Automated Test Case Generation for an Autopilot Requirement Prototype

Dimitra Giannakopoulou, Neha Rungta, and Michael Feary
NASA Ames Research Center
motivation

- need for Human – Automation Interaction (HAI) test support in the aircraft certification and approval process
- existing formal method algorithms and framework might help
- but any results must be transparent and usable by evaluator

automated test-case generation through symbolic execution
concept

source code (main method)

symbolic execution to derive execution paths

Usability Test
why symbolic execution?

```java
@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
```
void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
why symbolic execution?

```java
@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
  if (x > 0)
    y = y + x;
  else
    y = y - x;
}
```
why symbolic execution?

@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
why symbolic execution?

@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
why symbolic execution?

```java
@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
```
why symbolic execution?

```java
@Symbolic("true")
int x;
@Symbolic("true")
int y;

void testX() {
    if (x > 0)
        y = y + x;
    else
        y = y - x;
}
```

Test Input Generation

<table>
<thead>
<tr>
<th>X</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x &gt; 0)</td>
</tr>
<tr>
<td>0</td>
<td>(x \leq 0)</td>
</tr>
</tbody>
</table>
...when successful, automated test case generation automatically generates high quality test suites for full path coverage
Step 1: ADEPT to Java
Autopilot Example
```java
if (!isNominal && ((outputState == 1) || (outputState == 2)) &&
    selectedLateralTargetError > 179 &&
    (userPressesLateralTargetButton == true &&
     userPressesLateralHoldButton == false &&
     userPressesLNAVbutton == false)) {
    applyRule06();
}

if (!isNominal && ((outputState == 1) || (outputState == 2)) &&
    selectedLateralTargetError < -179 &&
    (userPressesLateralTargetButton == true &&
     userPressesLateralHoldButton == false &&
     userPressesLNAVbutton == false)) {
    applyRule07();
}

public void applyRule06() {
    outputState = 0;
    selectedLateralTargetError -= 360;
    selectedLateralTarget =
        preSelectedLateralTarget;
    lateralTarget = selectedLateralTarget;
    lateralTargetError =
        selectedLateralTargetError;
}

public void applyRule07() {
    outputState = 0;
    selectedLateralTargetError += 360;
    selectedLateralTarget =
        preSelectedLateralTarget;
    lateralTarget = selectedLateralTarget;
    lateralTargetError =
        selectedLateralTargetError;
}
```
Step 2: Symbolic Execution
what do we execute symbolically?

- method **execute** – parameters are user inputs (e.g., button presses) and are symbolic
- other (not user input) variables in the table that appear in rule conditions are eligible to be treated as symbolic; this allows us to explore different initial values that may lead us to different paths
- the **main** method calls method **execute** \( n \) times (\( n \) can be selected); each time, fresh values are picked for the symbolic parameters since each time the user input actions may vary
...
results and challenges

- automatically generated 16 test cases for n=1
- discovered through unsatisfiable path constraints that some rules disable each other

- (HAI challenge) provide support for modeling semantics of user interface components such momentary vs. toggle switch
- (HAI challenge) define coverage criteria – for example related to covering modes; also what values should we pick for n (what length of user inputs)?

- (generic challenge) scalability of symbolic execution

NASA  Federal Aviation Administration

Generic
thank you!

dimitra.giannakopoulou@nasa.gov
neha.s.rungta@nasa.gov
michael.s.feary@nasa.gov
symbolic execution for ADEPT HAI models

ADEPT Model → translate model → Executable Java Program

this represents the abstract model and is different than the prototype that ADEPT generates

Java PathFinder Symbolic Execution → tests!