PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles

Hyungsub Kim, Muslum Ozgur Ozmen,
Z. Berkay Celik, Antonio Bianchi, and Dongyan Xu

Purdue University

IEEE Symposium on Security and Privacy (S&P) 2022

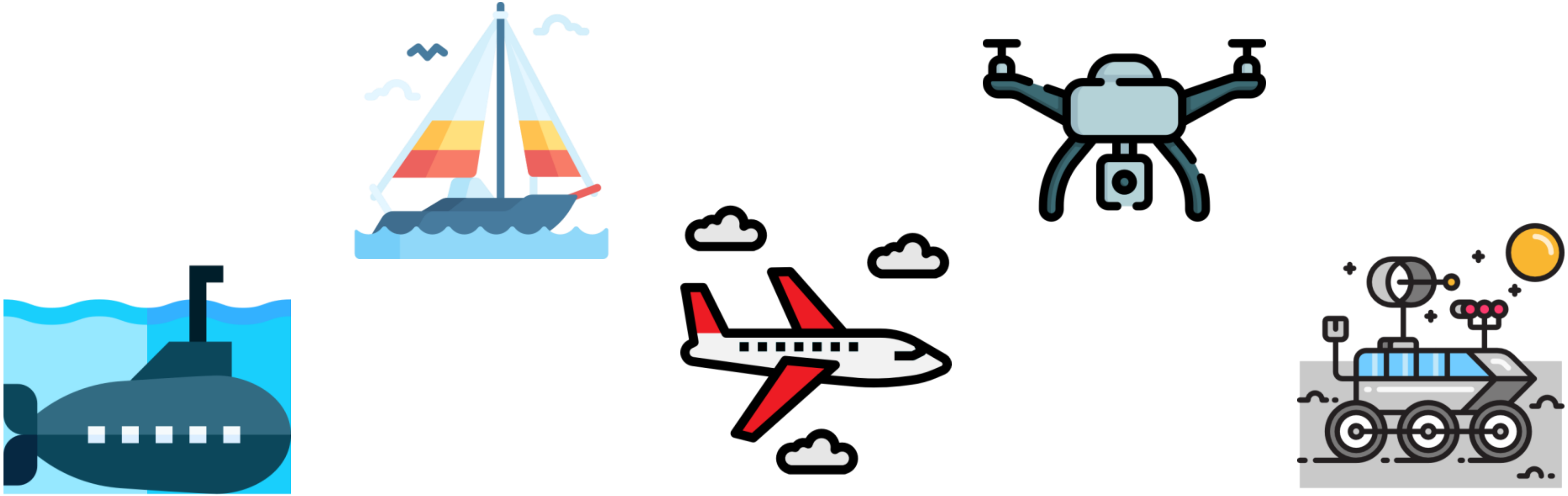
PURDUE UNIVERSITY

CERIAS

PurSecLab

Background

- What is the Robotic Vehicle (RV)?
  - Vehicles that move “autonomously” on the ground, in the air, on the sea, under the sea, or in space

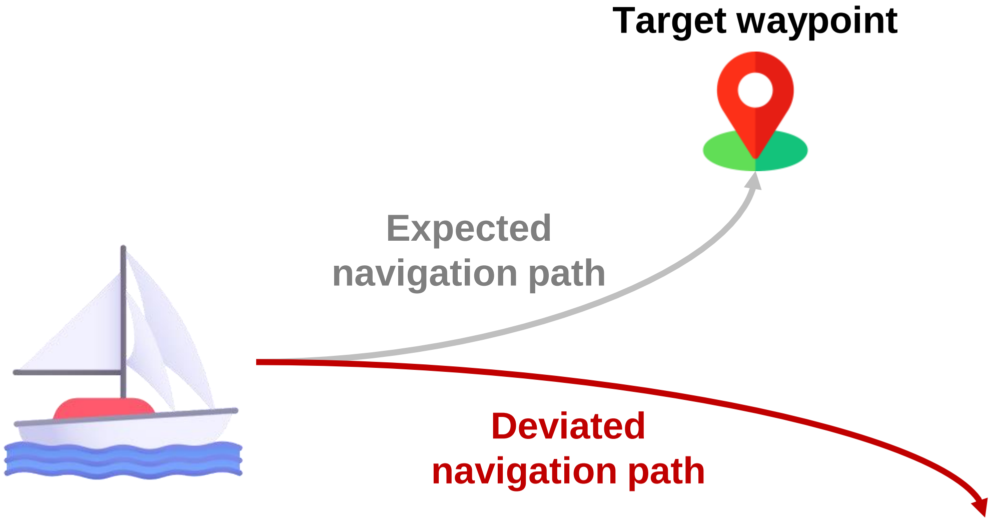


Background

- What is the logic bug?
  - Do not cause any program crash or memory corruption
  - Lead to undesired physical behavior

Background

- What is the logic bug?
  - Do not cause any program crash or memory corruption
  - Lead to undesired physical behavior



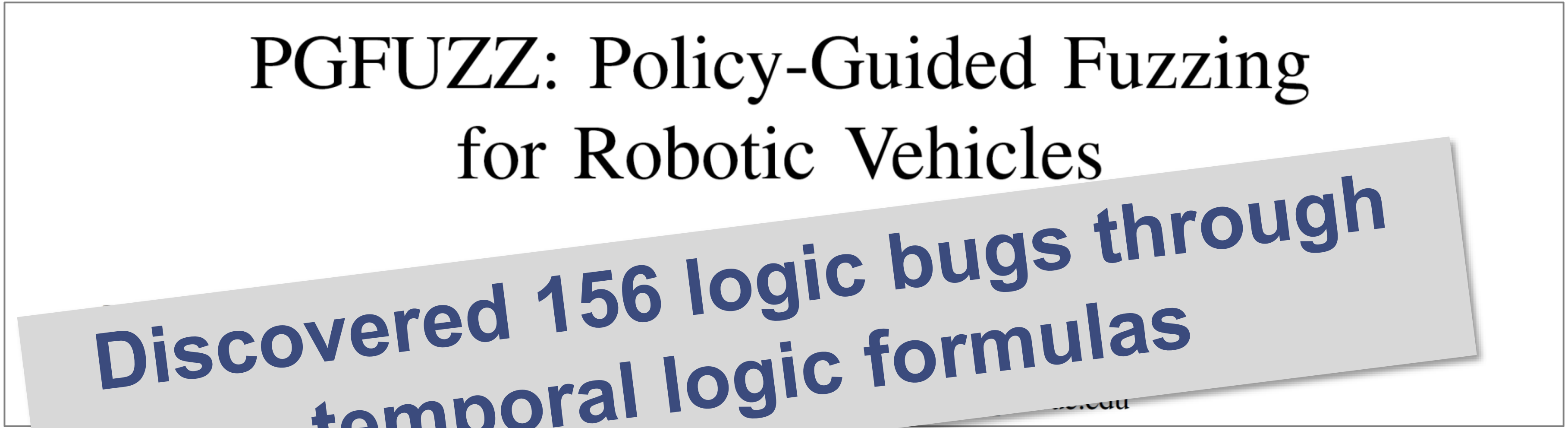
Background

- Why are logic bugs important in Robotic Vehicles (RV)?

- A preliminary survey about 1,257 bugs in RV software:
  - Most bugs in RV software are logic bugs
    - Logic bugs: 98.2%
    - Memory corruption bugs: 1.8%

  - 97.3% logic bugs lead to physical damage
    - Crashing on the ground
    - Unstable attitude/position

Photo credit: https://gbagenlaw.com/drone-related-injuries-are-becoming-more-common



PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles

Discovered 156 logic bugs through temporal logic formulas

Safety policies in the form of linear temporal logic (LTL)

$$\square\{(\mathtt{ALT_t} < \mathtt{RTL\_ALT}) \wedge (\mathtt{Mode_t} = \mathtt{RTL}) \rightarrow (\mathtt{ALT_{t-1}} < \mathtt{ALT_t})\}$$

$$\square\{(\mathtt{GPS_{fail}} = \mathtt{on}) \wedge (RC_t = off) \rightarrow (\mathtt{Mode_t} = \mathtt{LAND})\}$$

$$\square\{(\mathtt{Mode_t} = \mathtt{FLIP_1}) \rightarrow (\Diamond_{[0,2.5]}\mathtt{Mode_t} = \mathtt{FLIP_3})\}$$

...

Our previous work - PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles, NDSS 2021

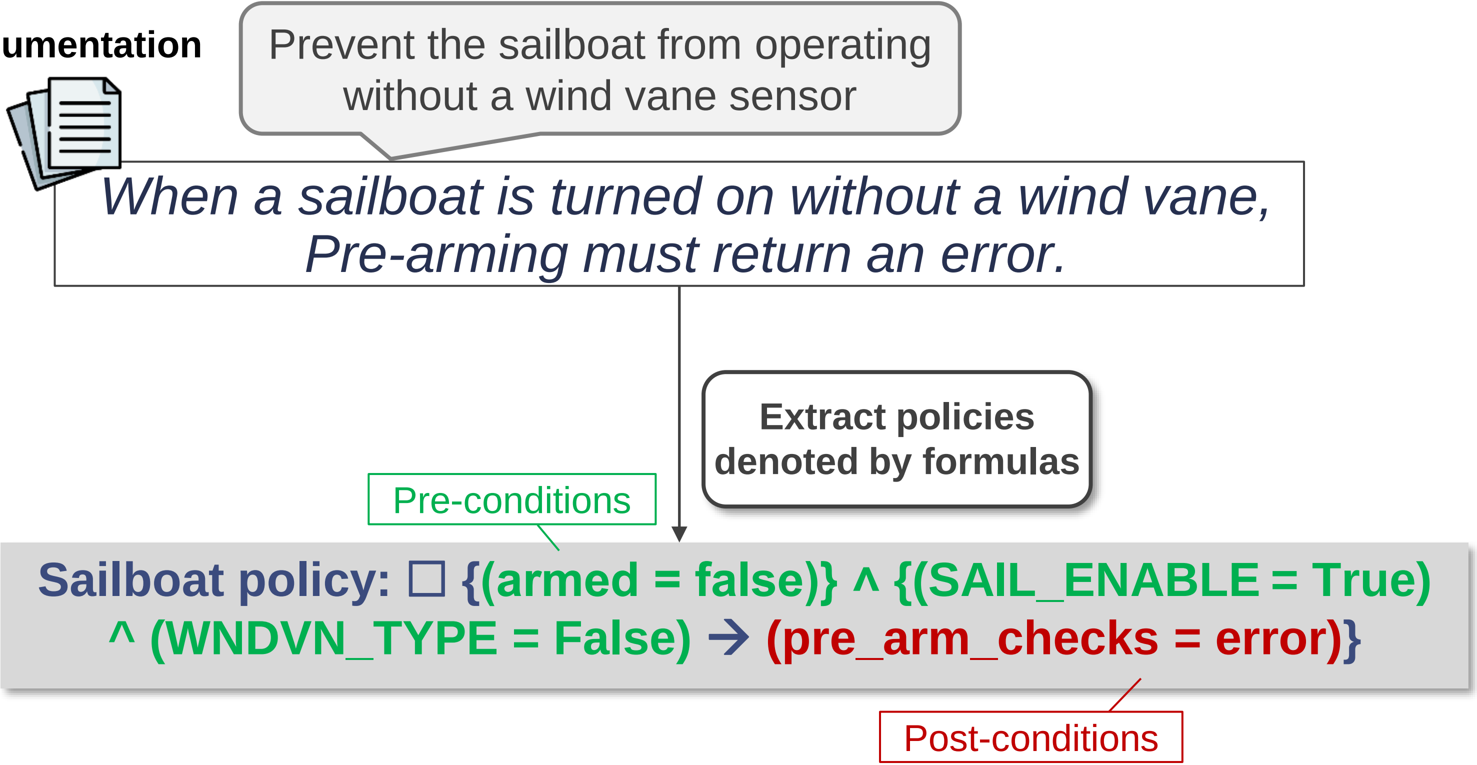
Motivation

Documentation

When a sailboat is turned on without a wind vane, Pre-arming must return an error.

Prevent the sailboat from operating without a wind vane sensor

Motivation



Documentation

Prevent the sailboat from operating without a wind vane sensor

When a sailboat is turned on without a wind vane, Pre-arming must return an error.

Extract policies denoted by formulas

Pre-conditions

Sailboat policy: □ {(armed = false)} ^ {(SAIL_ENABLE = True) ^ (WNDVN_TYPE = False) → (pre_arm_checks = error)}

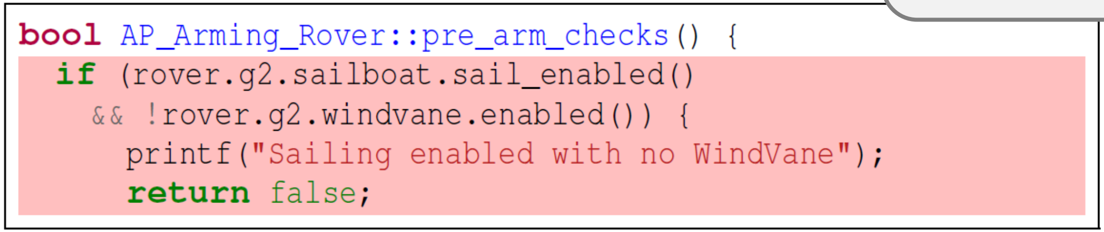
Post-conditions

Motivation

Pre-conditions

Sailboat policy: □ {(armed = false)} ^ {(SAIL_ENABLE = True) ^ (WNDVN_TYPE = False) → (pre_arm_checks = error)}

Post-conditions



```
bool AP_Arming_Rover::pre_arm_checks() {
  if (rover.g2.sailboat.sail_enabled()
    && !rover.g2.windvane.enabled()) {
      printf("Sailing enabled with no WindVane");
      return false;
```

The RV software initially did not implement this policy, causing potential safety violations

Can we automatically fix these logic bugs?

Motivation

- Limitations of prior program repair tools(1)

PROGRAM-REPAIR.ORG Home Bibliography Too

C/C

the code

on symbolic analysis

— system for feedback generation

DeepFix — tool for fi

fix error handling bugs for C programs

automated program repair tool based on genetic programmi

Kali — generate-and-validate patch

pair hints to assist the programmer

Limitation 1: Mainly focus on fixing memory corruptions

Limitation 2: Need a complete set of test cases

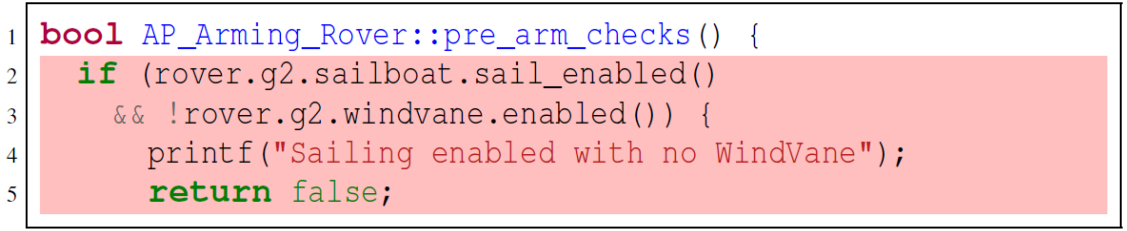
Limitation 3: Poor support for floating-point operations

(1) https://program-repair.org/tools.html

Motivation

Can we reuse formulas to fix the found bugs?

Sailboat policy: □ {(armed = false)} ^ {(SAIL_ENABLE = True) ^ (WNDVN_TYPE = False) → (pre_arm_checks = error)}



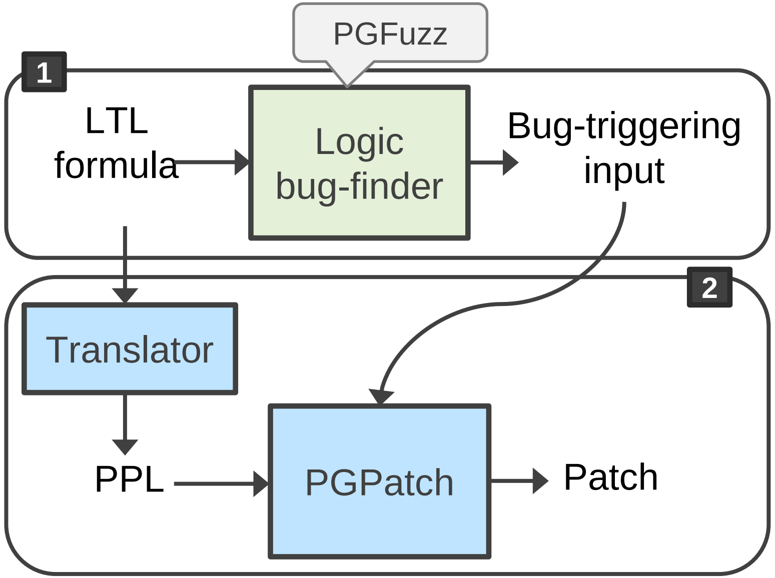
```
bool AP_Arming_Rover::pre_arm_checks() {
  if (rover.g2.sailboat.sail_enabled()
    && !rover.g2.windvane.enabled()) {
      printf("Sailing enabled with no WindVane");
      return false;
```

Overview of PGPatch

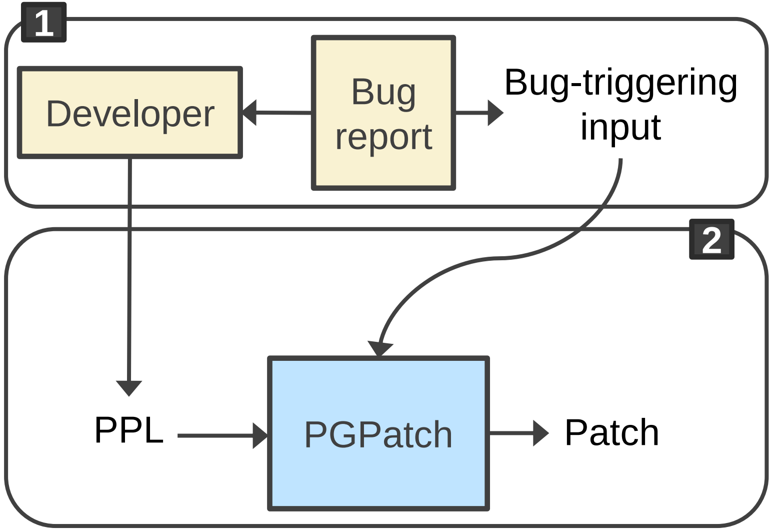
- PGPatch
    - New Automatic Program Repair tool to fix logic bugs from temporal logic formulas

Overview of PGPatch

- Two usage scenarios:
  1. Using existing LTL formulas
  2. Using developer-written formulas in PPL (PGPatch Language)



1. First usage scenario of PGPatch



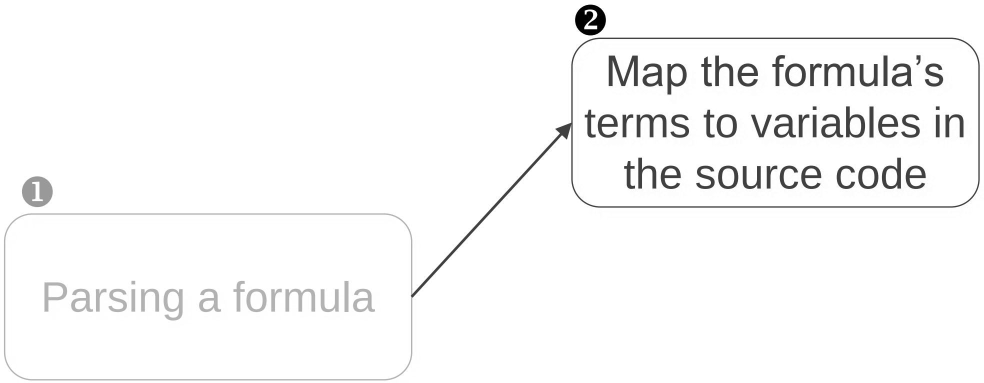
2. Second usage scenario of PGPatch

Overview of PGPatch

❶

Parsing a formula

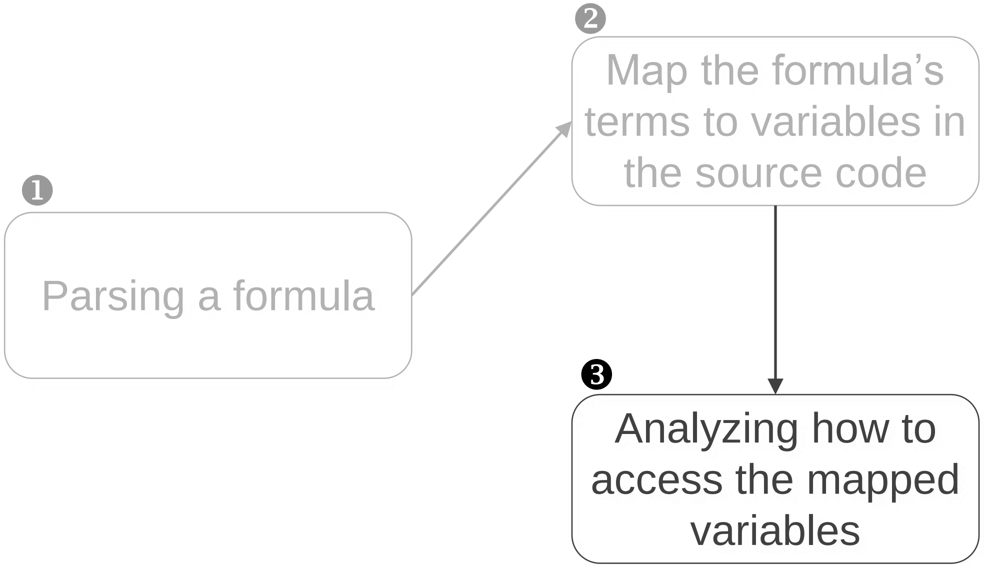
Overview of PGPatch



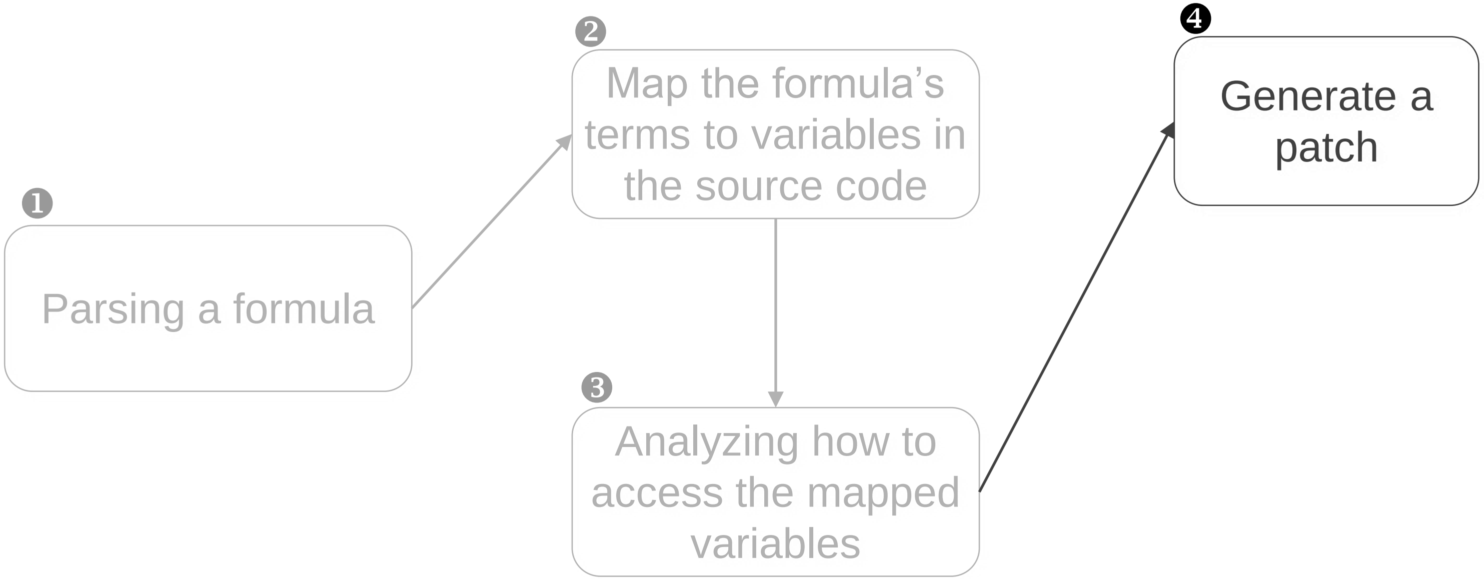
❶ Parsing a formula

❷ Map the formula’s terms to variables in the source code

Overview of PGPatch

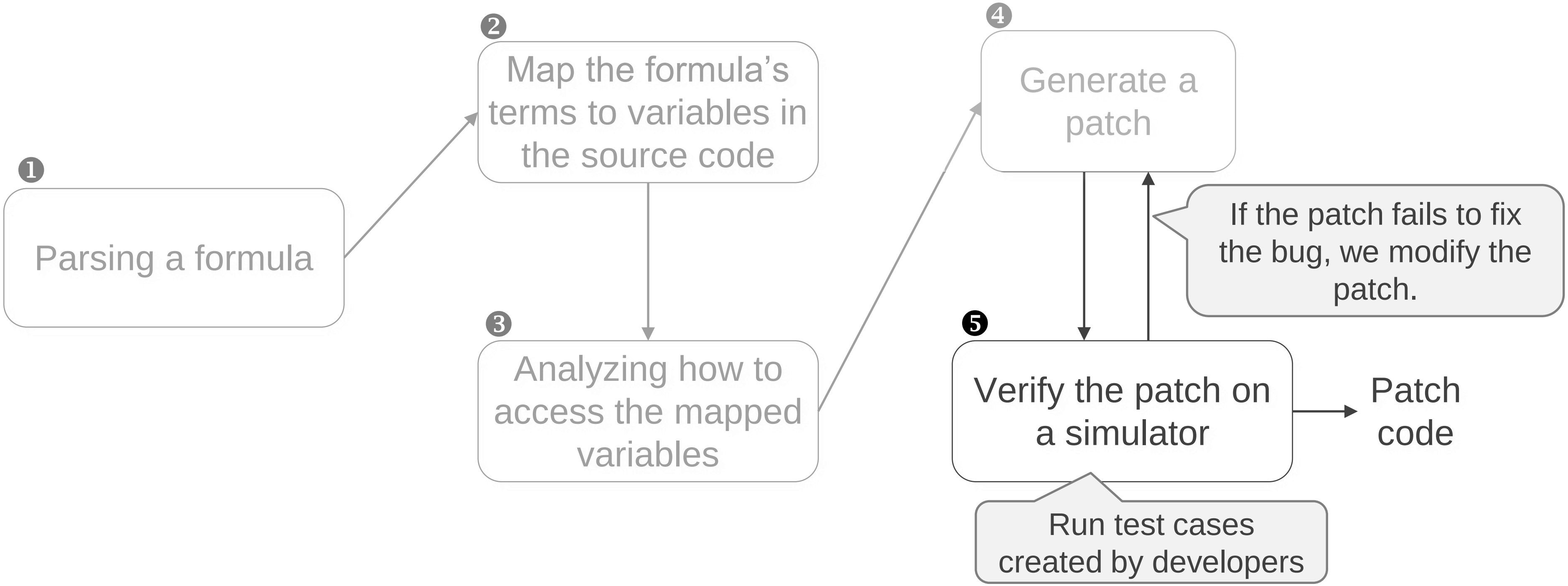

❶
Parsing a formula
❷
Map the formula's terms to variables in the source code
❸
Analyzing how to access the mapped variables

Overview of PGPatch



❶
Parsing a formula
❷
Map the formula's terms to variables in the source code
❸
Analyzing how to access the mapped variables
❹
Generate a patch

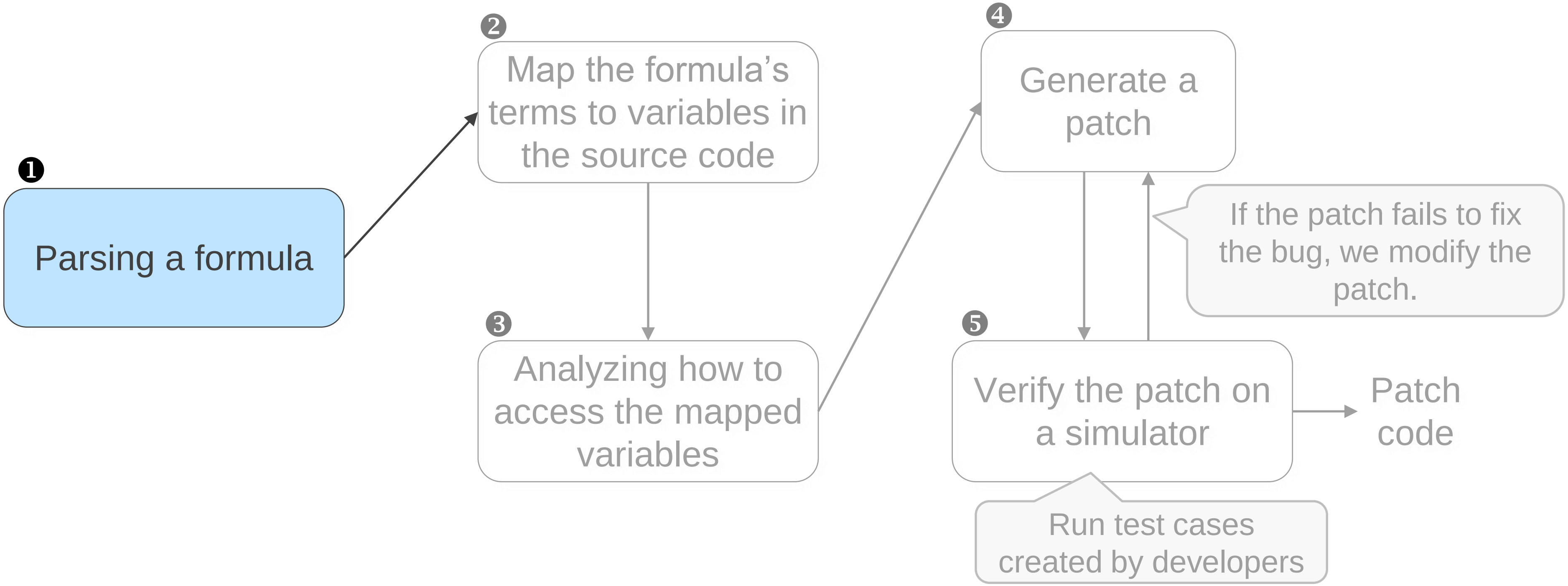
Overview of PGPatch


❶
Parsing a formula
❷
Map the formula’s terms to variables in the source code
❸
Analyzing how to access the mapped variables
❹
Generate a patch
❺
Verify the patch on a simulator
Patch code

If the patch fails to fix the bug, we modify the patch.

Run test cases created by developers

Overview of PGPatch



❶ Parsing a formula

❷ Map the formula’s terms to variables in the source code

❸ Analyzing how to access the mapped variables

❹ Generate a patch

If the patch fails to fix the bug, we modify the patch.

❺ Verify the patch on a simulator

Run test cases created by developers

Patch code

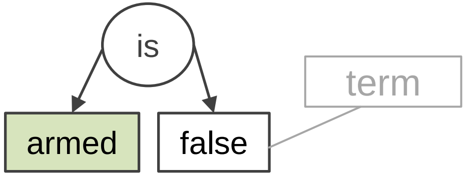
❶ Parsing Formulas

- Convert a formula to an expression tree

Proposition

Sailboat policy in PPL syntax: If *armed is false* and *SAIL_ENABLE* is *1* and *WNDVN_TYPE* is *0*, then *pre_arm_checks* is *error*

1) Convert the formula to a tree

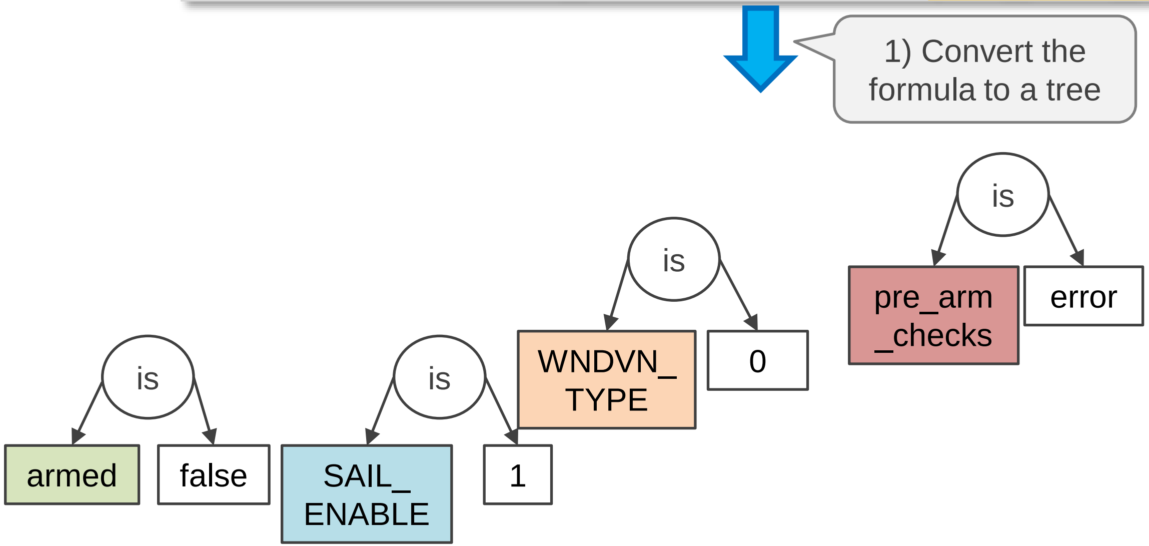


❶ Parsing Formulas

- Convert a formula to an expression tree

Proposition

Sailboat policy in PPL syntax: If *armed* is *false* and *SAIL_ENABLE* is *1* and *WNDVN_TYPE* is *0*, then *pre_arm_checks* is *error*

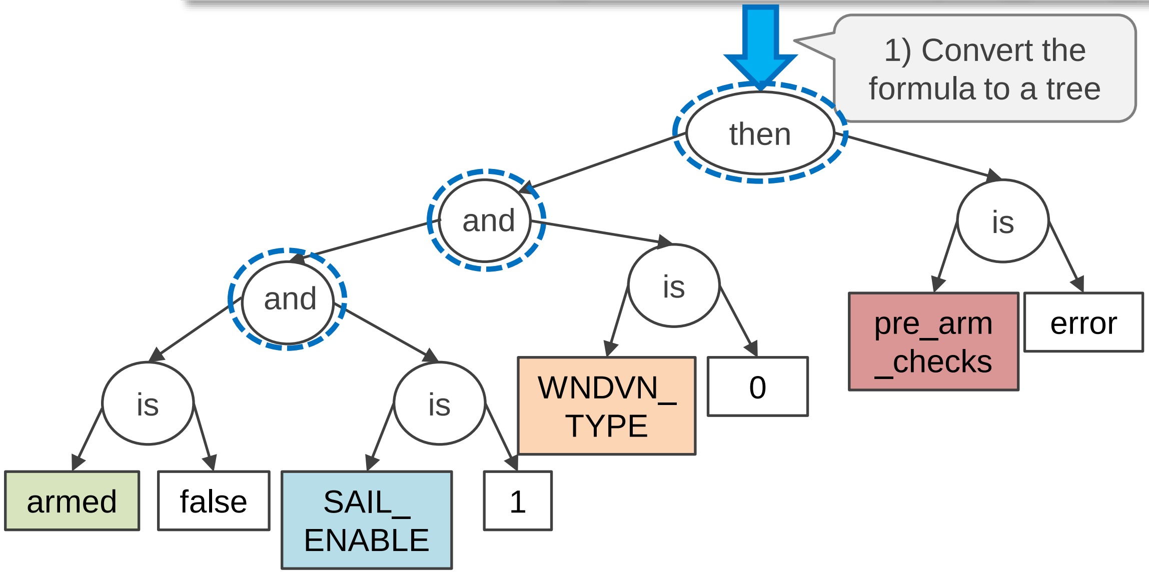


1) Convert the formula to a tree

❶ Parsing Formulas

- Convert a formula to an expression tree

Sailboat policy in PPL syntax: If *armed* is *false* and *SAIL_ENABLE* is *1* and *WNDVN_TYPE* is *0*, then *pre_arm_checks* is *error*

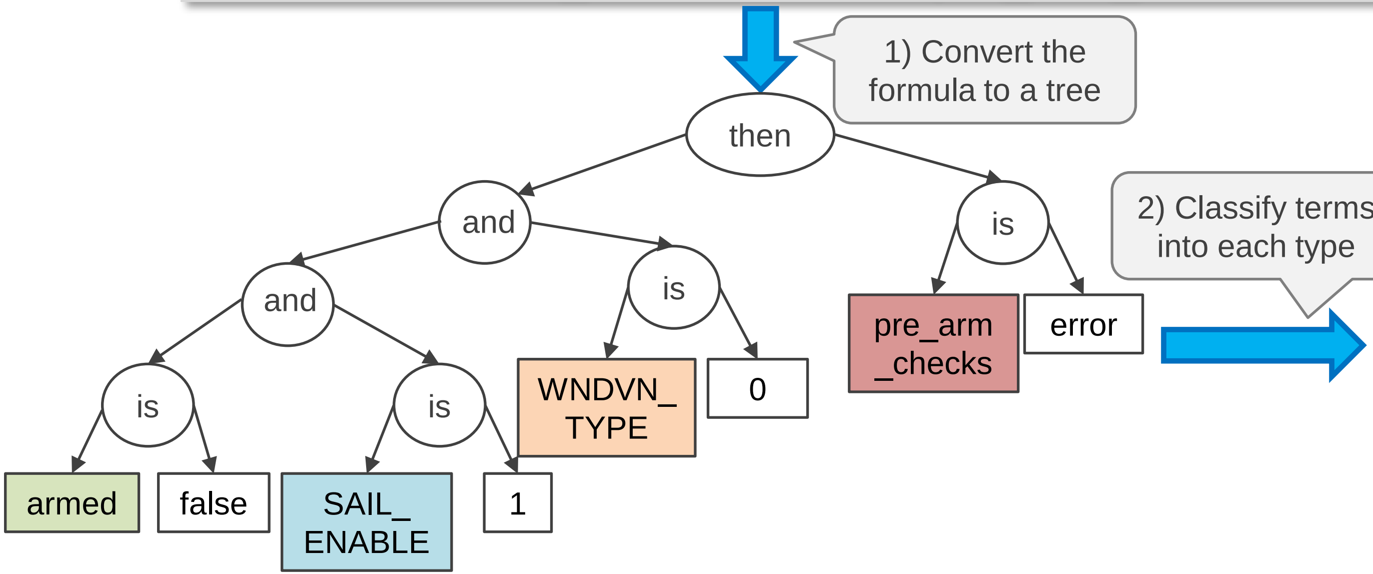


State (P): an RV's physical state, **State (C)**: an RV's configuration state

❶ Parsing Formulas

- Convert a formula to an expression tree

Sailboat policy in PPL syntax: If *armed* is *false* and *SAIL_ENABLE* is *1* and *WNDVN_TYPE* is *0*, then *pre_arm_checks* is *error*



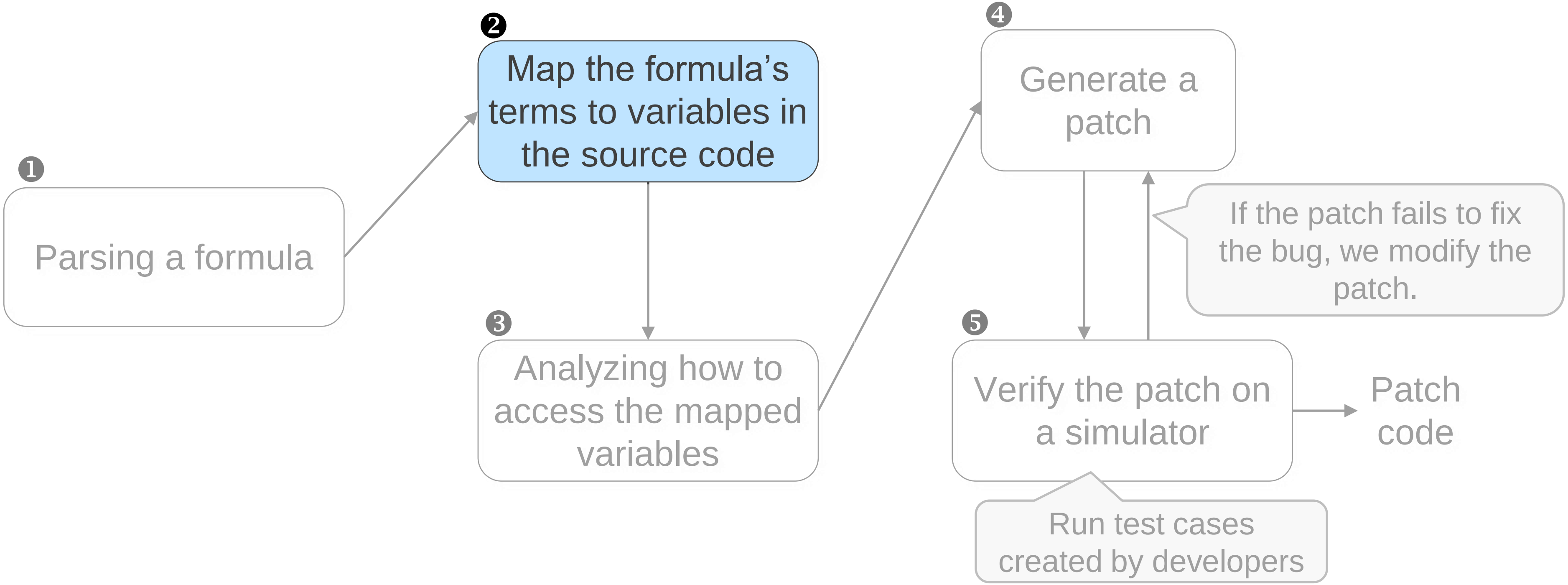
1) Convert the formula to a tree

2) Classify terms into each type

Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

State (P): an RV's physical state, **State (C)**: an RV's configuration state

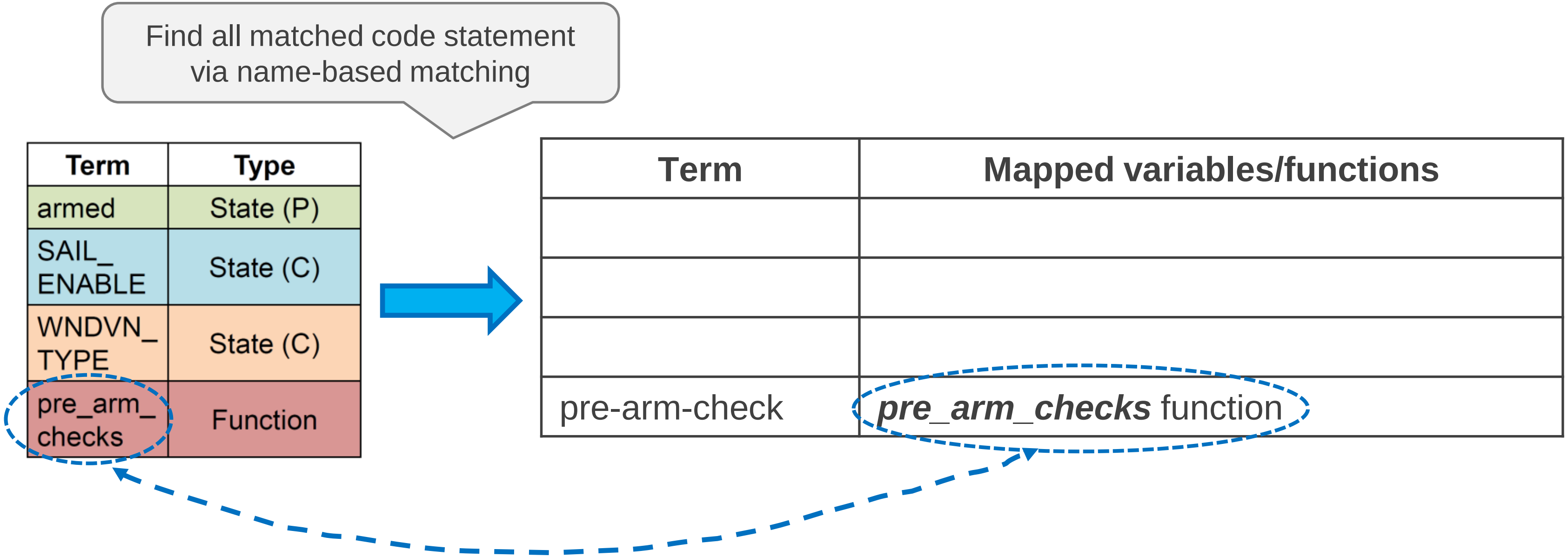
Overview of PGPatch



❶
Parsing a formula
❷
Map the formula’s terms to variables in the source code
❸
Analyzing how to access the mapped variables
❹
Generate a patch
If the patch fails to fix the bug, we modify the patch.
❺
Verify the patch on a simulator
Patch code
Run test cases created by developers

❷ Terms and Source Code Mapping

- How to match each term with the corresponding variables/functions in the source code?



Find all matched code statement via name-based matching

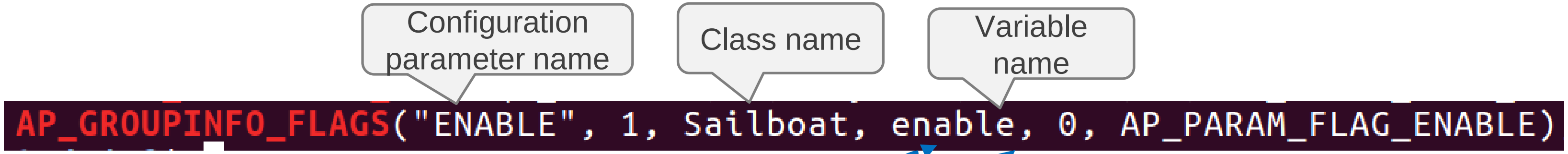
Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

Term	Mapped variables/functions
pre-arm-check	***pre_arm_checks*** function

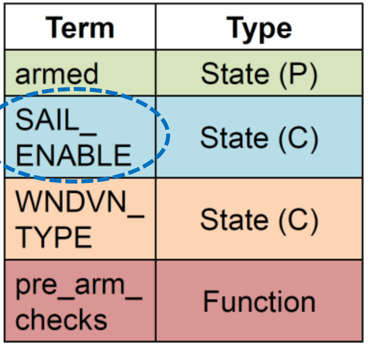
State (P): an RV’s physical state, **State (C)**: an RV’s configuration state

❷ Terms and Source Code Mapping

- Configuration parameters
  - Take advantage of heuristic (how the RV software port the configuration parameters from XML files to source code)



```
AP_GROUPINFO_FLAGS("ENABLE", 1, Sailboat, enable, 0, AP_PARAM_FLAG_ENABLE)
```



Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

Term	Mapped variables/functions
SAIL_ENABLE	Private ***enable*** in Sailboat class
WNDVN_TYPE	Private ***_direction_type*** in AP_WindVane class
pre-arm-check	***pre_arm_checks*** function

State (P): an RV's physical state, **State (C)**: an RV's configuration state

❷ Terms and Source Code Mapping

- Physical states
  - Take advantage of RV software’s strict coding conventions[1]
  - Each variable’s name denotes a physical state (e.g., altitude, height)

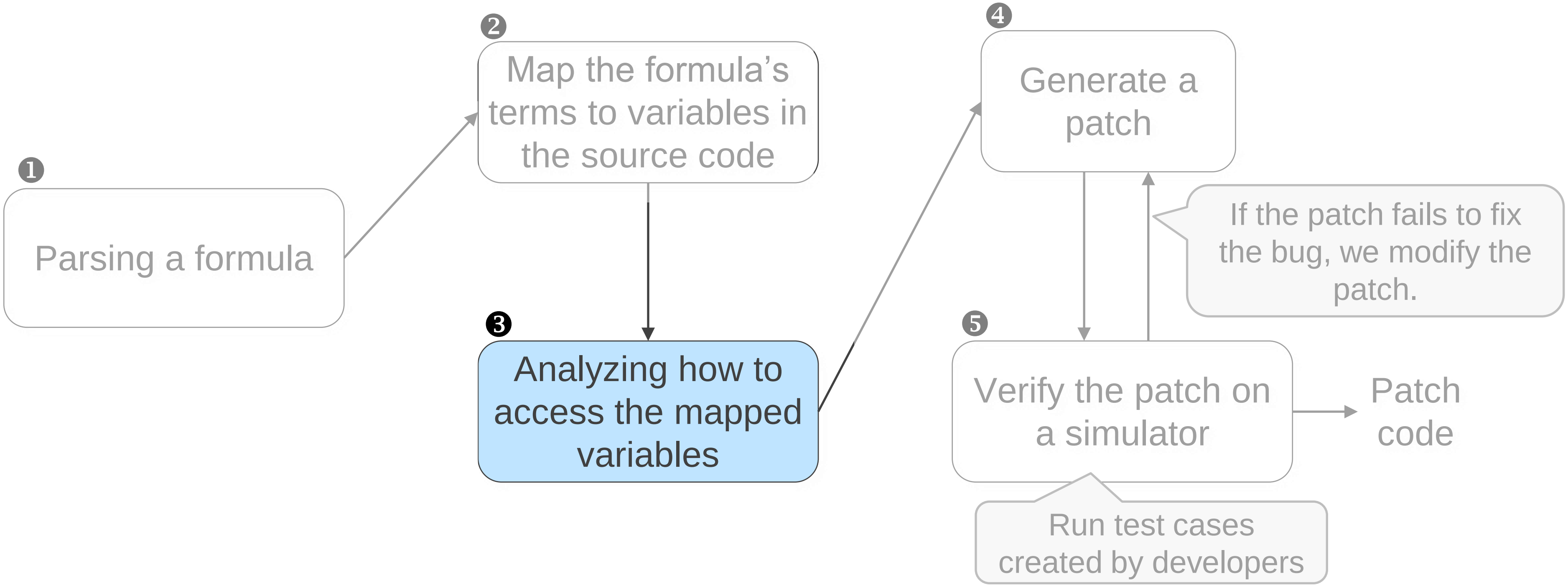
Find all matched code statement via name-based matching

Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

Term	Mapped variables/functions
armed	Public ***armed*** in Notify class
SAIL_ENABLE	Private ***enable*** in Sailboat class
WNDVN_TYPE	Private _***direction_type*** in AP_WindVane class
pre-arm-check	***pre_arm_checks*** function

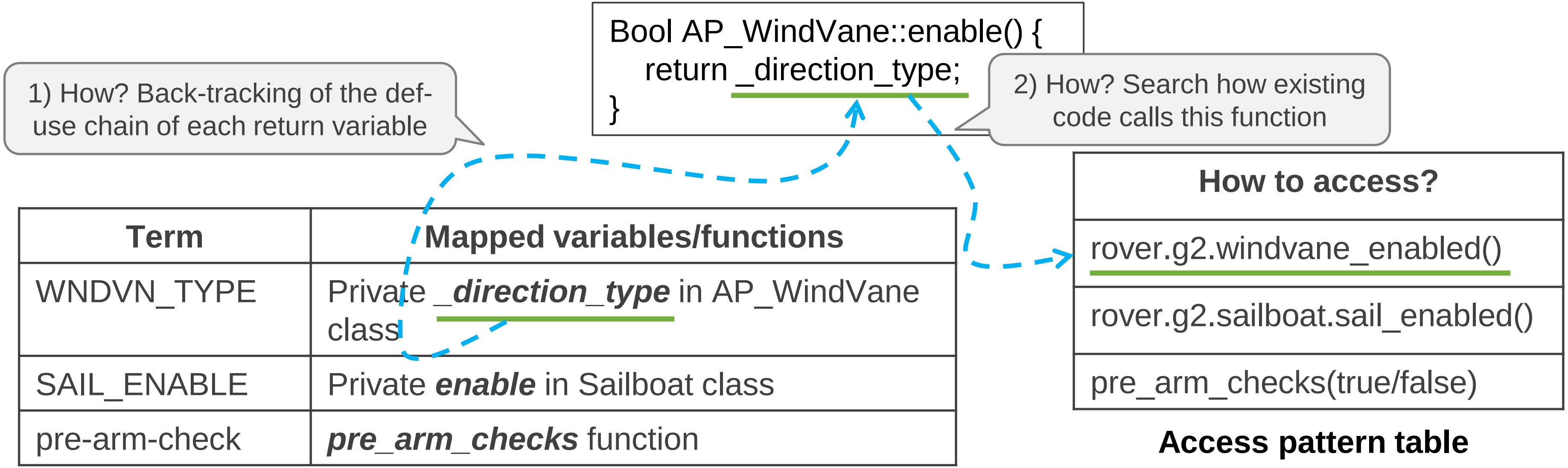
[1] https://ardupilot.org/dev/docs/style-guide.html

Overview of PGPatch


❶
Parsing a formula
❷
Map the formula’s terms to variables in the source code
❸
Analyzing how to access the mapped variables
❹
Generate a patch
If the patch fails to fix the bug, we modify the patch.
❺
Verify the patch on a simulator
Patch code
Run test cases created by developers

❸ Access Pattern Analyzer

- The patch can be placed in different locations than the mapped variables.
- How to access the private members (variables) from another class/function?
- Let’s find ‘*getter*’ functions that returns the variables.



1) How? Back-tracking of the def-use chain of each return variable

```
Bool AP_WindVane::enable() {
    return _direction_type;
}
```

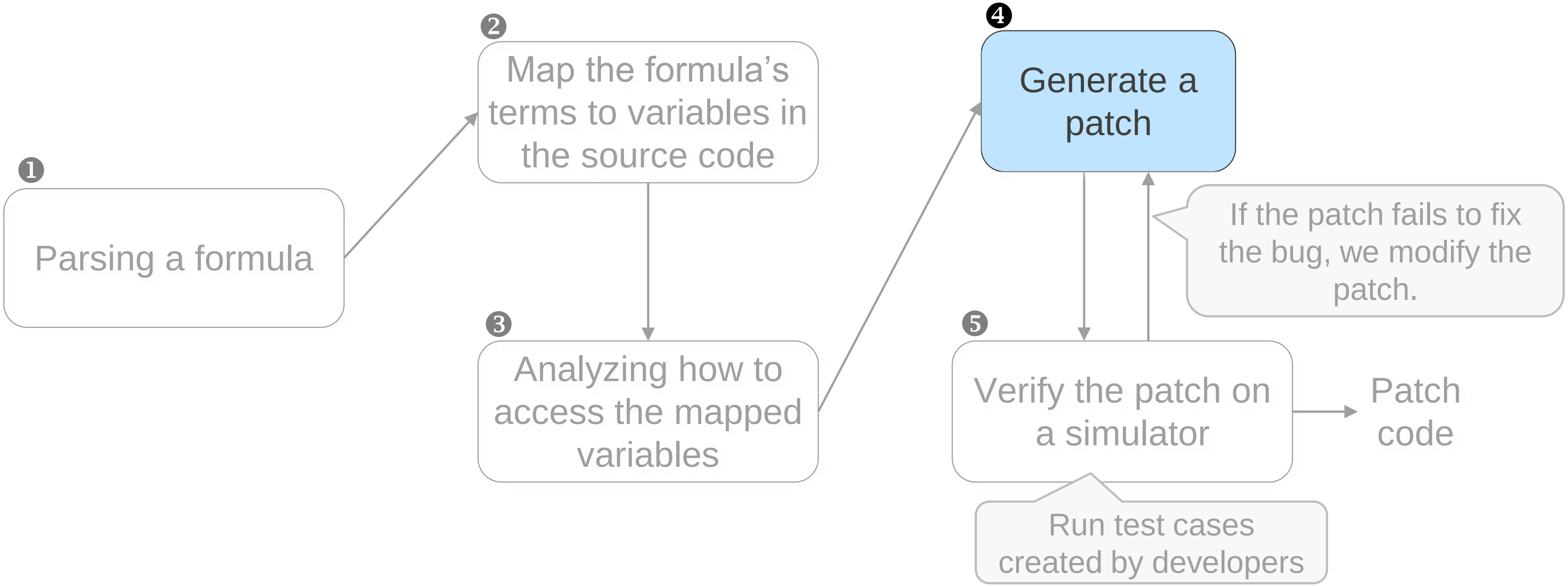
2) How? Search how existing code calls this function

Term	Mapped variables/functions
WNDVN_TYPE	Private ***_direction_type*** in AP_WindVane class
SAIL_ENABLE	Private ***enable*** in Sailboat class
pre-arm-check	***pre_arm_checks*** function

How to access?
rover.g2.windvane_enabled()
rover.g2.sailboat.sail_enabled()
pre_arm_checks(true/false)

Access pattern table

Overview of PGPatch



❶ Parsing a formula

❷ Map the formula’s terms to variables in the source code

❸ Analyzing how to access the mapped variables

❹ Generate a patch

If the patch fails to fix the bug, we modify the patch.

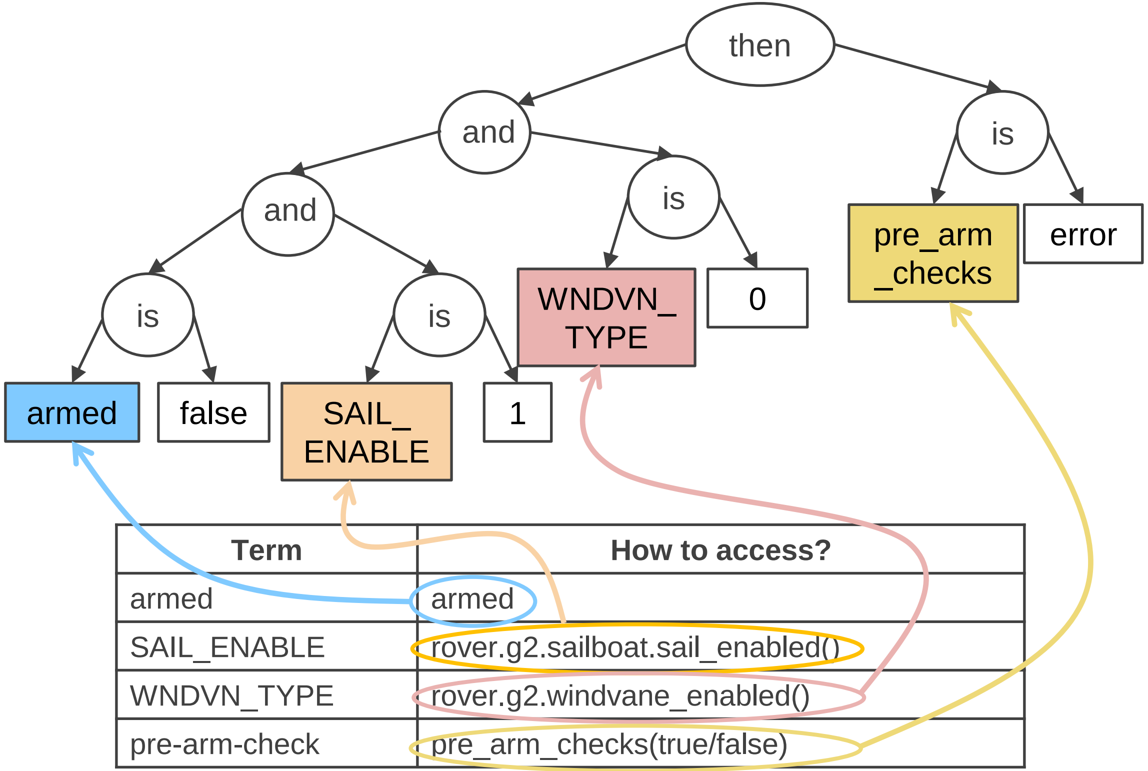
❺ Verify the patch on a simulator

Patch code

Run test cases created by developers

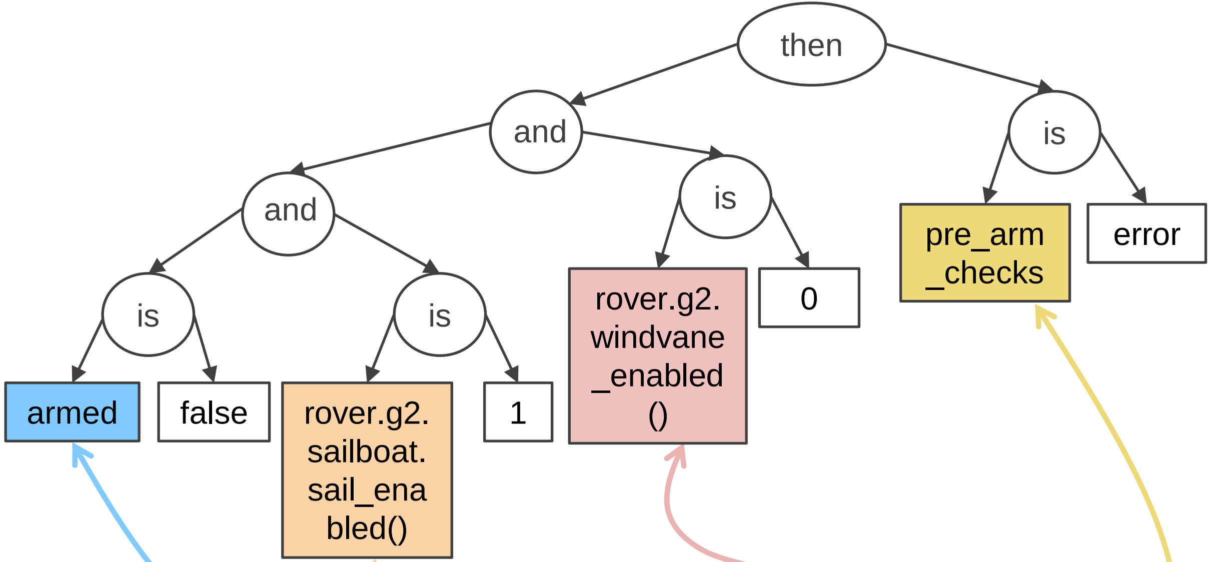
❹ Patch Generation

- Switch terminal nodes of the tree with the found access patterns



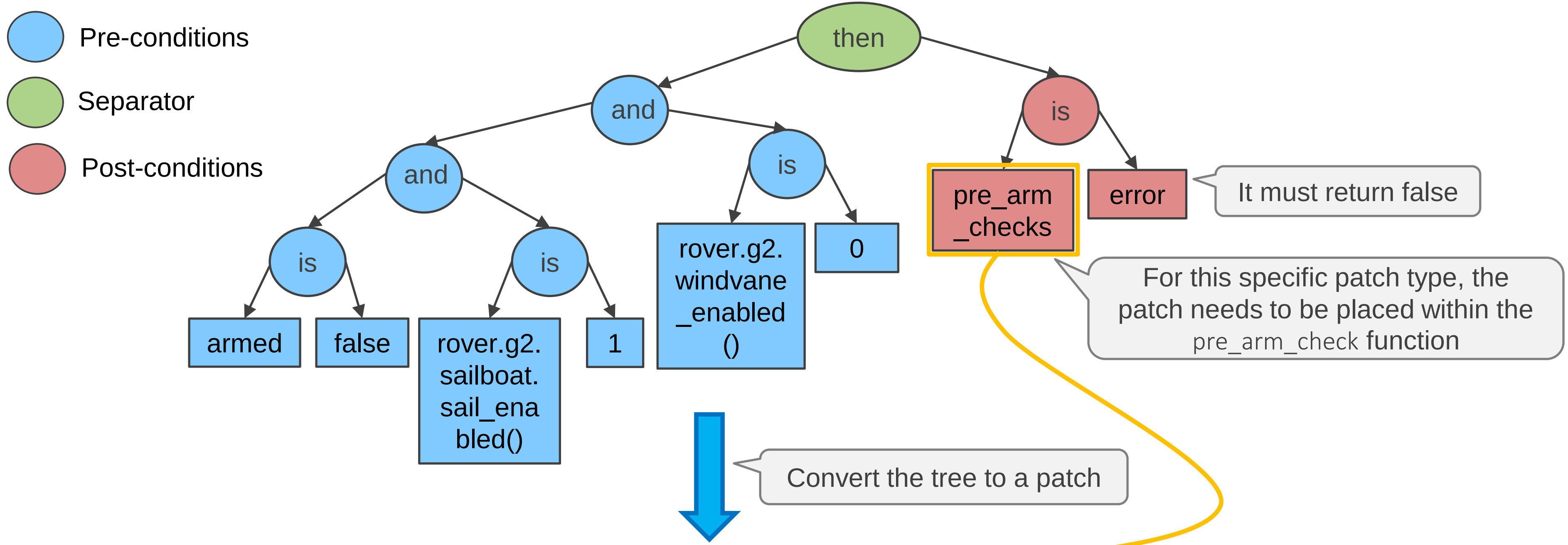
Term	How to access?
armed	armed
SAIL_ENABLE	rover.g2.sailboat.sail_enabled()
WNDVN_TYPE	rover.g2.windvane_enabled()
pre-arm-check	pre_arm_checks(true/false)

❹ Patch Generation



Term	How to access?
armed	armed
SAIL_ENABLE	rover.g2.sailboat.sail_enabled()
WNDVN_TYPE	rover.g2.windvane_enabled()
pre-arm-check	pre_arm_checks(true/false)

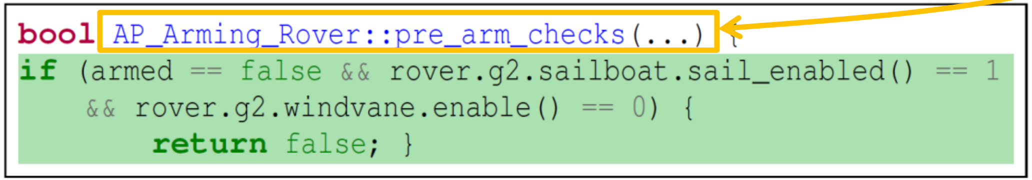
❹ Patch Generation



It must return false

For this specific patch type, the patch needs to be placed within the pre_arm_check function

Convert the tree to a patch



```
bool AP_Arming_Rover::pre_arm_checks(...) {
if (armed == false && rover.g2.sailboat.sail_enabled() == 1
    && rover.g2.windvane.enable() == 0) {
        return false; }
```

Patch Types

- PGPatch supports five patch types
  1) Disabling a statement
  2) Checking valid ranges of configuration parameters
  3) Updating a statement
  4) Adding a condition check
  5) Reusing an existing code snippet

The sailboat patch we have just explained

Please check our paper regarding how PGPatch generates other patch types

Quantitative Evaluation

- RV control software
  - ArduPilot, PX4, and Paparazzi
- Dataset (randomly selected 297 logic bugs)
  - 94 logic bugs from GitHub commit history
  - 203 logic bugs from RV fuzzing works[1]
- Results
  - PGPatch succeeds in fixing 258 out of 297 bugs (86.9%).

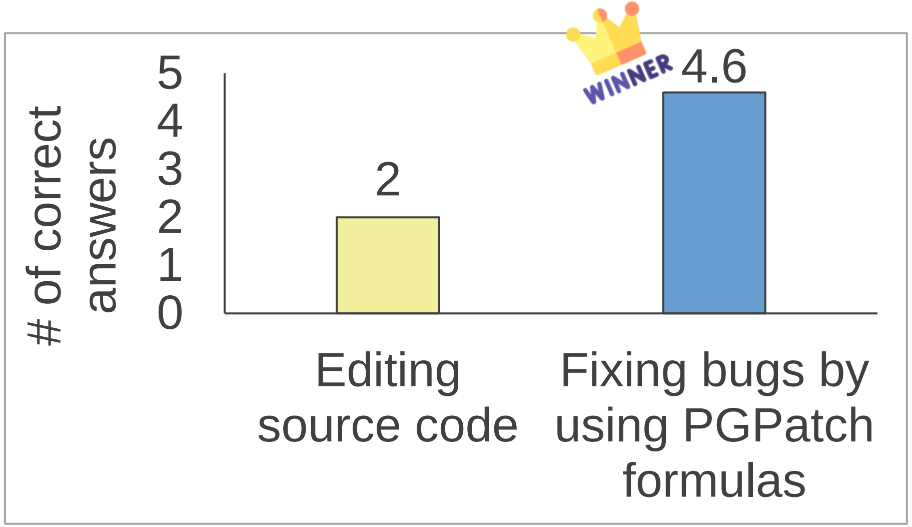
[1] PGFuzz: Policy-Guided Fuzzing for Robotic Vehicles, NDSS'21.
RVFuzzer: Finding Input Validation Bugs in Robotic Vehicles through Control-Guided Testing, USENIX'19.

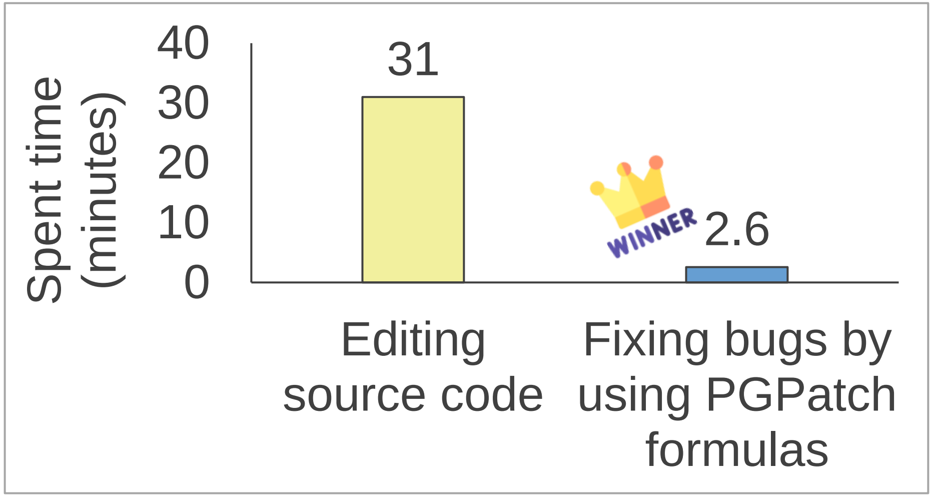
User Study

- We aim to determine
  - How efficient PGPatch is in patching logic bugs compared to manual patching
- Method
  - Recruit 6 RV developers and 12 experienced RV users
    - 1 subject was an official ArduPilot developer
  - Ask participants to create:
    - 5 PGPatch formulas
    - 5 corresponding source-level patches

User Study

- Correctness
  - 2 (editing source code) vs. 4.6 (fixing bugs through PGPatch)
- Spent time
  - 31 mins (editing source code) vs. 2.6 mins (fixing bugs through PGPatch)





Summary

- Logic bugs
  - Are the main bug type in RV control software
- PGPatch
  - Novel program repair approach to fix logic bugs
    - Reuse existing formulas
  - Supports five patch types
  - Is less error-prone compared to manually patching bugs

Thank you! Questions?

kim2956@purdue.edu

https://github.com/purseclab/PGPatch

PURDUE UNIVERSITY

CERIAS

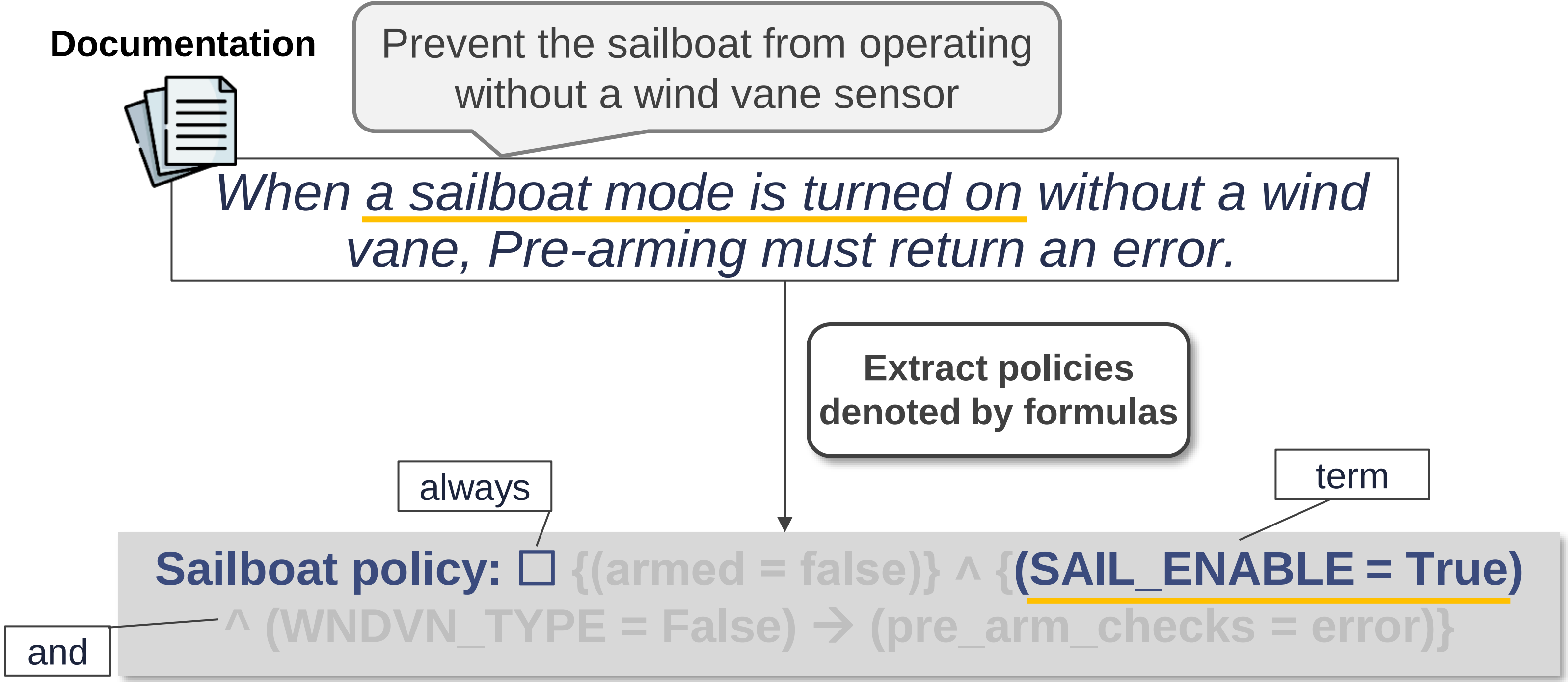
PurSecLab

Backup slides

Motivation

- Logic bug finding tools (e.g., PGFuzz)
  - More than 100 logic bugs found by PGFUZZ
- Can we automatically fix these bugs?
  - Existing automatic program repair (APR) tools cannot fix the found logic bugs in RVs.
- Do normal users know about temporal logic?
  - No, only 2 out of 18 participants know temporal logic syntax in our user study.

Motivation



Documentation

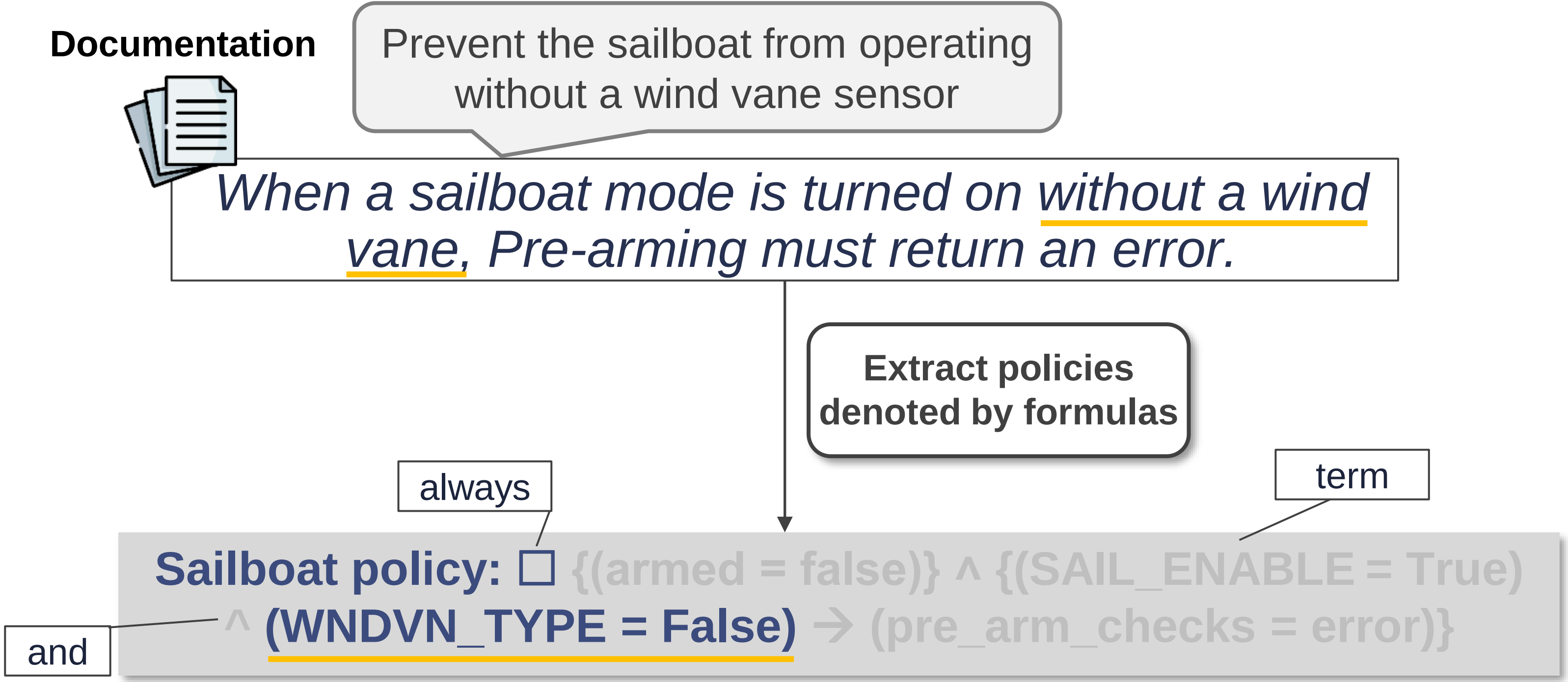
Prevent the sailboat from operating without a wind vane sensor

When a sailboat mode is turned on without a wind vane, Pre-arming must return an error.

Extract policies denoted by formulas

Sailboat policy: □ {(armed = false)} ^ {(SAIL_ENABLE = True) ^ (WNDVN_TYPE = False) → (pre_arm_checks = error)}

Motivation



Documentation

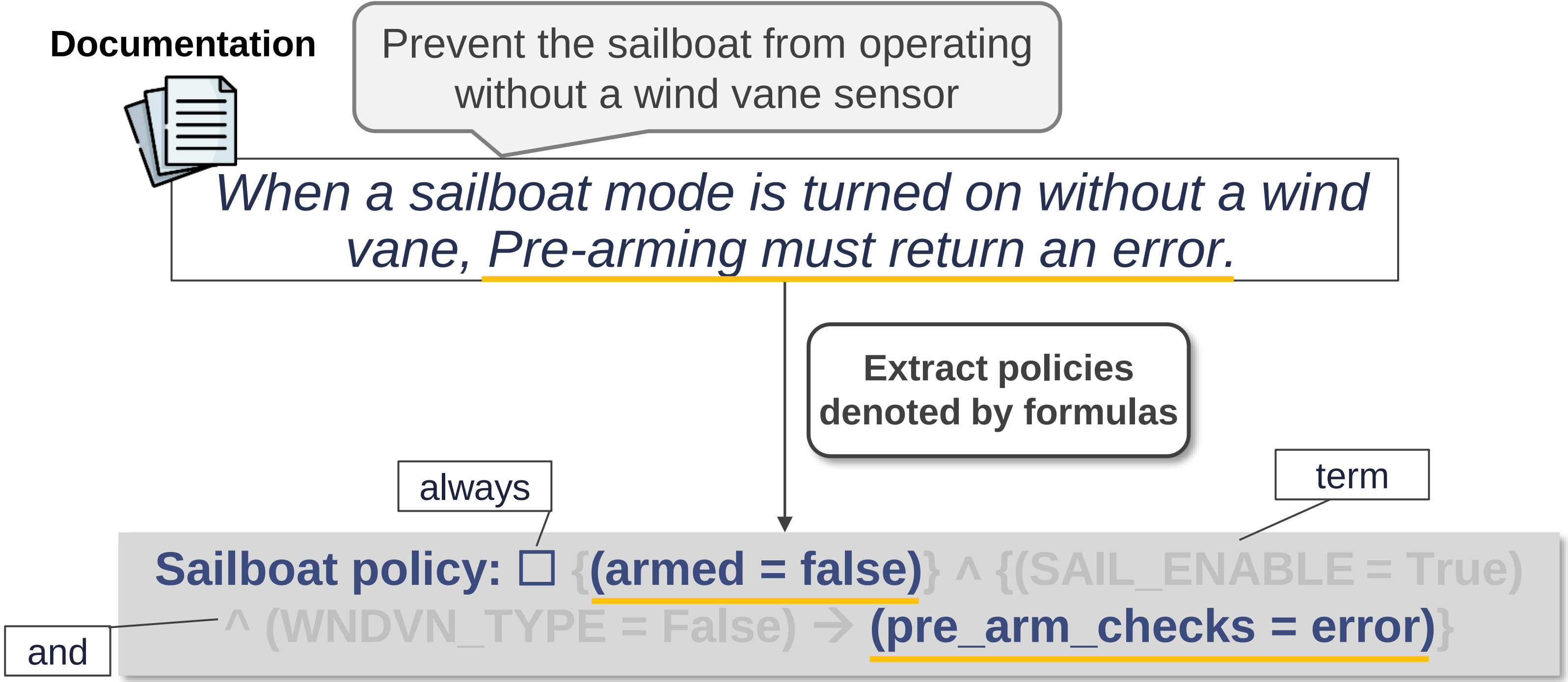
Prevent the sailboat from operating without a wind vane sensor

When a sailboat mode is turned on without a wind vane, Pre-arming must return an error.

Extract policies denoted by formulas

Sailboat policy: □ {(armed = false)} ∧ {(SAIL_ENABLE = True) ∧ **(WNDVN_TYPE = False)** → (pre_arm_checks = error)}

Motivation



Documentation

Prevent the sailboat from operating without a wind vane sensor

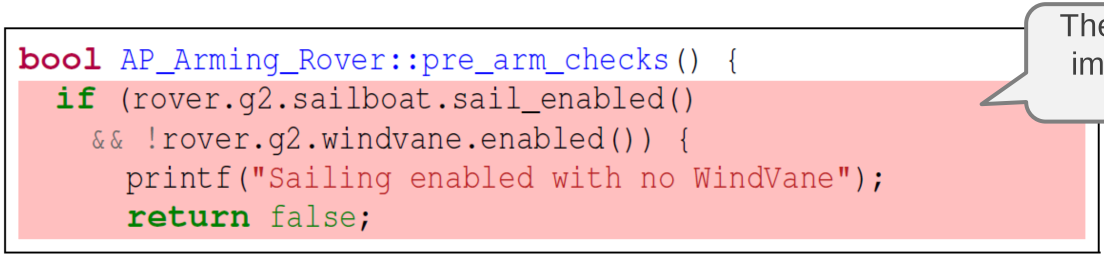
When a sailboat mode is turned on without a wind vane, Pre-arming must return an error.

Extract policies denoted by formulas

Sailboat policy: □ {(armed = false)} ^ {(SAIL_ENABLE = True) ^ (WNDVN_TYPE = False) → (pre_arm_checks = error)}

Motivation

- The sailboat policy must hold
  - Boat-type RVs cannot navigate to a waypoint without the wind direction obtained from the wind vane



```
bool AP_Arming_Rover::pre_arm_checks() {
  if (rover.g2.sailboat.sail_enabled()
    && !rover.g2.windvane.enabled()) {
      printf("Sailing enabled with no WindVane");
      return false;
```

The RV software initially did not implement this policy, causing potential safety violations

Can we automatically fix this bug through existing tools?

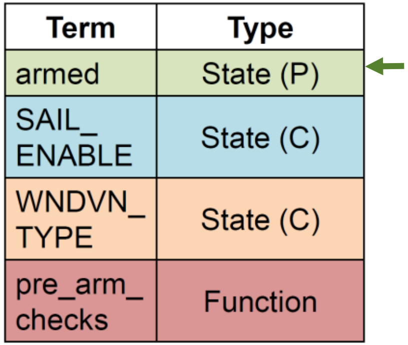
Preprocessor

- How to classify terms into three different types?
  - Physical states
  - Configuration parameters
  - Functions

Manually build a list of physical states

States	Synonym
Altitude	alt, height
arm	armed
roll	…
…	…

<A list of physical states in the RV software>



Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

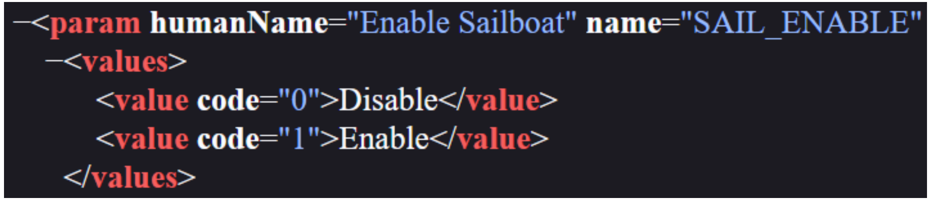
State (P): an RV's physical state, **State (C)**: an RV's configuration state

Preprocessor

- How to classify terms into three different types?
  - Physical states
  - Configuration parameters
  - Functions

Parse XML files

Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function



```
−<param humanName="Enable Sailboat" name="SAIL_ENABLE"
  −<values>
      <value code="0">Disable</value>
      <value code="1">Enable</value>
   </values>
```

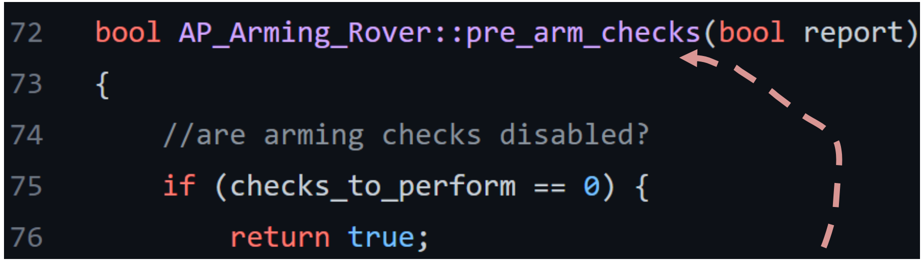
<An XML file contains a full list of configuration parameters in the RV software>

State (P): an RV’s physical state, **State (C)**: an RV’s configuration state

Preprocessor

- How to classify terms into three different types?
  - Physical states
  - Configuration parameters
  - Functions

Write LLVM passes to obtain all names of function



```
bool AP_Arming_Rover::pre_arm_checks(bool report)
{
    //are arming checks disabled?
    if (checks_to_perform == 0) {
        return true;
```

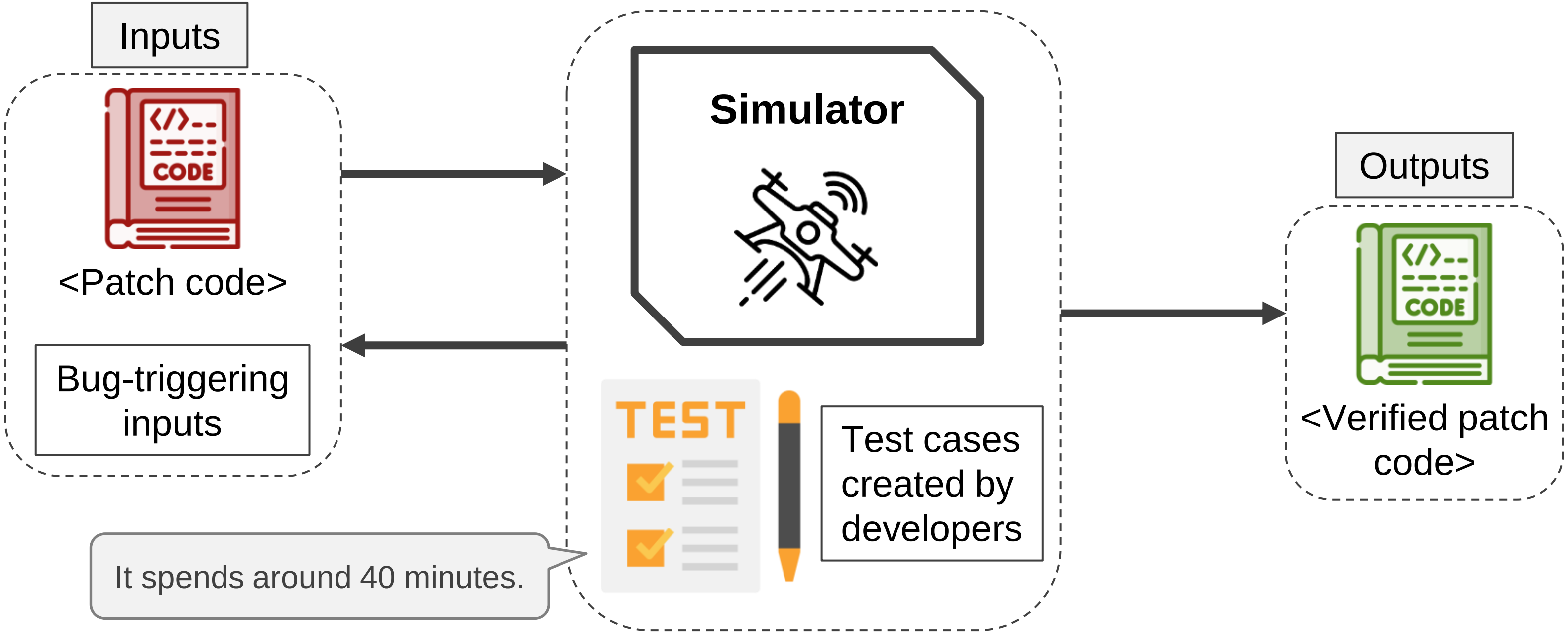
<Source code in ArduPilot>

Term	Type
armed	State (P)
SAIL_ ENABLE	State (C)
WNDVN_ TYPE	State (C)
pre_arm_ checks	Function

State (P): an RV's physical state, **State (C)**: an RV's configuration state

Patch Verification

- Verify whether
  - The buggy behavior occurs in the different test cases
  - The patch breaks the other functionalities



It spends around 40 minutes.

Case Study

< Documentation >

If battery fail-safe mode is triggered and the home_distance between the drone and the GCS is less than 100 meters, then the drone's flight mode must switch to LAND mode.

```
if (home_distance() < 10,000) {
    desired_action = LAND;
}
```

<Patch code snippet in *ArduCopter/event.cpp*>

- **The patch code looks simple.**
- **Yet, 8 out of 18 participants correctly fixed this bug.**
- **To fix it, they spent, on average: 40 mins (RV users) and 20 mins (RV developers).**

GCS: ground control system

Case Study

- Why did they spend so much time fixing this bug?

```
if (home_distance() < 10,000) {
    desired_action = LAND;
}
```

<Patch code snippet in *ArduCopter/event.cpp*>

Tricky part 1

Most of the subjects failed to locate the correct patch location because a total of 65 source code files include 'failsafe' logic.

Tricky part 2

Documentation mentions 100 meters, but some code locations leverage different metrics (e.g., centimeters).

Case Study

- 17 out of the 18 subjects correctly created a PGPatch formula
  - They spent, on average, 2.2 minutes.

Fail-safe policy in PPL syntax: If fail-safe is on and home_distance is less than 100, then mode is LAND

- PGPatch created a patch from the formula.

1) PGPatch uses the correct unit

2) PGPatch inserts this patch into each candidate patch location and conducts the patch verification

```
if (home_distance()/100 < 100) {
    desired_action = LAND;
}
```

<Patch code snippet created by PGPatch>