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
source code (main method)
symbolic execution to derive execution paths
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;
}
```
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;
}
```

[Diagram showing symbolic execution with conditions and states]
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;
}

x:X, y:Y
PC: true

x:X, y:Y
PC: X > 0

x:X, y:Y
PC: X <= 0

x:X, y:Y+X
PC: X > 0

x:X, y:Y-X
PC: X <= 0
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:X, y:Y</td>
</tr>
<tr>
<td></td>
<td>PC: X &gt; 0</td>
</tr>
<tr>
<td>0</td>
<td>x:X, y:Y-X</td>
</tr>
<tr>
<td></td>
<td>PC: X &lt;= 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
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.
if(!isNominal && ((outputState == 1) ||
(outputState == 2)) &&
selectedLateralTargetError > 179 &&
(userPressesLateralTargetButton == true &&
userPressesLateralHoldButton == false &&
userPressesNAVbutton == false)){
    applyRule06();
}
if(!isNominal &&((outputState == 1) ||
(outputState == 2)) &&
selectedLateralTargetError < -179 &&
(userPressesLateralTargetButton == true &&
userPressesLateralHoldButton == false &&
userPressesNAVbutton == 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; }
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
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
  - Java PathFinder Symbolic Execution

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