



RESEARCH REVIEW 2025

Carnegie  
Mellon  
University  
Software  
Engineering  
Institute

# Formalizing and Automating STPA with Robustness (FASR)

NOVEMBER 14–16, 2025

**Sam Procter**, Keaton Hanna, Lutz Wrage  
Eunsuk Kang, Ian Dardik, Yining She

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

©2025 Carnegie Mellon University

Innovation in Action



# Document Markings

Copyright 2025 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of War under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

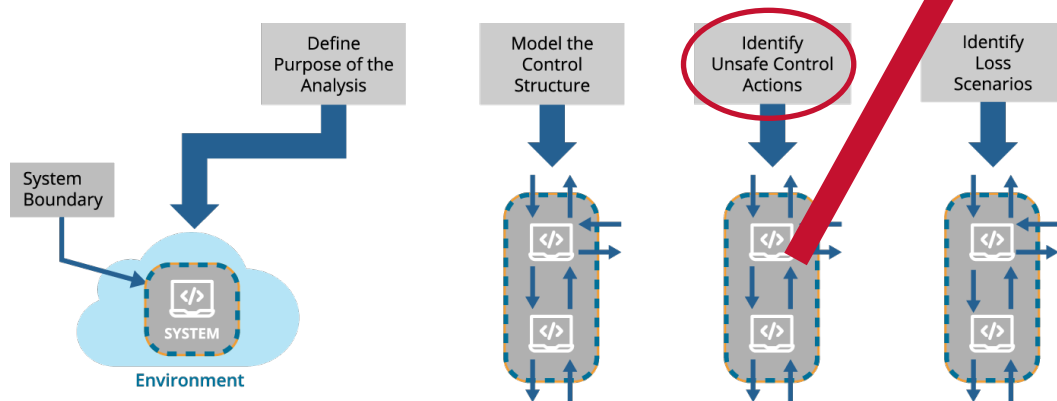
This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

DM25-1396

# Background: Safety Analysis Is Slow, Hard, and You Don't Know When You're Done

Most critical systems are evaluated for safety before use, often using *hazard analysis* techniques.

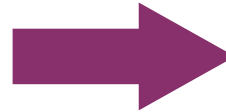
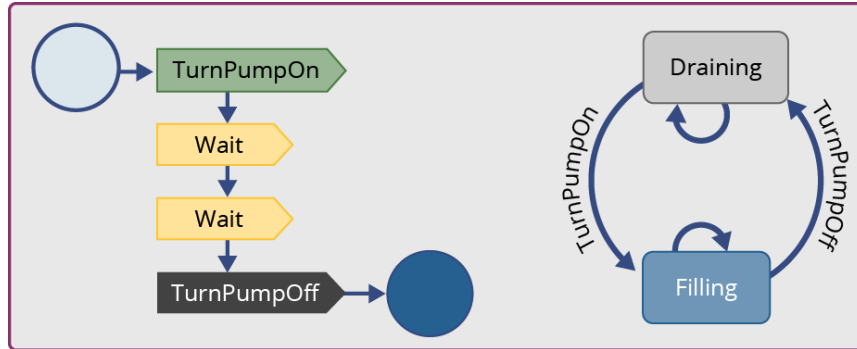
Control Action	Not Providing Causes Hazard	Providing Causes Hazard	Too Early, Too Late, Out of Order	Stopped Too Soon, Applied Too Long
Brake	UCA-1: Brake Does Not Engage	UCA-2: Brake Engages During Takeoff	UCA-3: Brake Engages Too Late After Touchdown	UCA-4: Brake Disengages Before Safe Taxi Speed is Attained



Research question: Can we use *formal methods* to improve analysis **speed** and **accuracy** while providing a **measure of completeness**?

# Goal: Automatically Calculate Unsafe Control Actions from System Models

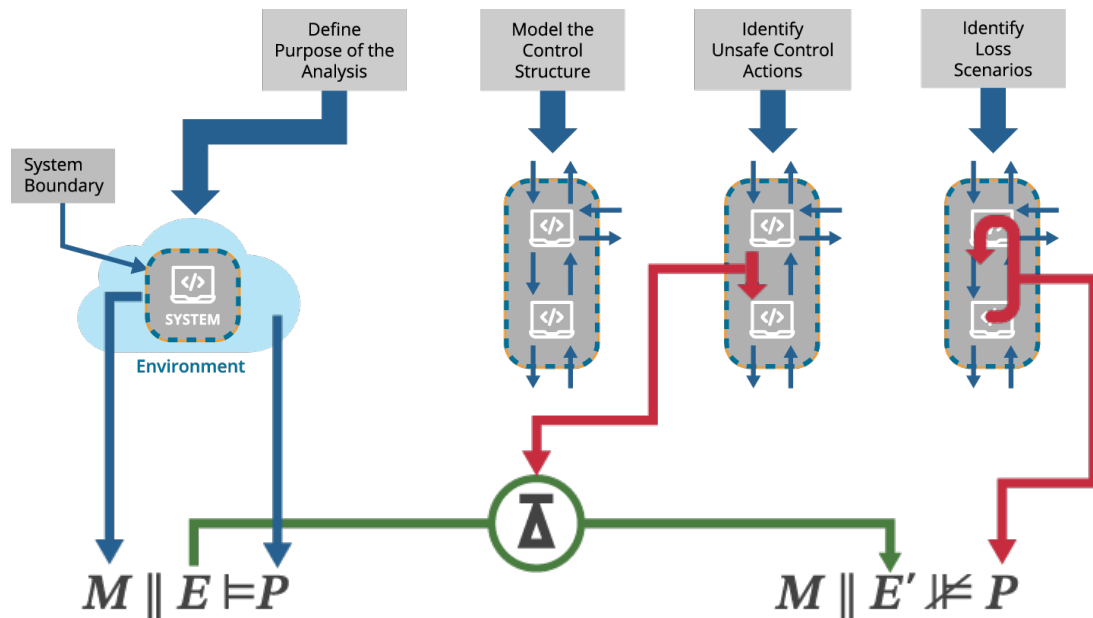
## SysML



## RAAML/STPA

	Providing Causes Hazard	Not Providing Causes Hazard
TurnPumpOff		1. TurnPumpOn 2. Wait 3. Wait
TurnPumpOn	1. TurnPumpOn 2. Wait 3. Wait 4. TurnPumpOff	

# Robustness: Safety in the Presence of Environmental Deviations

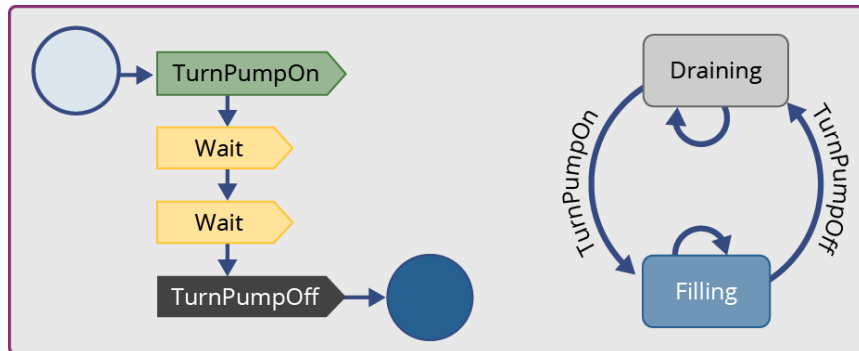


“... a system is *robust* with respect to a *property* and a particular set of *environmental deviations* if the system continues to satisfy the property even if the environment exhibits those deviations.”

Zhang, Changjian, David Garlan, and Eunsuk Kang. 2020. “A Behavioral Notion of Robustness for Software Systems.” *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA), November 8, 111–22.  
<https://doi.org/10.1145/3368089.3409753>.

# Method: Translate SysML into Precise Specification

## SysML



## TLA+

Module Tank

Extends Integers

Variables WaterLevel, PumpOn  
 vars = <<waterLevel,  
 pumpOn>>

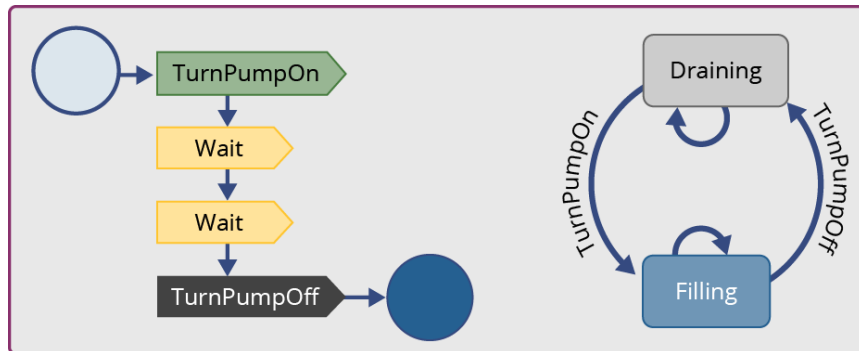
Init ==  
 ∧ waterLevel = 0  
 ∧ pumpOn = False  
 TurnPumpOn ==...

## RAAML/STPA

	Providing Causes Hazard	Not Providing Causes Hazard
TurnPumpOff		1. TurnPumpOn 2. Wait 3. Wait
TurnPumpOn	1. TurnPumpOn 2. Wait 3. Wait 4. TurnPumpOff	

# Method: Use CMU *Fortis* to Find Unsafe Behaviors

## SysML



## RAAML/STPA

	Providing Causes Hazard	Not Providing Causes Hazard
TurnPumpOff		1. TurnPumpOn 2. Wait 3. Wait
TurnPumpOn	1. TurnPumpOn 2. Wait 3. Wait 4. TurnPumpOff	

## TLA+

Module Tank

Extends Integers

Variables WaterLevel, PumpOn  
 vars = <<waterLevel,  
 pumpOn>>

Init ==  
 ∧ waterLevel = 0  
 ∧ pumpOn = False  
 TurnPumpOn ==...

## JSON

```

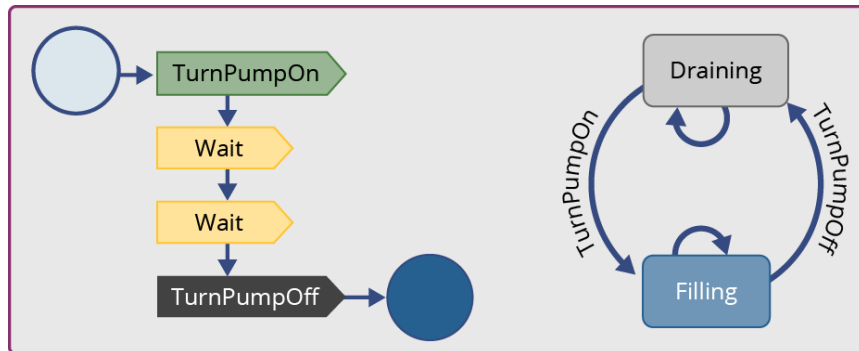
[
  {
    "goodTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
    "badTrace": ["TurnPumpOn", "wait", "wait", "wait"]
    "vioComp": ["WaterTank"]
    "vioInv": ["NoOverflow"]
  }
]
  
```

```

"good Trace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
"badTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff", "TurnPumpOn", "wait"]
"vioComp": ["WaterTank"]
"vioInv": ["NoOverflow"]
}
]
  
```

# Method: Categorize Behaviors with STPA Guidewords

## SysML



## RAAML/STPA

	Providing Causes Hazard	Not Providing Causes Hazard
TurnPumpOff		<ol style="list-style-type: none"> <li>1. TurnPumpOn</li> <li>2. Wait</li> <li>3. Wait</li> </ol>
TurnPumpOn	<ol style="list-style-type: none"> <li>1. TurnPumpOn</li> <li>2. Wait</li> <li>3. Wait</li> <li>4. TurnPumpOff</li> </ol>	

## TLA+

Module Tank

Extends Integers

Variables WaterLevel, PumpOn  
vars = <<waterLevel,  
pumpOn>>

Init ==  
 $\wedge$  waterLevel = 0  
 $\wedge$  pumpOn = False  
 TurnPumpOn ==...

## JSON

```
[
  {
    "goodTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
    "badTrace": ["TurnPumpOn", "wait", "wait", "wait"]
    "vioComp": ["WaterTank"]
    "vioInv": ["NoOverflow"]
  }
]
```

```
"good Trace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
"badTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff", "TurnPumpOn", "wait"]
"vioComp": ["WaterTank"]
"vioInv": ["NoOverflow"]
}
]
```

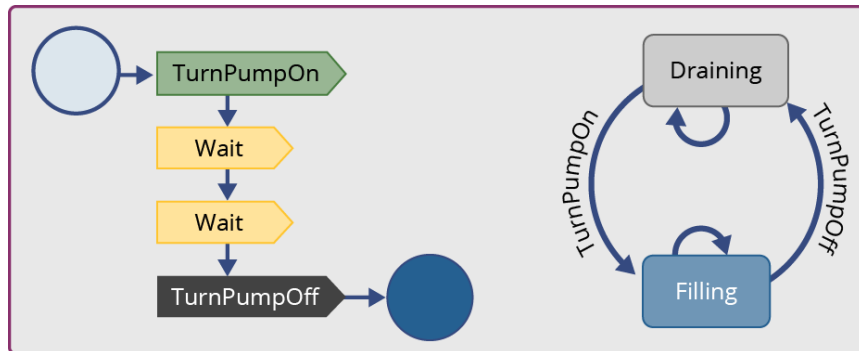
## JSON

```
[
  {
    "source": "WaterTank"
    "guideword": "NOT_PROVIDING"
    "controlAction": "TurnPumpOff"
    "context": ["TurnPumpOn", "wait", "wait"]
    "violatedConstraint": "NoOverflow"
  }
]
```



# Method: Display STPA Output Using RAAML

## SysML



## RAAML/STPA

	Providing Causes Hazard	Not Providing Causes Hazard
TurnPumpOff		1. TurnPumpOn 2. Wait 3. Wait
TurnPumpOn	1. TurnPumpOn 2. Wait 3. Wait 4. TurnPumpOff	

## TLA+

Module Tank

Extends Integers

Variables WaterLevel, PumpOn  
 vars = <<waterLevel,  
 pumpOn>>

Init ==  
 ∧ waterLevel = 0  
 ∧ pumpOn = False  
 TurnPumpOn ==...

## JSON

```
[
  {
    "goodTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
    "badTrace": ["TurnPumpOn", "wait", "wait", "wait"]
    "vioComp": ["WaterTank"]
    "vioInv": ["NoOverflow"]
  }
]
```

```

"good Trace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff"]
"badTrace": ["TurnPumpOn", "wait", "wait", "TurnPumpOff", "TurnPumpOn", "wait"]
"vioComp": ["WaterTank"]
"vioInv": ["NoOverflow"]
}
]
```

## JSON

```
[
  {
    "source": "WaterTank"
    "guideword": "NOT_PROVIDING"
    "controlAction": "TurnPumpOff"
    "context": ["TurnPumpOn", "wait", "wait"]
    "violatedConstraint": "NoOverflow"
  }
]
```

# Team Contact

Email:

[info@sei.cmu.edu](mailto:info@sei.cmu.edu)



**Sam  
Procter**

PI, Sr. Architecture  
Researcher



**Keaton  
Hanna**

Associate Software  
Engineer



**Lutz  
Wrage**

Senior Member of  
the Technical Staff



**Eunsuk  
Kang**

Associate Professor



**Ian  
Dardik**

Ph.D. Student



**Yining  
She**

Ph.D. Student



For more information, follow this QR code to  
[https://cmu-soda.github.io/projects/project\\_fasr.html](https://cmu-soda.github.io/projects/project_fasr.html)