Carnegie Mellon University
Software Engineering Institute

FASR: Formalizing and Automating STPA with Robustness

APRIL 14, 2026

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

Document Markings

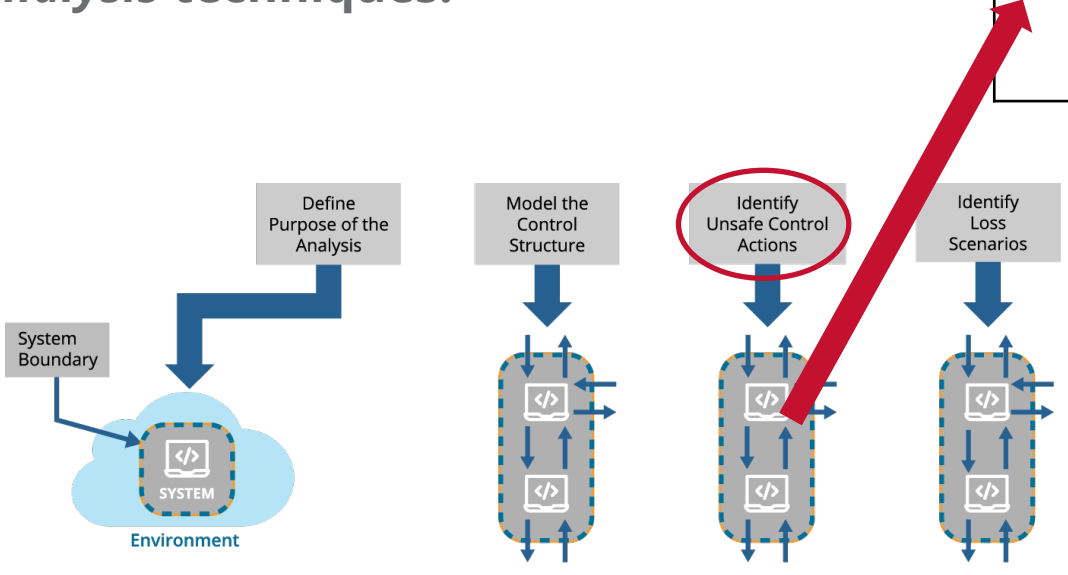
This material is based upon work supported by the Department of War under Air Force Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM26-0397

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



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

Agenda

Background: Three Technologies

- STPA: System-Theoretic Process Analysis
- Model-Based Engineering
- Robustness

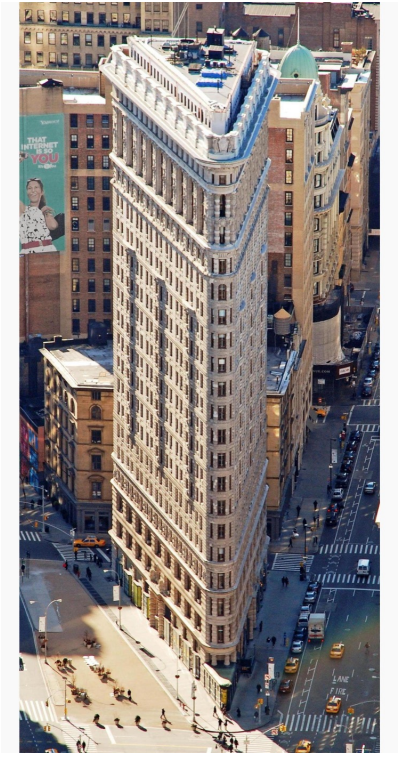
Case Study: Braking System Control Unit (BSCU)

FASR Tool

Exploratory Study

Future Work

Model-Based Engineering



What is the purpose of a model?

- Expose some details
- Hides others

What's not in a model?

- As important (or more!) than what's included
- Enables focus on what matters

Lego image; By Sijysuis - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=48346782

Photograph: By Imelenchon (original work) - This is a retouched picture, which means that it has been digitally altered from its original version. Modifications: cropped. The original can be viewed here: Edificio Fuller (Flatiron) en 2010 desde el Empire State.jpg: . Modifications made by Beyond My Ken., Public Domain, https://commons.wikimedia.org/w/index.php?curid=12762087

MBE + Safety: RAAML

Risk Analysis and Assessment Modeling Language

RAAML

OMG RISK ANALYSIS AND ASSESSMENT MODELING LANGUAGE

SysML: Graphical modeling language for systems

RAAML: Extension of SysML for risk – safety, security, etc.

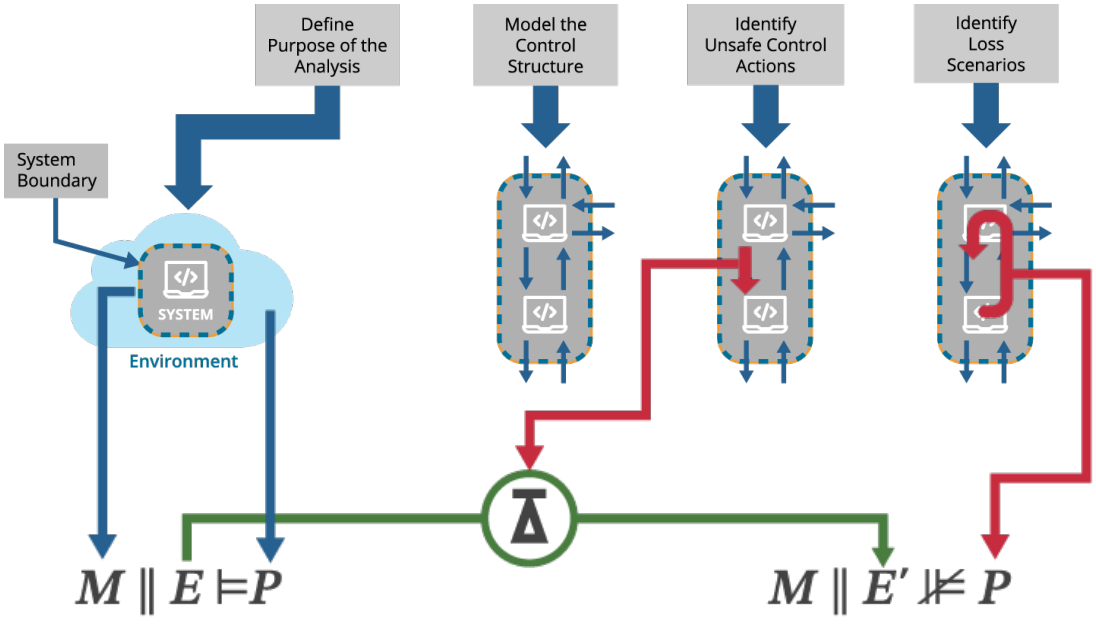
- Has an STPA “view”

Cameo Enterprise Architecture (CEA): Modeling software for systems

- Supports RAAML as of 2024

Risk Analysis and Assessment Modeling Language (RAAML) Libraries and Profiles Version 1.1 Beta 2. Tech. Rep. ptc/24-03-02, OMG Standards Development Organization (March 2024)

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.

Agenda

Background: Three Technologies

Case Study: Braking System Control Unit (BSCU)

- System Description
- STPA Analysis

FASR Tool

Exploratory Study

Future Work

Example: Aircraft Braking System Control Unit

An aircraft has automated braking functionality (the Braking System Control Unit or BSCU) that engages when the plane lands in order to slow it to a safe speed for taxiing.

The crew must turn on and arm the system before landing, leaving sufficient time for the system to initialize and then arm itself after sending the command to arm.

If any abnormalities in the system are observed during initialization and arming, the crew should first disarm and then turn off the BSCU and operate the brakes manually.

Case Study: Braking System Control Unit (BSCU)

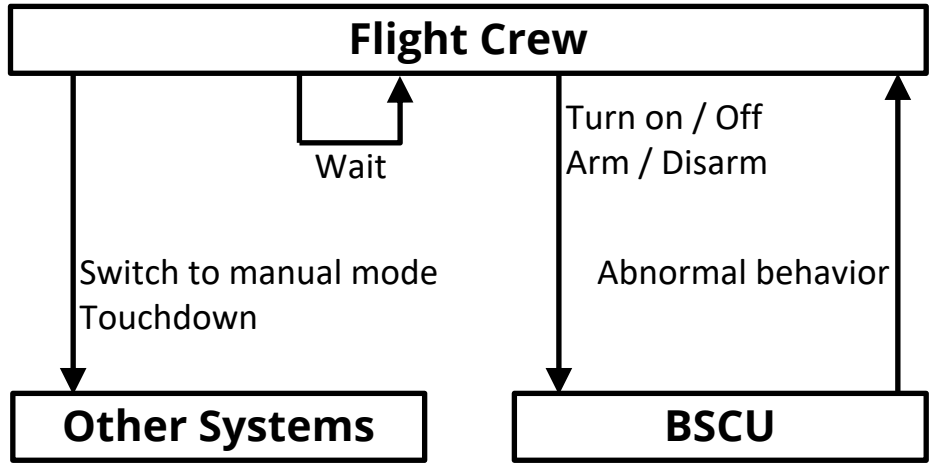
Basic process:

1. Turn on BSCU
2. Arm BSCU
3. If abnormal behavior has been observed:
    1. Disarm BSCU
    2. Turn BSCU off
    3. Engage manual braking
4. Initiate touchdown

Multiple high-quality analyses exist:

- AIR6110 [1]
- MBSA with AADL [2]
- STPA Handbook [3] ← We use this version

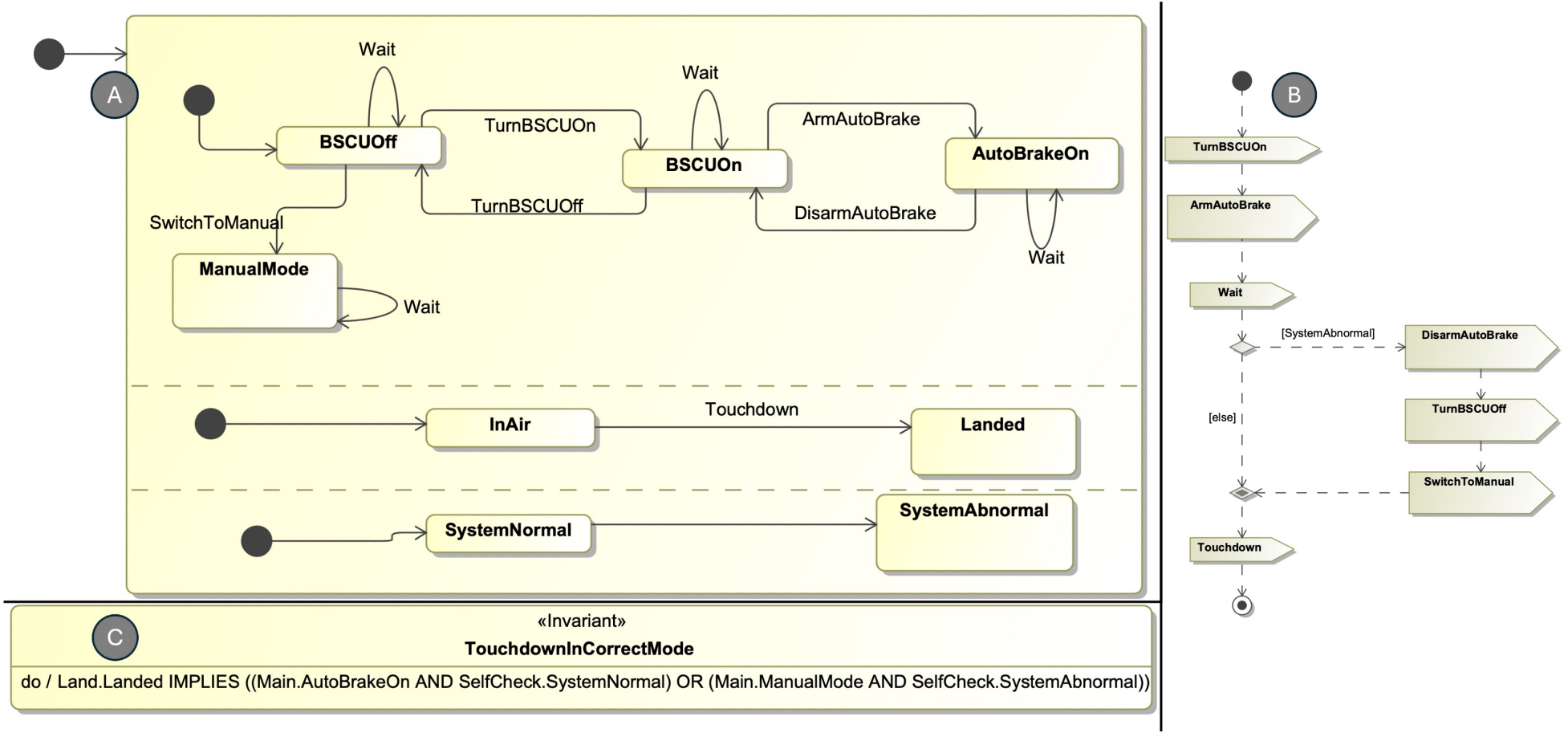
STPA Step 2 Control Structure



[1]: SAE International: AIR 6110 Rev. A: Contiguous Aircraft/System Development Process Example. Tech. Rep. AIR6110A, SAE International (March 2024)
[2]: Stewart, D., Whalen, M.W., Cofer, D., Heimdahl, M.P.E.: Architectural Modeling and Analysis for Safety Engineering. In: Bozzano, M., Papadopoulos, Y. (eds.) Model-Based Safety and Assessment. pp. 97–111. Springer International Publish ing, Cham (2017). https://doi.org/10.1007/978-3-319-64119-5_7
[3]: Leveson, N., Thomas, J.: STPA Handbook. Tech. rep. (2018)

BSCU: Detailed Machine and Process

SysML Modeled using Cameo Enterprise Architecture



BSCU: STPA Analysis

Typically, an Analyst would proceed through the four steps of STPA:

1. Define the Purpose of the Analysis
2. Model the Control Structure
3. Identify Unsafe Control Actions
4. Identify Loss Scenarios

We focus on Step 3: Identifying Unsafe Control Actions (UCAs)

	Provided	**Not Provided**
Arm BSCU	*Not unsafe for this hazard*	Crew turns on the BSCU but does not arm it before landing
Disarm BSCU	Crew turns on and arms the BSCU but then (inadvertently or mistakenly) disarms it	*Not unsafe for this hazard*

Agenda

Carnegie Mellon University Software Engineering Institute

Background: Three Technologies

Case Study: Braking System Control Unit (BSCU)

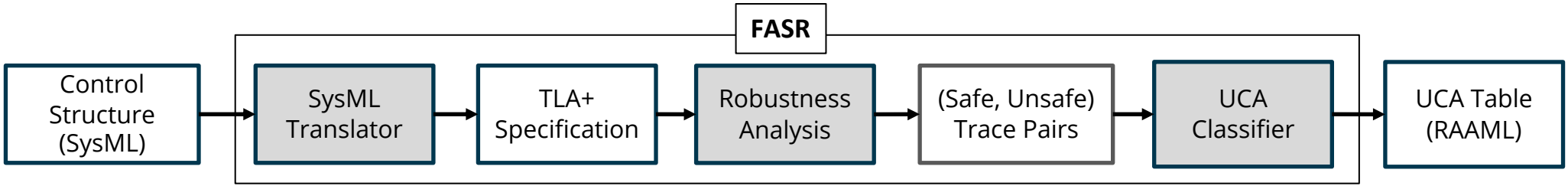
FASR Tool

- SysML → TLA$^+$
- TLA$^+$ → Traces
- Trace Classification
- Performance

Exploratory Study

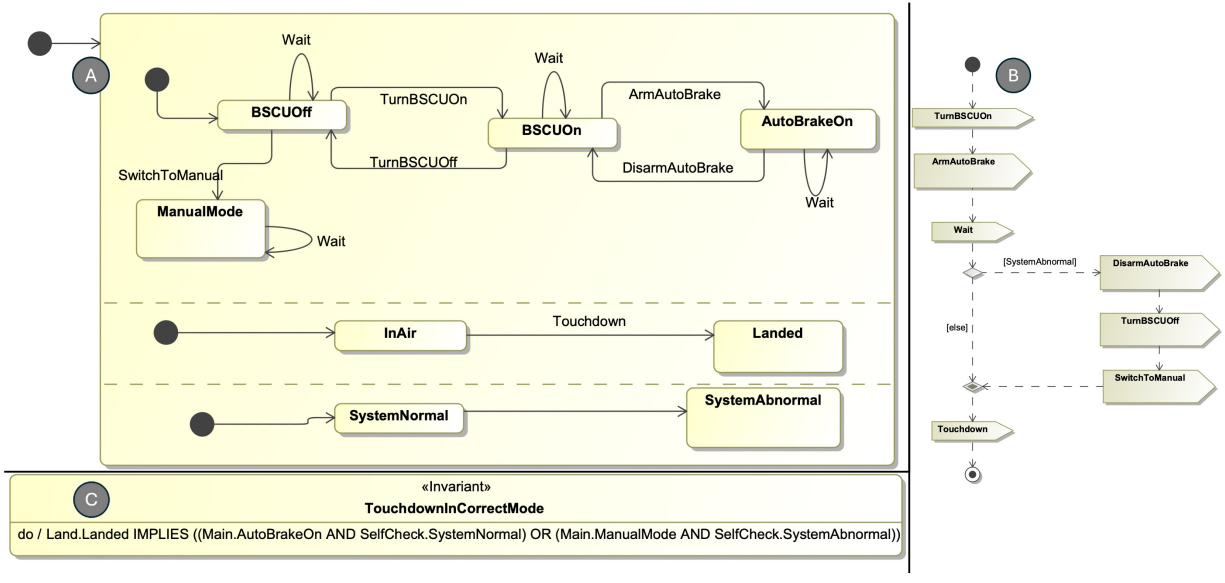
Future Work

FASR Tool: Overview



SysML → TLA+

TLA+ is a formal language for specifying systems. It's amenable to formal analysis.

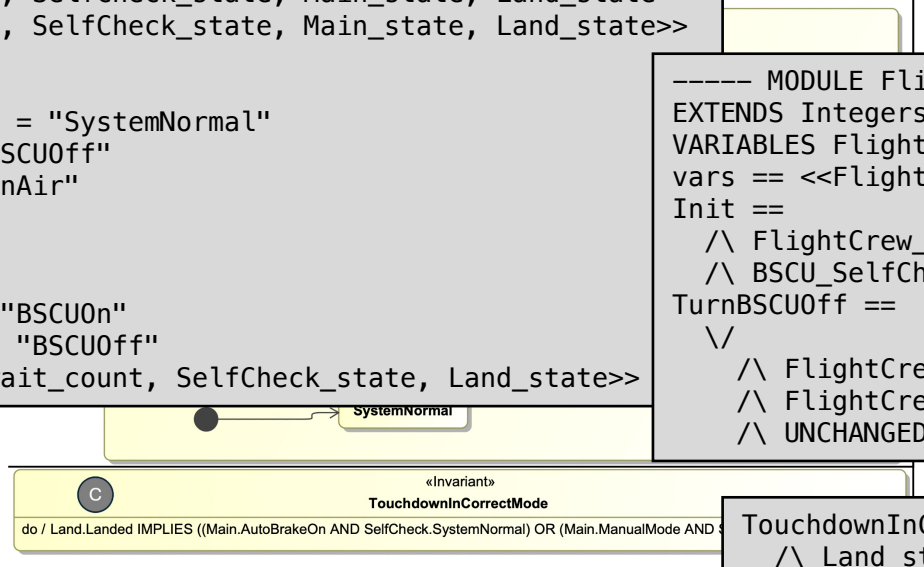


SysML → TLA+

ns. It's amenable to formal analysis.

```
----- MODULE BSCU -----
EXTENDS Integers
VARIABLES wait_count, SelfCheck_state, Main_state, Land_state
vars == <<wait_count, SelfCheck_state, Main_state, Land_state>>
Init ==
  /\ wait_count = 0
  /\ SelfCheck_state = "SystemNormal"
  /\ Main_state = "BSCUOff"
  /\ Land_state = "InAir"

TurnBSCUOff ==
  \/
    /\ Main_state = "BSCUOn"
    /\ Main_state' = "BSCUOff"
    /\ UNCHANGED <<wait_count, SelfCheck_state, Land_state>>
```




```
----- MODULE FlightCrew -----
EXTENDS Integers
VARIABLES FlightCrew_state, BSCU_SelfCheck_SystemAbnormal
vars == <<FlightCrew_state, BSCU_SelfCheck_SystemAbnormal>>
Init ==
  /\ FlightCrew_state = "INIT"
  /\ BSCU_SelfCheck_SystemAbnormal = FALSE
TurnBSCUOff ==
  \/
    /\ FlightCrew_state = "in_DisarmAutoBrake_out_TurnBSCUOff"
    /\ FlightCrew_state' = "in_TurnBSCUOff_out_SwitchToManual"
    /\ UNCHANGED <<BSCU_SelfCheck_SystemAbnormal>>
```

```
TouchdownInCorrectMode ==
  /\ Land_state = "Landed" =>
    \/
      /\ SelfCheck_state = "SystemAbnormal"
      /\ Main_state = "ManualMode"
    \/
      /\ SelfCheck_state = "SystemNormal"
      /\ Main_state = "AutoBrakeOn"
```

SysML → TLA+

ns. It's amenable to formal analysis.

MODULE *BSCU*

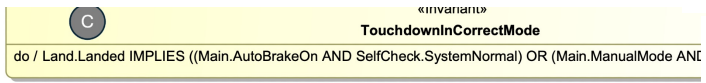
EXTENDS *Integers*

VARIABLES *Main_state*, *wait_count*, *SelfCheck_state*, *Land_state*

$$vars \triangleq \langle Main\_state,\ wait\_count,\ SelfCheck\_state,\ Land\_state \rangle$$

$$\begin{aligned} Init \triangleq\ & \wedge Main\_state = \text{“BSCUOff”} \\ & \wedge wait\_count = 0 \\ & \wedge SelfCheck\_state = \text{“SystemNormal”} \\ & \wedge Land\_state = \text{“InAir”} \end{aligned}$$

$$\begin{aligned} TurnBSCUOff \triangleq\ & \vee \\ & \quad \wedge Main\_state = \text{“BSCUOn”} \\ & \quad \wedge Main\_state' = \text{“BSCUOff”} \\ & \quad \wedge \text{UNCHANGED } \langle wait\_count,\ SelfCheck\_state,\ Land\_state \rangle \end{aligned}$$



MODULE *FlightCrew*

EXTENDS *Integers*

VARIABLES *FlightCrew_state*, *BSCU_SelfCheck_SystemAbnormal*

$$vars \triangleq \langle FlightCrew\_state,\ BSCU\_SelfCheck\_SystemAbnormal \rangle$$

$$\begin{aligned} Init \triangleq\ & \wedge FlightCrew\_state = \text{“INIT”} \\ & \wedge BSCU\_SelfCheck\_SystemAbnormal = \text{FALSE} \end{aligned}$$

$$\begin{aligned} TurnBSCUOff \triangleq\ & \vee \\ & \quad \wedge FlightCrew\_state = \text{“in\_DisarmAutoBrake\_out\_TurnBSCUOff”} \\ & \quad \wedge FlightCrew\_state' = \text{“in\_TurnBSCUOff\_out\_SwitchToManual”} \\ & \quad \wedge \text{UNCHANGED } \langle BSCU\_SelfCheck\_SystemAbnormal \rangle \end{aligned}$$

$$\begin{aligned} TouchdownInCorrectMode \triangleq\ & \wedge Land\_state = \text{“Landed”} \Rightarrow \\ & \quad \vee \\ & \quad\quad \wedge SelfCheck\_state = \text{“SystemAbnormal”} \\ & \quad\quad \wedge Main\_state = \text{“ManualMode”} \\ & \quad \vee \\ & \quad\quad \wedge SelfCheck\_state = \text{“SystemNormal”} \\ & \quad\quad \wedge Main\_state = \text{“AutoBrakeOn”} \end{aligned}$$

TLA⁺ → Traces: Robustness Calculation with Fortis


```
----- MODULE BSCU -----
EXTENDS Integers
VARIABLES wait_count, SelfCheck_state, Main_state, Land_state
vars == <<wait_count, SelfCheck_state, Main_state, Land_state>>
Init ==
  /\ wait_count = 0
  /\ SelfCheck_state = "SystemNormal"
  /\ Main_state = "BSCUOff"
  /\ Land_state = "InAir"

TurnBSCUOff ==
  \/
    /\ Main_state = "BSCUOn"
    /\ Main_state' = "BSCUOff"
    /\ UNCHANGED <<wait_count, SelfCheck_state, Land_state>>
```

```
----- MODULE FlightCrew -----
EXTENDS Integers
VARIABLES FlightCrew_state, BSCU_SelfCheck_SystemAbnormal
vars == <<FlightCrew_state, BSCU_SelfCheck_SystemAbnormal>>
Init ==
  /\ FlightCrew_state = "INIT"
  /\ BSCU_SelfCheck_SystemAbnormal = FALSE
TurnBSCUOff ==
  \/
    /\ FlightCrew_state = "in_DisarmAutoBrake_out_TurnBSCUOff"
    /\ FlightCrew_state' = "in_TurnBSCUOff_out_SwitchToManual"
    /\ UNCHANGED <<BSCU_SelfCheck_SystemAbnormal>>
```

```
TouchdownInCorrectMode ==
  /\ Land_state = "Landed" =>
    \/
      /\ SelfCheck_state = "SystemAbnormal"
      /\ Main_state = "ManualMode"
    \/
      /\ SelfCheck_state = "SystemNormal"
      /\ Main_state = "AutoBrakeOn"
```

Fortis

```
{
  "goodTrace": [
    "TurnBSCUOn",
    "ArmAutoBrake",
    "Wait"
  ],
  "badTrace": [
    "TurnBSCUOn",
    "ArmAutoBrake",
    "Touchdown",
    "DisarmAutoBrake"
  ],
  "violatingComponents": [
    "BSCU"
  ],
  "violatedInvs": []
}
```

```
{
  "goodTrace": [
    "TurnBSCUOn"
  ],
  "badTrace": [
    "Touchdown"
  ],
  "violatingComponents": [
    "BSCU"
  ],
  "violatedInvs": []
}
```

Fortis: Weakest Assumptions & Robustness

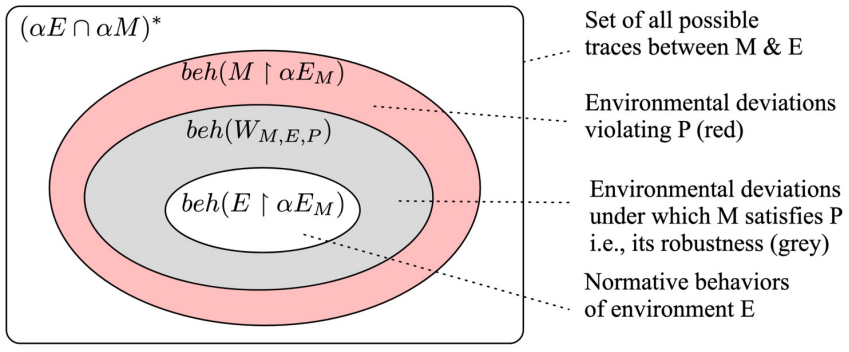


Figure 2: Behavioral relationships between possible environments.

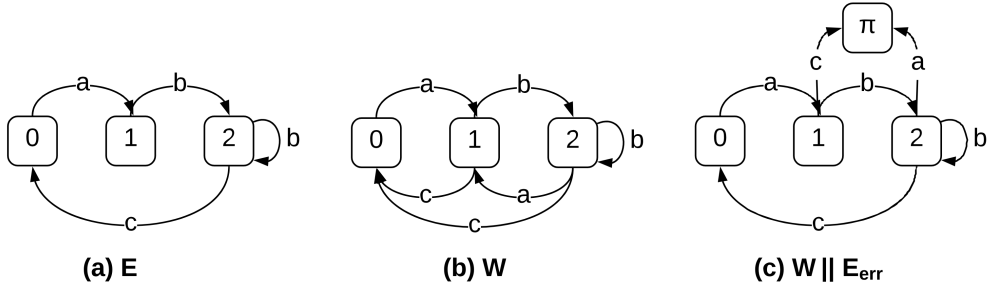


Figure 4: LTS's for a simple example illustrating the construction of robustness.

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.

Trace Classification

We originally planned on creating *deviation models* – essentially a model of the environment models extended with system-specific failures – which the existing implementation of Fortis uses to label the equivalence classes it groups unsafe behaviors into.

- Deviation models must be generated before the analysis begins.

Eunsuk suggested a different approach: classifying unsafe behaviors after their generation.

- Fortis was modified to output both unsafe error behavior traces (i.e., sequences of actions) and very similar safe behavior traces.
  - Example: *Safe:* "Arm BSCU, Wait, Touchdown" *Unsafe:* "Wait, Wait, Touchdown"

Trace Classification: STPA & Time

The Wait Action

Four of the seven STPA Guidewords involve time, but the model of time can be thought of as being between ordering and clocked time.

- The three non-time guidewords: Provided, Not Provided, Out of Order
- The four time-related guidewords: Too Early, Too Late, Stopped too Soon, Applied too Long

STPA uses “Goldilocks” time: there is too little, too much, or the right amount.

We have a special action, called `Wait`, which can be used to represent the passing of time.

Trace Classification: Edit Distance

Key Insight: Comparing safe and unsafe behavior traces is similar to comparing correctly-spelled and misspelled words.

- Example: much as "`aat`" is one letter away from "`cat`," "`Wait`, `Wait`, `Touchdown`" is one action away from "`Arm BSCU`, `Wait`, `Touchdown`"

This concept is sometimes called *edit distance*: the number of atomic edits required to change some string *p* into some string *s*.

- Example: `ba` → `acb` = 2
  - `ba` → `ab` (Transposition)
  - `ab` → `acb` (Addition)

But! We don't need the edit distance, we need information about the edits themselves, e.g., for `ba` → `acb` we don't want "2", we want "Transposition (`ba`, prefix = `ϵ`), Addition (`c`, prefix = `a`)"

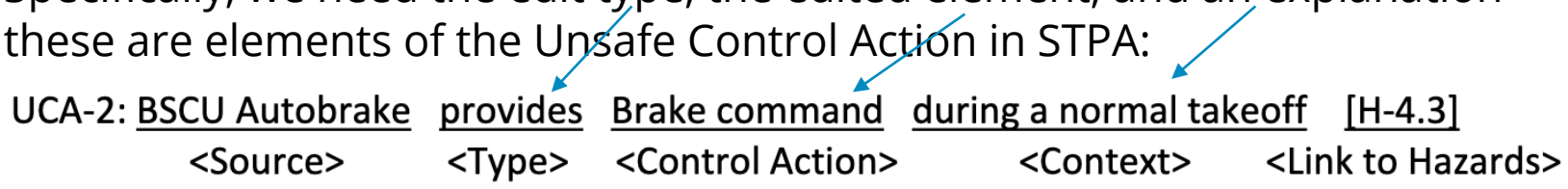
- Or at least the first edit (the utility of subsequent edits is unclear)

Trace Classification: Damerau-Levenshtein Algorithm

The Damerau-Levenshtein algorithm is a well-known and powerful algorithm for calculating edit distance. It is a dynamic programming algorithm (i.e., it builds the output progressively on subsets of the input), and it supports four types of atomic edits: addition, deletion, substitution, and transposition.

We can use the Damerau-Levenshtein algorithm to automate the classification of STPA guidewords if we can build:

1. Modifications to the algorithm to track which edits are used
   - Specifically, we need the edit type, the edited element, and an explanation – these are elements of the Unsafe Control Action in STPA:



2. A mapping from atomic string edit types to STPA Guidewords

Trace Classification: 1. Modifications to Track Edits

ALGORITHM 2: Computation of the unrestricted Damerau-Levenshtein distance between strings p and s

```
var C: array [0..|p|,0..|s|] of Integer
var CP: array [1..|Σ|] of Integer
var i', j', CS: Integer

for i := 0 to |p| do C[i, 0] := i
for j := 0 to |s| do C[0, j] := j

for i := 1 to |Σ| do CP[i] := 0

for i := 1 to |p| do begin
    CS := 0
    for j := 1 to |s| do begin
        if p[i] = s[j] then d := 0 else d:= 1
        C[i, j] := min(C[i − 1, j] + 1, C[i, j − 1] + 1, C[i − 1, j − 1] + d)

Comment CP[c] stores the largest index i' < i such that p[i'] = c.
        CS stores the largest index j' < j such that s[j'] = p[i].

        i' := CP[s[j]]
        j' := CS
        if i' > 0 and j' > 0 then begin
            C[i, j] := min(C[i, j], C[i' − 1, j' − 1] + (i − i') + (j − j') − 1)
        end
        if p[i] = s[j] then CS := j
    end
    CP[p[i]] := i
end
output C[|p|, |s|]
```

Leonid Boytsov. 2011. Indexing methods for approximate dictionary searching: Comparative analysis. ACM J. Exp. Algorithmics 16, Article 1.1 (2011), 91 pages. https://doi.org/10.1145/1963190.1963191

`C` is a 2D array of edit distances that is built iteratively using substrings of inputs *p* and *s*.

`C[i][j]` is the minimum edit distance to convert the `i`-length initial substring of *p* into the `j`-length initial substring of *s*.

We introduce a new 2D array `CG` (C-Guidewords) which collects information about the edits.

- Each element is an Unsafe Control Action corresponding to the initial edit

```
public record UnsafeControlAction(
        String source,
        Guideword guideword,
        String controlAction,
        String context,
        String violatedConstraint) {
};
```

Trace Classification: 2. Mapping Atomic String Edits

We define `Any` to mean any action that is a) not `Wait` and b) not correct

Edit	Removed	Guideword	Control Action
Deletion	Wait	Too Early	Early Action
Deletion	Any	Not Providing	Removed Action

Edit	Added	Guideword	Control Action
Addition	Wait	Too Late	Late Action
Addition	Any	Providing	Additional Action

Here `Any`'s meaning is similar, but not identical: it means any action that is a) not `Wait` and b) not incorrect if used in the **Correct** column, not correct if used in the **Incorrect** column

Edit	Correct	Incorrect	Guideword	Control Action
Substitution	Any	Any	Providing	Provided Action
Substitution	Any	Wait	Not Providing	Removed Action
Substitution	Wait	Any	Providing	Additional Action
Transposition	Any	Any	Out of Order	Incorrect Action
Transposition	Any	Wait	Too Late	Late Action
Transposition	Wait	Any	Too Early	Early Action

What about *Stopped too Soon* and *Applied too Long*?

There is no clear edit distance correlation for these guidewords.

Instead, we require the user specifies *Activities*, which have three elements:

1. A start action
2. A stop action
3. A name

Example: A hypothetical `Brake` activity is started by `ApplyBrakes` and stopped by `ReleaseBrakes`.

Trace Classification: Computing Activity Duration

With the activity specifications, classifying the final two guidewords is straightforward: compare the number of wait actions between the start and end actions of each activity in the safe and unsafe traces.

Activity Duration in Unsafe Trace	Guideword
Shorter	Stopped too Soon
Longer	Applied too Long

FASR Output

Goal:

	Provided	Not Provided
Arm BSCU	*Not unsafe for this hazard*	Crew turns on the BSCU but does not arm it before landing
Disarm BSCU	Crew turns on and arms the BSCU but then (inadvertently or mistakenly) disarms it	*Not unsafe for this hazard*

Output:

#	△ Name	Provided	Not Provided
1	S ArmAutoBrake		After "TurnBSCUOn" the environment did not perform the expected "ArmAutoBrake" action; it instead performed "Wait". It subsequently performed a "Touchdown" action.
2	S DisarmAutoBrake	After "TurnBSCUOn" -> "ArmAutoBrake" -> "Wait" the environment performed an unexpected "DisarmAutoBrake" action. It subsequently performed a "Touchdown" action.	

Performance

	Time (Sec)		Generated UCAs		Fortis	
	Cold	Warm	Fortis	Filtered	States	Transitions
BSCU, No Abnormal Behavior	4.7	4.1	31	14	241	3,100
BSCU, with Abnormal Behavior	7.5	5.8	55	23	1,249	50,024

Agenda

Background: Three Technologies

Case Study: Braking System Control Unit (BSCU)

FASR Tool

Exploratory Study

- Study Description
- Results

Future Work

User Study: Description

Research Questions:

1. Do study participants perceive any benefits from using FASR to identify UCAs?
2. What downsides and improvements to the tool do study participants identify?

Demographics:

- 10 invited, 9 completed
- 6 given tool access, 3 manual
- Most familiar with MBE, range of FM and STPA expertise

Process:

1. (Whole Cohort, 60 minutes) Training
2. (1-on-1, 60 minutes) Study
3. (1-on-1, 30 minutes) Interview

User Study: Results

RQ1: Do study participants perceive any benefits from using FASR to identify UCAs?

- Every participant answered in the affirmative
- Some had more caveats than others:
  - "This is absolutely the sort of tool I would use [if I were still working in this area]. I really like what I'm seeing here, [the tool] generated more information than the tools we used to have that were less connected to formalisms."
  - "Yeah, [although] this iteration, maybe not yet… This would be super helpful if you could clean it up a bit."
- Particularly helped with low-STPA-knowledge users

RQ2: What downsides and improvements to the tool do study participants identify?

- A lot (maybe too much) output
- Input language could be changed or expanded
  - Additional SysML diagram types
  - SysMLv2
  - STPA Step 4 Support

Agenda

Background: Three Technologies

Case Study: Braking System Control Unit (BSCU)

FASR Tool

Exploratory Study

Future Work

Future Work

- Evaluation including model creation time / expertise
- Better filtering / clustering of Fortis output
- Broader process support