Automated Test Case Generation for an Autopilot Requirement Prototype

Dimitra Giannakopoulou, Neha Rungta, and Michael Feary

NASA Ames Research Center

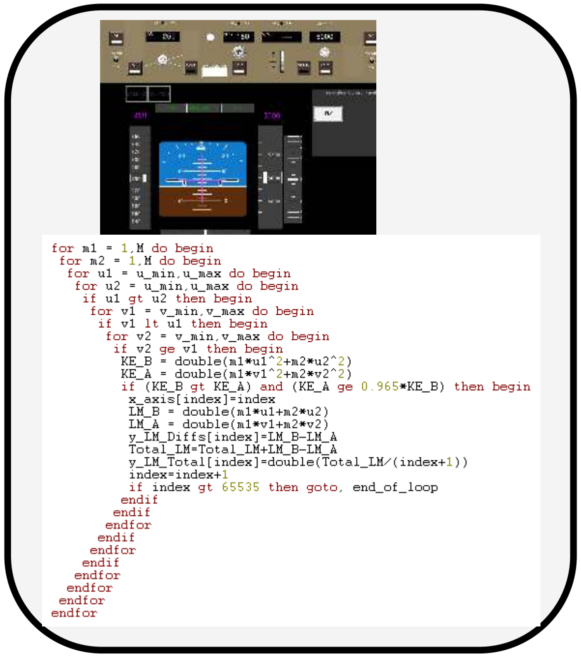
Federal Aviation Administration

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

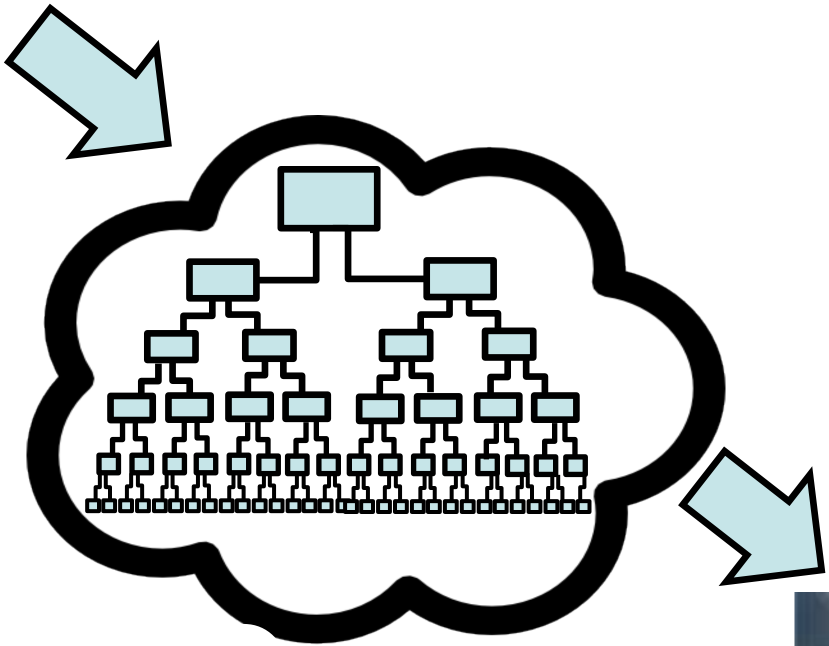
automated test-case generation through symbolic execution

concept



```
for m1 = 1,M do begin
 for m2 = 1,M do begin
  for u1 = u_min,u_max do begin
   for u2 = u_min,u_max do begin
    if u1 gt u2 then begin
     for v1 = v_min,v_max do begin
      if v1 lt u1 then begin
       for v2 = v_min,v_max do begin
        if v2 ge v1 then begin
         KE_B = double(m1*u1^2+m2*u2^2)
         KE_A = double(m1*v1^2+m2*v2^2)
         if (KE_B gt KE_A) and (KE_A ge 0.965*KE_B) then begin
          x_axis[index]=index
          LM_B = double(m1*u1+m2*u2)
          LM_A = double(m1*v1+m2*v2)
          y_LM_Diffs[index]=LM_B-LM_A
          Total_LM=Total_LM+LM_B-LM_A
          y_LM_Total[index]=double(Total_LM/(index+1))
          index=index+1
          if index gt 65535 then goto, end_of_loop
         endif
        endif
       endfor
      endif
     endfor
    endif
   endfor
  endfor
 endfor
endfor
```

source code
(main method)



symbolic execution to
derive execution paths

Usability Test

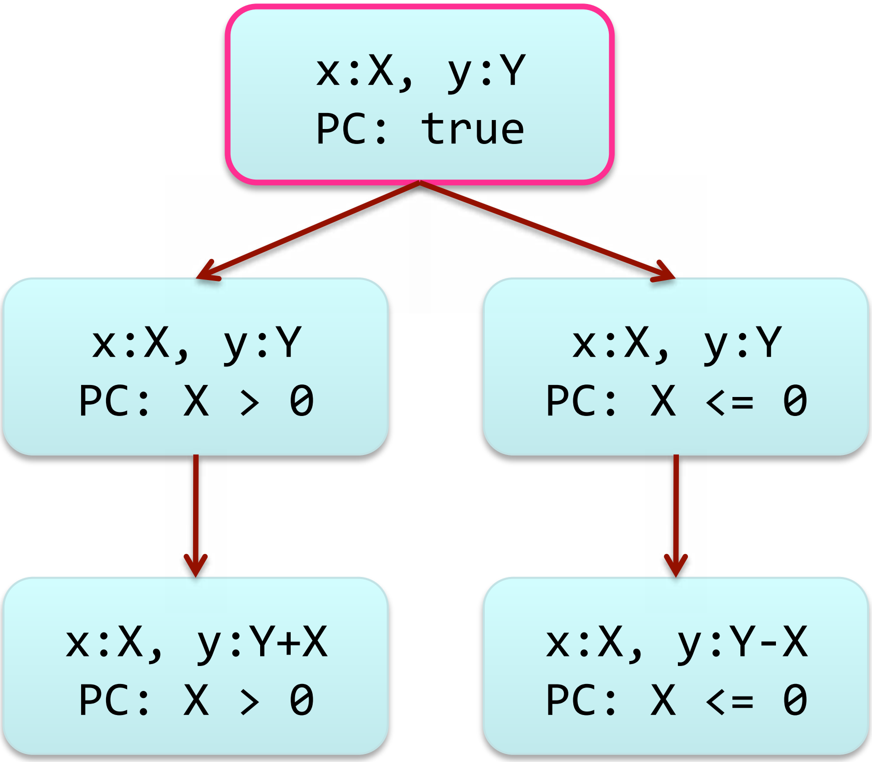
Federal Aviation
Administration

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?

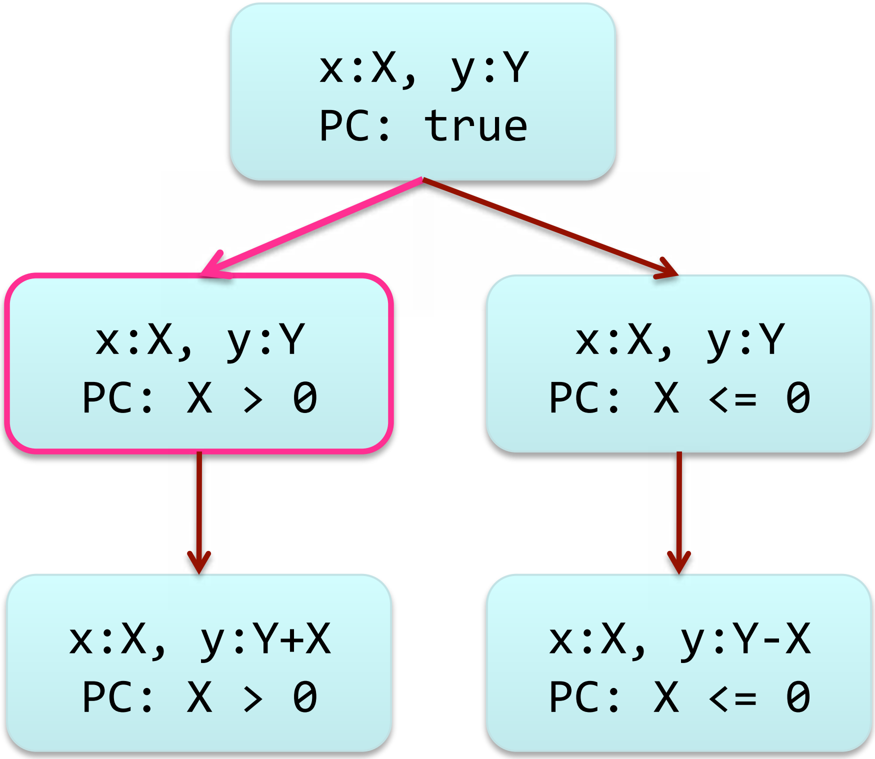
```
@Symbolic(“true”)
int x;
@Symbolic(“true”)
int y;

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



Federal Aviation Administration

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

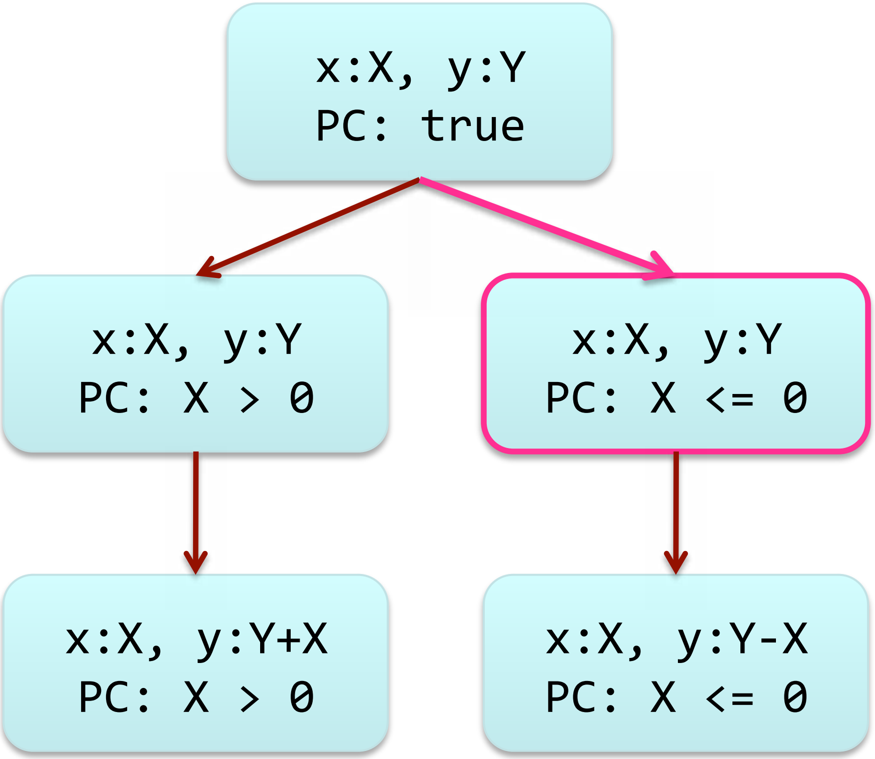
x:X, y:Y
PC: X <= 0

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

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

why symbolic execution?

```
@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

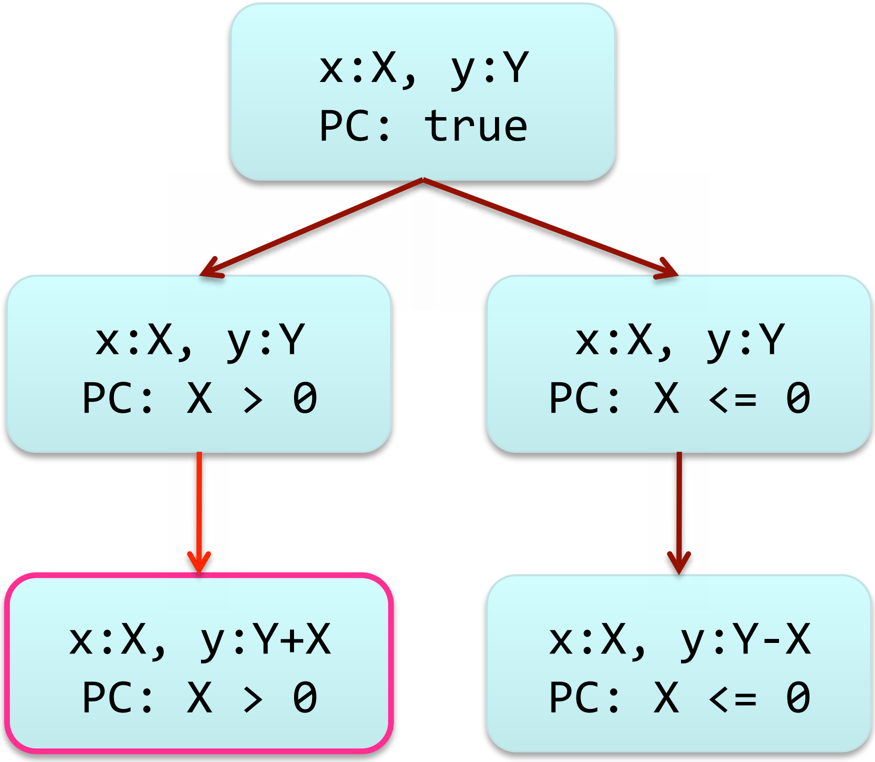
x:X, y:Y+X
PC: X > 0

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

Federal Aviation
Administration

why symbolic execution?

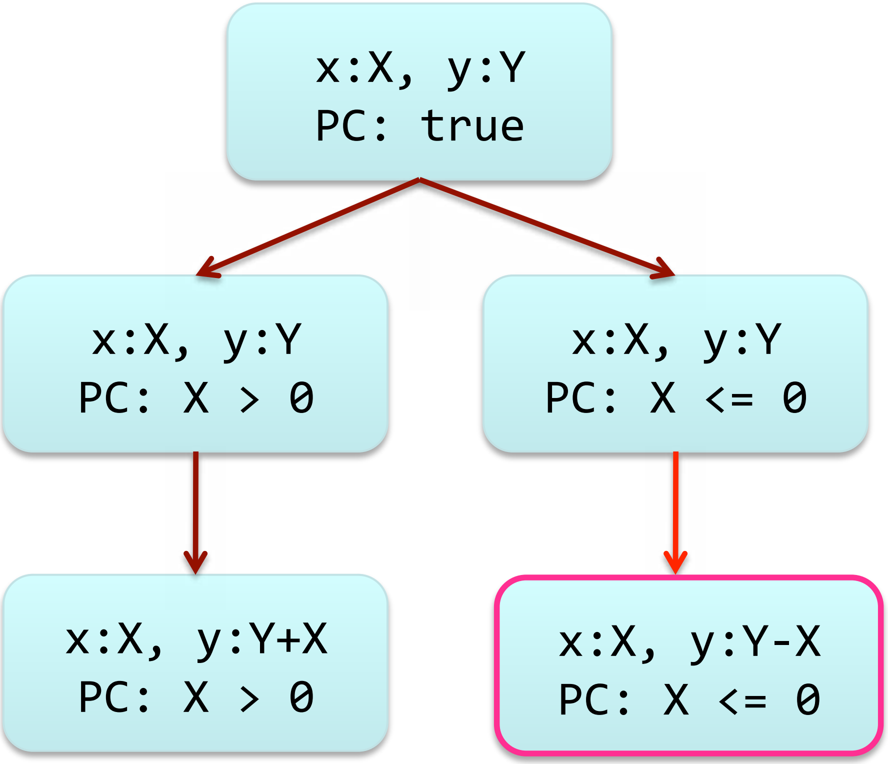
```
@Symbolic(“true”)
int x;
@Symbolic(“true”)
int y;

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



Federal Aviation
Administration

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

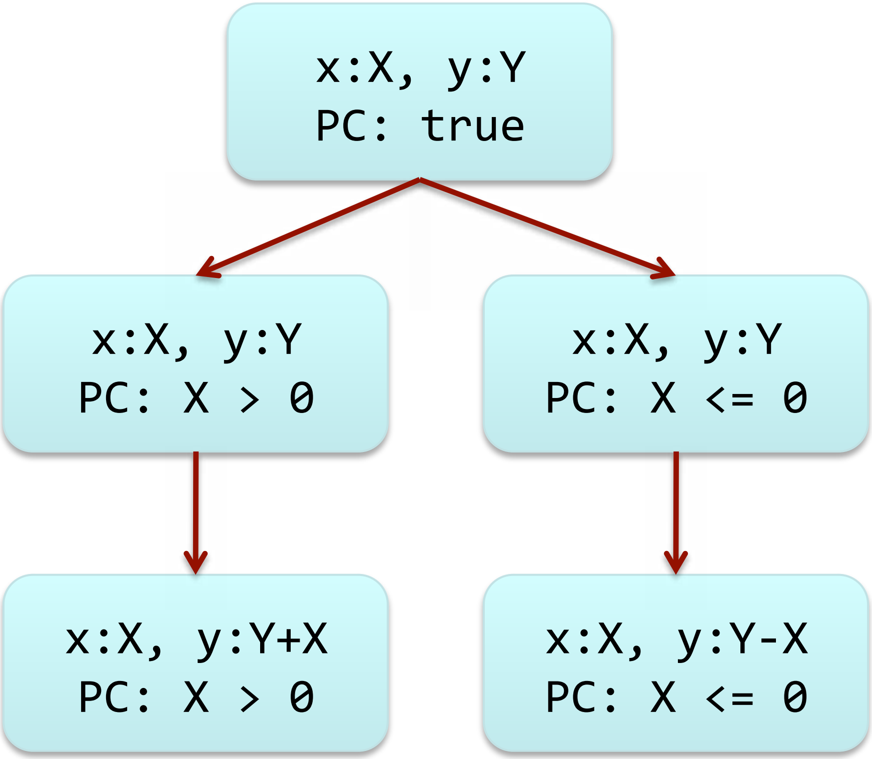
x:X, y:Y+X
PC: X > 0

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

Federal Aviation
Administration

why symbolic execution?

```
@Symbolic(“true”)
int x;
@Symbolic(“true”)
int y;

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



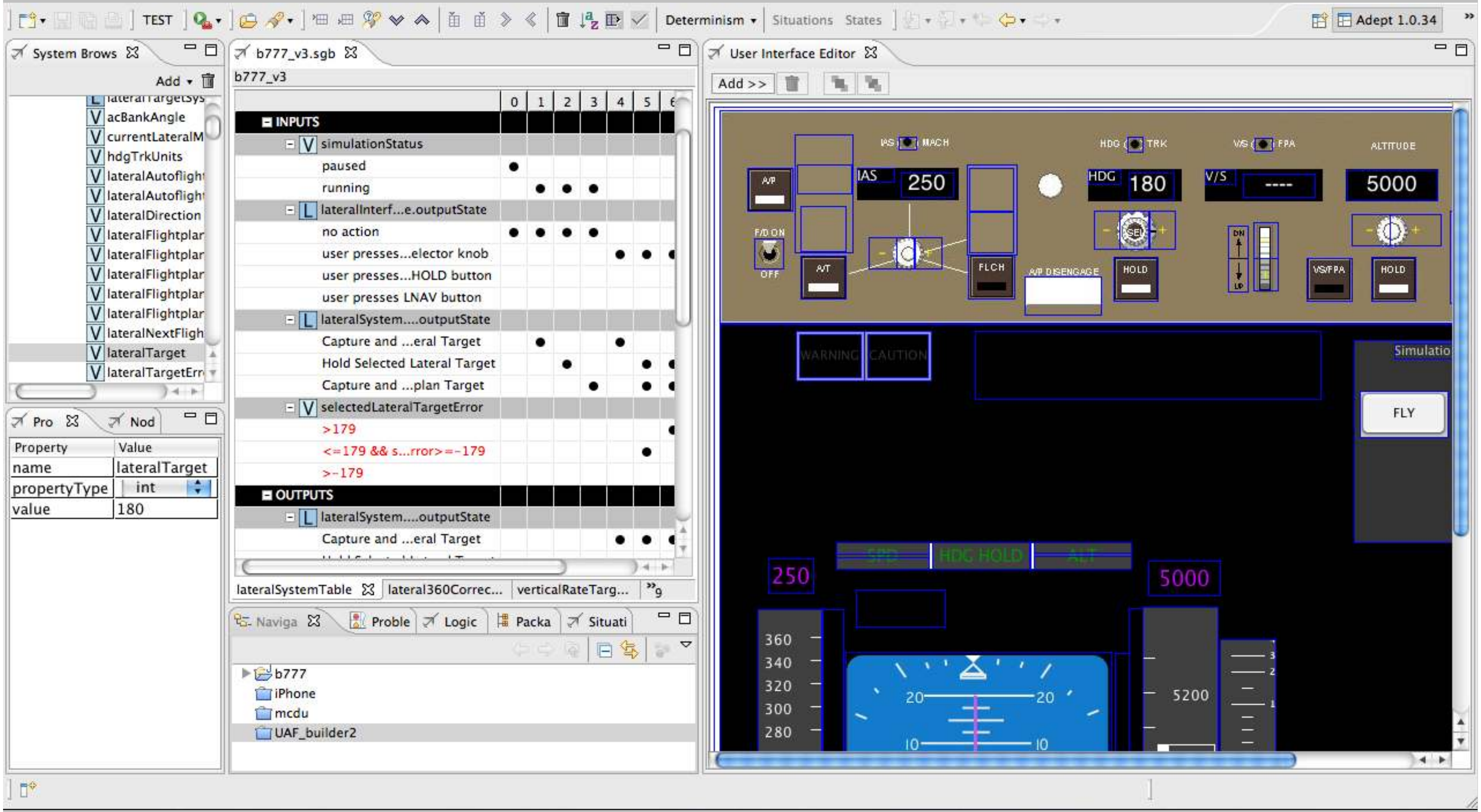
Test Input Generation X = 1 X = 0

Federal Aviation
Administration

…when successful, automated test case generation automatically generates high quality test suites for full path coverage

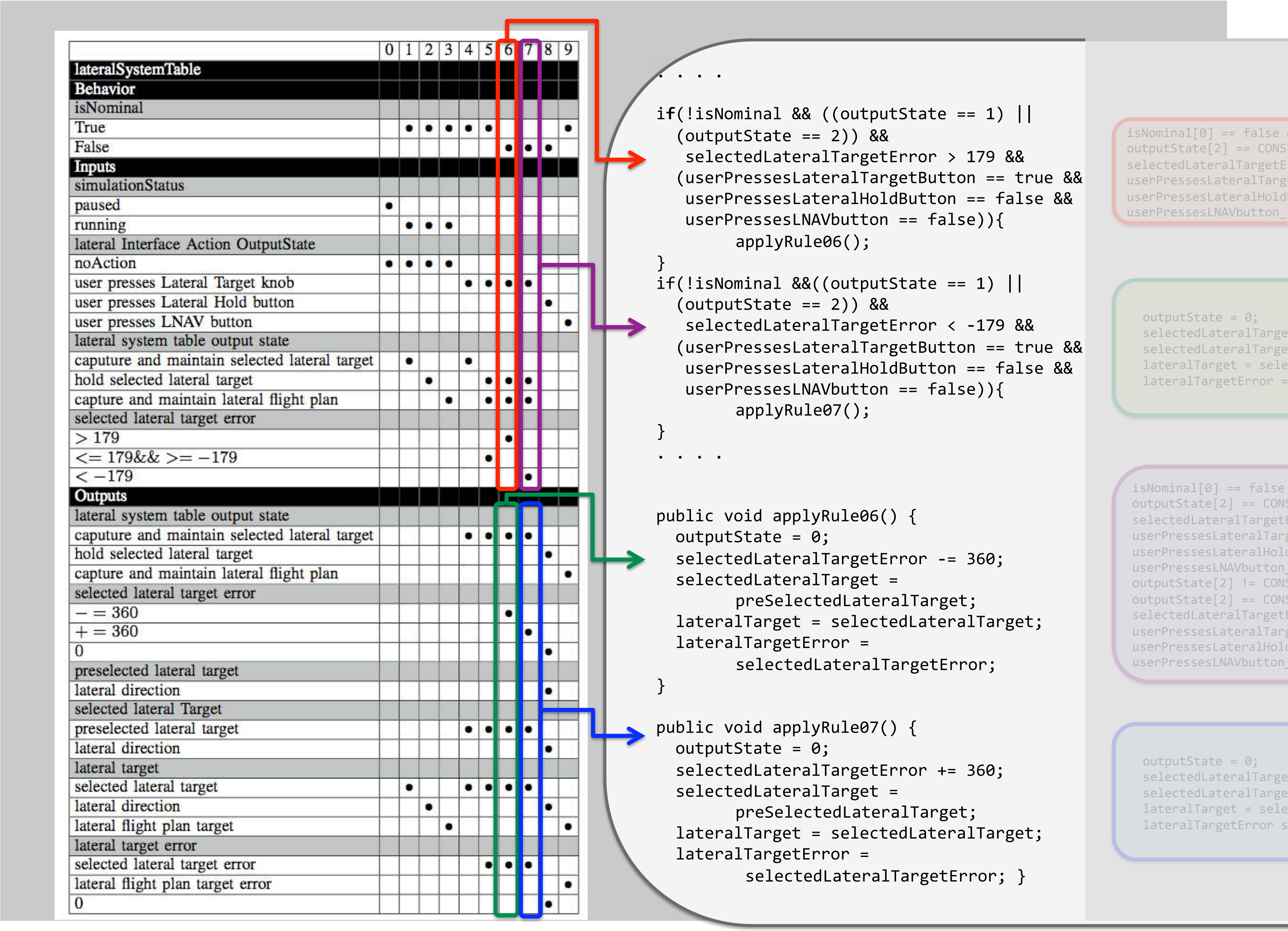
Step 1: ADEPT to Java

Autopilot Example



Property	Value
name	lateralTarget
propertyType	int
value	180

	0	1	2	3	4	5	6
INPUTS							
V simulationStatus							
paused	●						
running		●	●	●			
L lateralInterf...e.outputState							
no action	●	●	●	●			
user presses...elector knob					●	●	●
user presses...HOLD button							
user presses LNAV button							
L lateralSystem....outputState							
Capture and ...eral Target		●			●		
Hold Selected Lateral Target			●			●	●
Capture and ...plan Target				●		●	●
V selectedLateralTargetError							
>179							●
<=179 && s...rror>=-179						●	
>-179							
OUTPUTS							
L lateralSystem....outputState							
Capture and ...eral Target					●	●	●



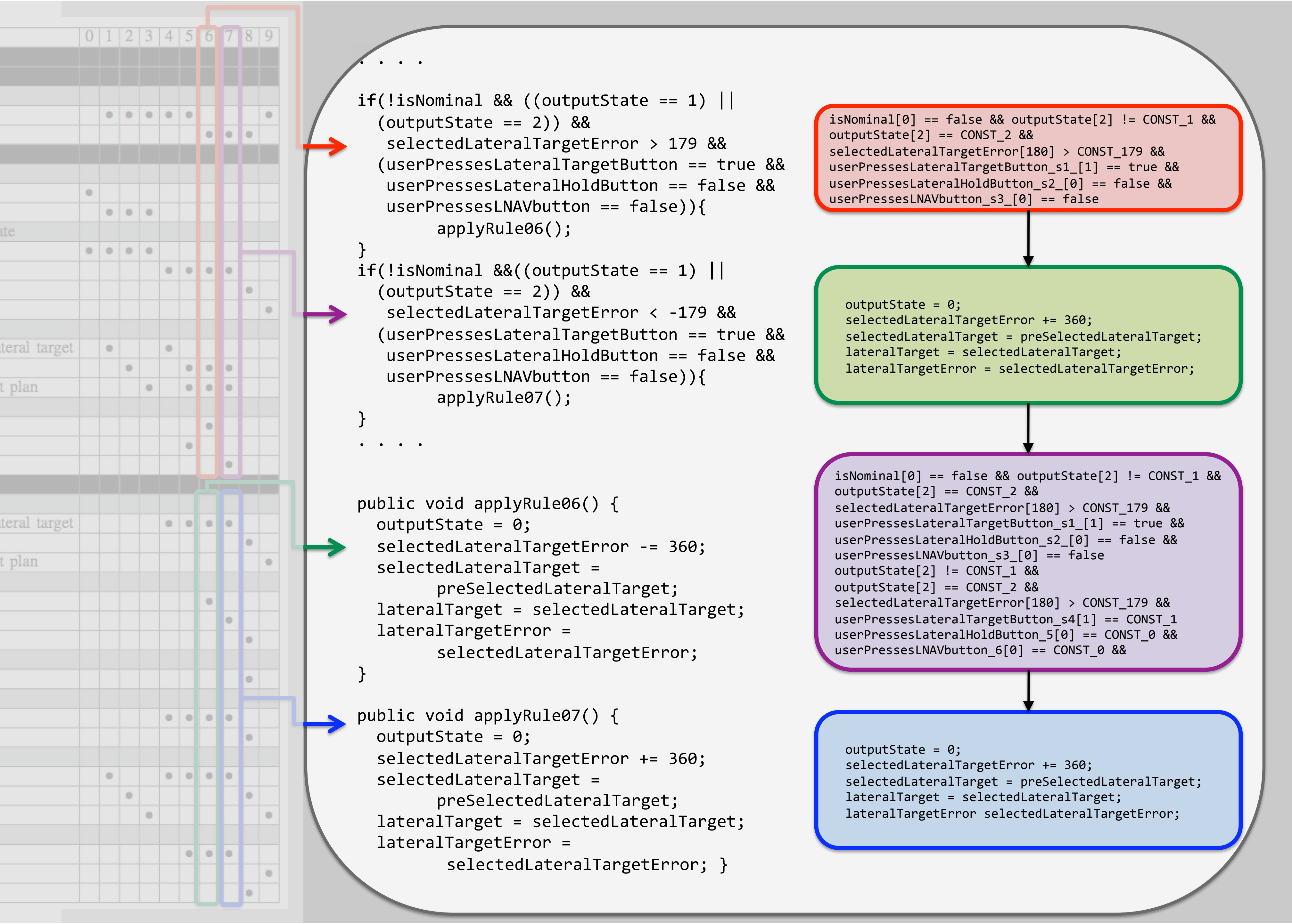
	0	1	2	3	4	5	6	7	8	9
lateralSystemTable										
Behavior										
isNominal										
True		●	●	●	●	●				●
False							●	●	●	
Inputs										
simulationStatus										
paused	●									
running		●	●	●						
lateral Interface Action OutputState										
noAction	●	●	●	●						
user presses Lateral Target knob					●	●	●	●		
user presses Lateral Hold button									●	
user presses LNAV button										●
lateral system table output state										
caputure and maintain selected lateral target		●			●					
hold selected lateral target			●			●	●	●		
capture and maintain lateral flight plan				●		●	●	●		
selected lateral target error										
> 179							●			
$<= 179\&\& >= -179$						●				
< -179								●		
Outputs										
lateral system table output state										
caputure and maintain selected lateral target					●	●	●	●		
hold selected lateral target									●	
capture and maintain lateral flight plan										●
selected lateral target error										
$- = 360$							●			
$+ = 360$								●		
0									●	
preselected lateral target										
lateral direction									●	
selected lateral Target										
preselected lateral target					●	●	●	●		
lateral direction									●	
lateral target										
selected lateral target		●			●	●	●	●		
lateral direction			●						●	
lateral flight plan target				●						●
lateral target error										
selected lateral target error						●	●	●		
lateral flight plan target error										●
0									●	

```
. . . .

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 (eg 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 &&
   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; }
```

```
isNominal[0] == false && outputState[2] != CONST_1 &&
outputState[2] == CONST_2 &&
selectedLateralTargetError[180] > CONST_179 &&
userPressesLateralTargetButton_s1_[1] == true &&
userPressesLateralHoldButton_s2_[0] == false &&
userPressesLNAVbutton_s3_[0] == false
```

```
outputState = 0;
selectedLateralTargetError += 360;
selectedLateralTarget = preSelectedLateralTarget;
lateralTarget = selectedLateralTarget;
lateralTargetError = selectedLateralTargetError;
```

```
isNominal[0] == false && outputState[2] != CONST_1 &&
outputState[2] == CONST_2 &&
selectedLateralTargetError[180] > CONST_179 &&
userPressesLateralTargetButton_s1_[1] == true &&
userPressesLateralHoldButton_s2_[0] == false &&
userPressesLNAVbutton_s3_[0] == false
outputState[2] != CONST_1 &&
outputState[2] == CONST_2 &&
selectedLateralTargetError[180] > CONST_179 &&
userPressesLateralTargetButton_s4[1] == CONST_1
userPressesLateralHoldButton_5[0] == CONST_0 &&
userPressesLNAVbutton_6[0] == CONST_0 &&
```

```
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)?

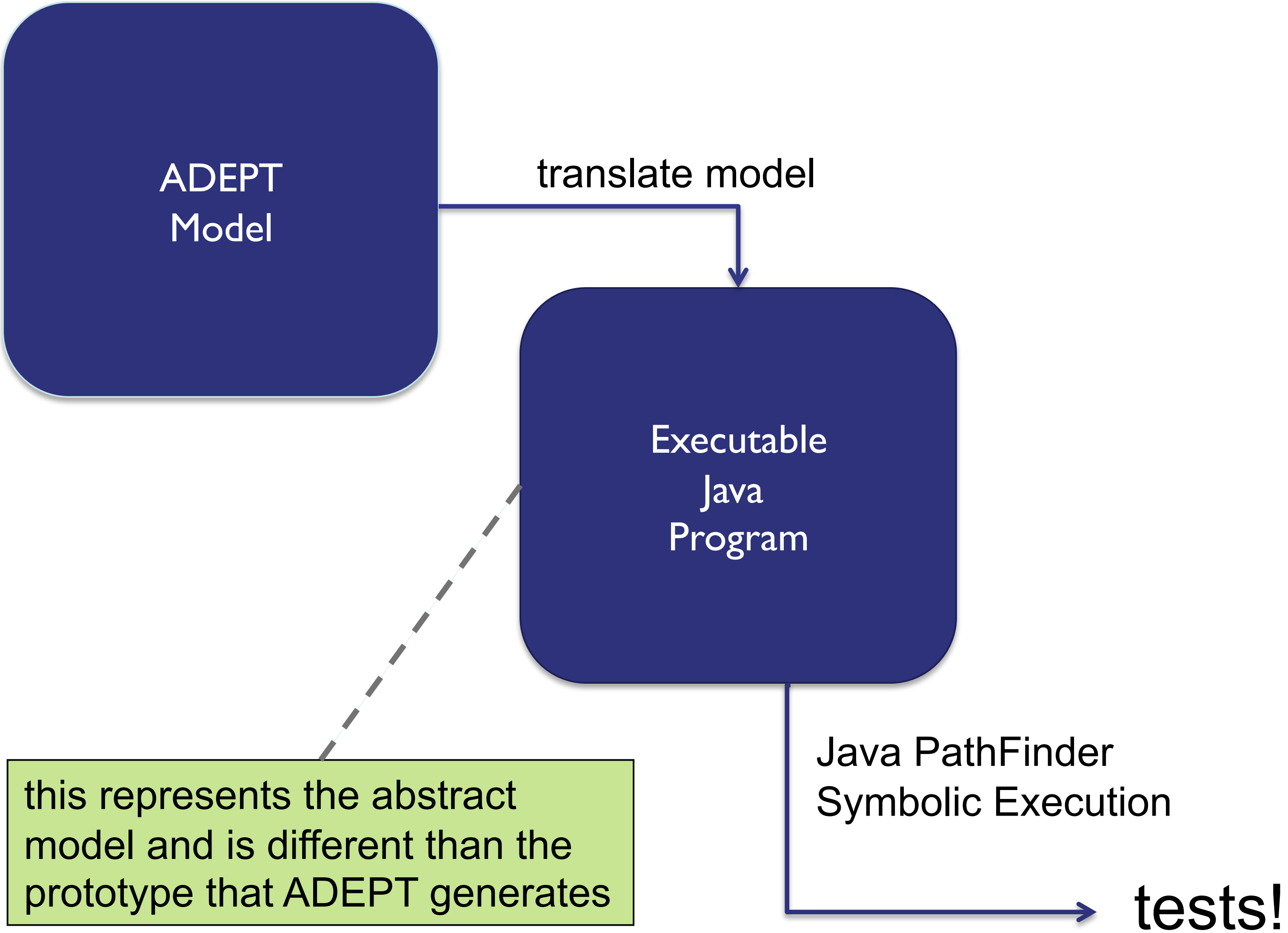
- (generic challenge) scalability of symbolic execution

- 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!

Federal Aviation Administration