-optimally deviation minimizing shield.

Determinization

- Multiple choices of outputs.
- Satisfy HDC and being H-optimal for soft deviation constraint.
- Preference ordering by the user to obtain deterministic controller.

QDDC (Quantified Discrete Duration Calculus)

- Interval temporal logic for specifying REQ(I, O) and HDC.
- Formulas represented by a finite state automaton.
- Decidability checked using DCVALID tool.
- More sophisticated than LTL.

QDDC: Traffic Light Example

Let R, Y, G denote the red, yellow, green boolean signals of traffic light respectively.

- Once red, the light cannot become green immediately.

LTL: $\Box(R \rightarrow \neg \circ G)$

QDDC: $\Box(\lceil R \rceil^0 \wedge \eta = 2 \rightarrow true \wedge \lceil \neg G \rceil^0)$

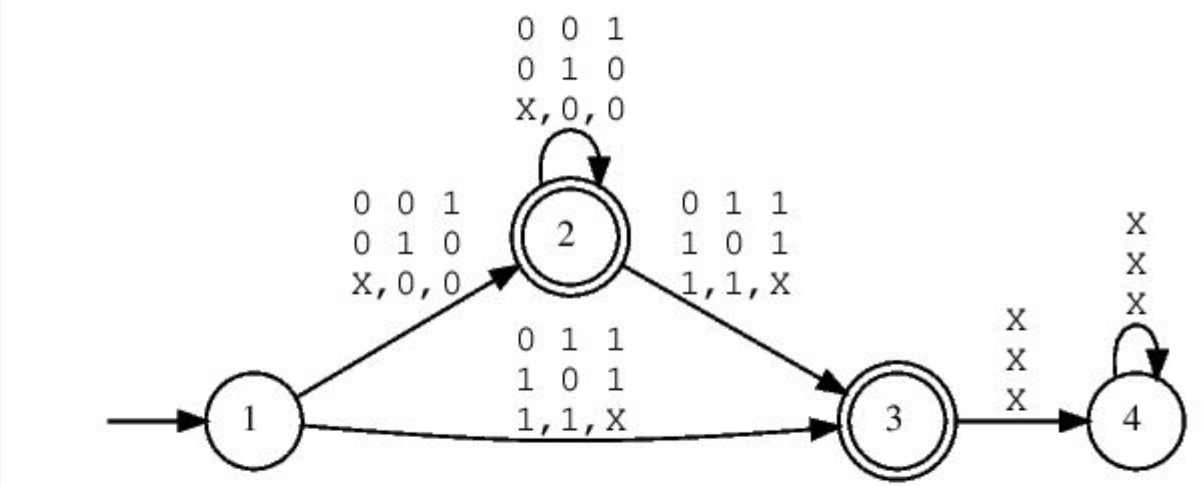
- No two traffic lights can blink simultaneously.

LTL: $\Box((R \rightarrow \neg(Y \vee G)) \,\&\&\, (Y \rightarrow \neg(R \vee G)) \,\&\&\, (G \rightarrow \neg(R \vee Y)))$

QDDC: $\Box((R \rightarrow \neg(Y \vee G)) \,\&\&\, (Y \rightarrow \neg(R \vee G)) \,\&\&\, (G \rightarrow \neg(R \vee Y)))$

DCVALID: Traffic Light Example

```
ANALYSIS
A counter-example of least length (2) is:
R                X 0X
Y                X 1X
G                X 1X

R = {}
Y = {0}
G = {0}

A satisfying example of least length (1) is:
R                X 0
Y                X 0
G                X X

R = {}
Y = {}
G = {}

Total time: 00:00:00.00
```



No two traffic lights can blink simultaneously.

Existing Approach 2

Run-time Enforcement Shield using Slugs

Introduction: Slugs

- Compute implementation from the specifications.

- Slugs architecture allows one to use multiple plugins at the same time, where each plugin only modies a part of the synthesis process.

```
src/slugs <Options> [InputFileName.slugs]
```

  Here options denotes the plugins that can be used to build the strategy.

- Specification as text file in two formats:
  - slugs-in format
  - structured slugs format

Introduction: Slugs-in

- Multiple sections, started by section headers. For ex:
  - [INPUT] / [OUTPUT] for atomic input and output propositions resp.
  - [SYS_TRANS] for system safety guarantees.
  - [ENV_TRANS] for environment safety assumptions.
  - [SYS_LIVENESS] for system liveness assumptions.
  - [ENV_LIVENESS] for environment liveness assumptions.
  - [SYS_INIT] for system initialization guarantees.
  - [ENV_INIT] for environment initialization assumptions.

- Boolean input/output variables.
- Constraints in prefix notation.
- Operators: !, &, |, “| !” as →, “! ^” as ↔.
- # for single line comment.

Introduction: Structured Slugs

- Strict extension of slugs tool keeping basic structure similar.

- Two extensions:
  - Using infix notation in the constraints
  - Support for non-negative integer variables.

- To run the tool, convert structured slugs to slugs-in format using:

```
tools/StructuredSlugsParser/compiler.py [InputFile.
                    structuredslugs] > [OutputFile.slugsin]
```

Slugs Example: Water Reservoir

```
[INPUT]
inflow1
inflow2

[OUTPUT]
level: 3...107
outflow

[SYS_TRANS]
(inflow1 & inflow2 & outflow) -> (level' = level+1)
(inflow1 & inflow2 & !outflow) -> (level' = level+4)
(inflow1 & ! inflow2 & outflow) -> (level'+1 = level)
(inflow1 & ! inflow2 & !outflow) -> (level' = level+2)
(!inflow1 & inflow2 & outflow) -> (level'+1 = level)
(!inflow1 & inflow2 & !outflow) -> (level' = level+2)
(!inflow1 & ! inflow2 & outflow) -> (level'+3 = level)
(!inflow1 & ! inflow2 & !outflow) -> (level' = level)

(level'<=100)
(level'>=10)

[SYS_INIT]
! outflow
level = 10

[ENV_INIT]
! inflow1
! inflow2

[ENV_TRANS]
| ! inflow1 ! inflow1'
| ! inflow2 ! inflow2'
```

Specification:

- Maintain the level of water between 10 and 100 with the help of 2 inflow pipes and 1 outflow pipes.
- Inflow pipe increase the water level by 2 per unit. Outflow decrease the water level by 3 per unit.

- Written in structured slugs format.
- 2 boolean input variables: inflow1, inflow2
- 1 boolean output variable: outflow
- 1 integer output variable: level

Slugs Example: Water Reservoir

```
samay@samay-VM:~/slugs$ src/slugs --explicitStrategy --jsonOutput examples/water_reservoir.slugsin
SLUGS: SmaLl bUt complete Gr(1) Synthesis tool (see the documentation for an author list).
RESULT: Specification is realizable.
{"version": 0,
 "slugs": "0.0.1",

 "variables": ["inflow1", "inflow2", "level@0.3.107", "level@1", "level@2", "level@3", "level@4", "level@5", "level@6", "outflow"],

 "nodes": {
"0": {
        "rank": 0,
        "state": [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        "trans": [0, 1, 2, 3]
},

"1": {
        "rank": 0,
        "state": [0, 1, 1, 1, 1, 0, 0, 0, 0, 0],
        "trans": [4, 5]
},
```

Slugs Example: Un-realizable Scenario

```
[INPUT]
a
b

[OUTPUT]
x
y


[ENV_INIT]
! a
! b

[SYS_INIT]
! x
! y

[ENV_LIVENESS]
& a ! b
& b ! a

[SYS_LIVENESS]
& a b

[SYS_TRANS]
| ! x ! y
```

```
samay@samay-VM:~/slugs$ src/slugs --explicitStrategy --jsonOutput examples/unrealizable1.slugsin
SLUGS: SmaLl bUt complete Gr(1) Synthesis tool (see the documentation for an author list).
RESULT: Specification is unrealizable.
```

- Written in slugs-in format (no integer variable).
- 2 boolean input variables: a, b
- 2 boolean output variables: x, y

Simple Safety Example

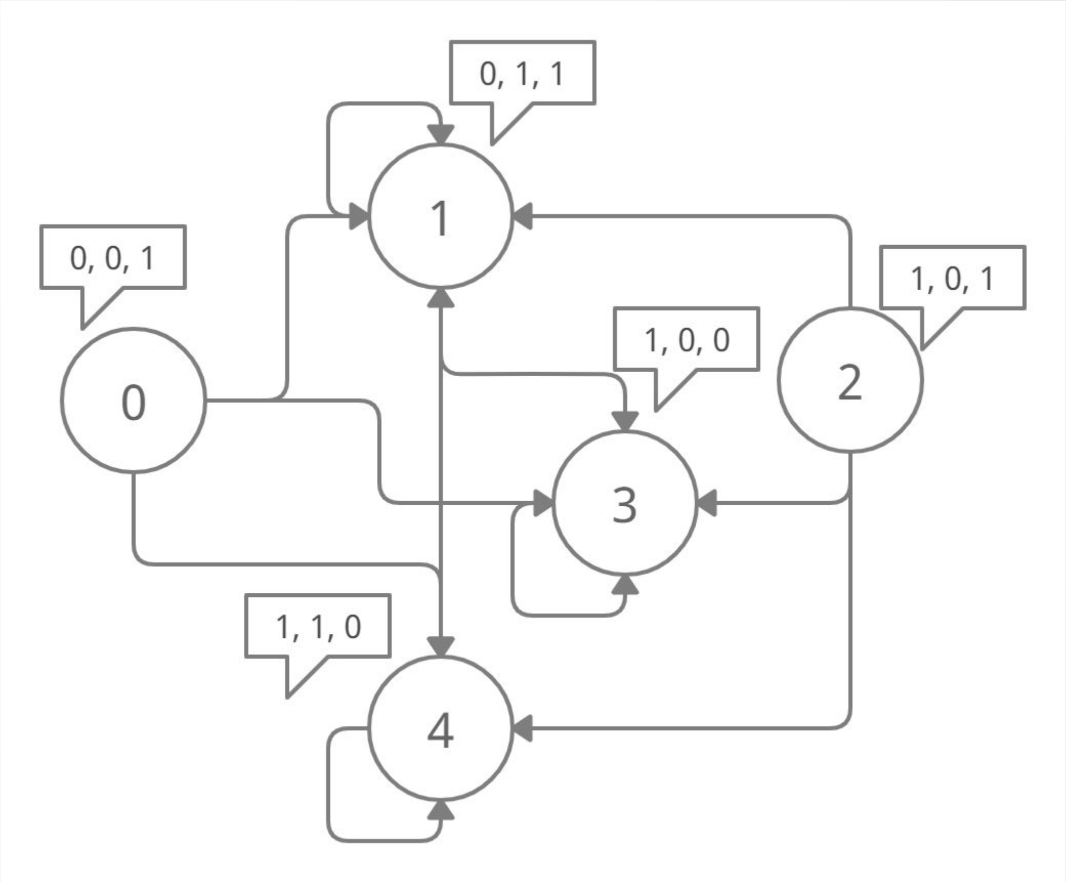
```
[INPUT]
a
b

[OUTPUT]
c

[ENV_INIT]
| ! a ! b

[SYS_INIT]
c

[ENV_TRANS]
| a' b'

[SYS_TRANS]
^ c' a'
```

```
samay@samay-VM:~/slugs$ src/slugs --explicitStrategy examples/simple_safety_example.slugsin
SLUGS: SmaLL bUt complete Gr(1) Synthesis tool (see the documentation for an author list).
RESULT: Specification is realizable.
State 0 with rank 0 -> <a:0, b:0, c:1>
        With successors : 1, 3, 4
State 1 with rank 0 -> <a:0, b:1, c:1>
        With successors : 1, 3, 4
State 2 with rank 0 -> <a:1, b:0, c:1>
        With successors : 1, 3, 4
State 3 with rank 0 -> <a:1, b:0, c:0>
        With successors : 1, 3, 4
State 4 with rank 0 -> <a:1, b:1, c:0>
        With successors : 1, 3, 4
```

- Initial states: 0, 1, 2.
- Environment transition states in next step either a or b must be true: state 1, 2, 3, 4 holds.
- System transition states in next step exactly one of c and a must hold: state 1, 3, 4 holds.

14

Simple Safety Example



Questions?