Symbolic Evaluation/Execution
Today's Reading Material

Symbolic Evaluation/Execution

• Creates a functional representation of a path of an executable component

• For a path $\Pi$
  • $D[\Pi]$ is the domain for path $\Pi$
  • $C[\Pi]$ is the computation for path $\Pi$
Functional Representation of an Executable Component

\[ P : X \rightarrow Y \]

\( P \) is composed of partial functions corresponding to the executable paths

\[ P = \{P_1, \ldots, P_r\} \]

\[ P_i : X_i \rightarrow Y \]
Functional Representation of an Executable Component

$X_i$ is the domain of path $P_i$

Denoted $D[\ P_i\ ]$

$X = D[P_1] \cup \ldots \cup D[P_r] = D[P]$

$D[P_i] \cap D[P_j] = \emptyset, \ i \neq j$
Representing Computation

- **Symbolic names** represent the input values.
- The **path value PV** of a variable for a path describes the value of that variable in terms of those symbolic names.
- The **computation** of the path $C[P]$ is described by the path values of the outputs for the path.
Representing Conditionals

• an interpreted branch condition or interpreted predicate is represented as an inequality or equality condition

• the path condition $PC$ describes the domain of the path and is the conjunction of the interpreted branch conditions

• the domain of the path $D[P]$ is the set of input values that satisfy the PC for the path
Example program

procedure Contrived is
   X, Y, Z : integer;
1   read X, Y;
2   if X \geq 3 then
3      Z := X+Y;
   else
4      Z := 0;
   endif;
5   if Y > 0 then
6      Y := Y + 5;
   endif;
7   if X - Y < 0 then
8      write Z;
   else
9      write Y;
   endif;
end Contrived;

Stmt   PV             PC
1 X← x    true
     Y← y
2,3 Z ← x+y    true \land x \geq 3 = x \geq 3
5,6 Y ← y+5   x \geq 3 \land y > 0
7,9 x \geq 3 \land y > 0 \land x - (y+5) \geq 0
    = x \geq 3 \land y > 0 \land (x-y) \geq 5
Presenting the results

procedure Contrived is
X, Y, Z : integer;
read X, Y;
if X ≥ 3 then
   Z := X+Y;
else
   Z := 0;
endif;
if Y > 0 then
   Y := Y + 5;
endif;
if X - Y < 0 then
   write Z;
else
   write Y;
endif
end Contrived

$\begin{array}{cll}
\text{Statements} & \text{PV} & \text{PC} \\
1 & X \leftarrow x & \text{true} \\
2 & Y \leftarrow y & \\
2,3 & Z \leftarrow x+y & \text{true } \land x \geq 3 = x \geq 3 \\
5,6 & Y \leftarrow y+5 & x \geq 3 \land y > 0 \\
7,9 & x \geq 3 \land y > 0 \land x-(y+5) \geq 0 = x \geq 3 \land y > 0 \land (x-y) \geq 5 \\
\end{array}$
Results (feasible path)

\[ y > 0 \]
\[ x \geq 3 \]
\[ (x-y) \geq 5 \]

\[ P = 1, 2, 3, 5, 6, 7, 9 \]
\[ D[P] = \{ (x,y) | x \geq 3 \land y > 0 \land x - y \geq 5 \} \]
\[ C[P] = PV.Y = y + 5 \]
Evaluating another path

procedure Contrived is
  X, Y, Z : integer;
  1 read X, Y;
  2 if X ≥ 3 then
  3     Z := X+Y;
  4   else
  5     Z := 0;
  6   endif;
  7 if Y > 0 then
  8     Y := Y + 5;
  9   endif;
  10 if X - Y < 0 then
  11     write Z;
  12   else
  13     write Y;
  14   endif;
end Contrived;

Stmts   PV   PC
  1     X ← x   true
  2,3   Z ← x+Y   true ∧ x≥3 = x≥3
  5,7                       x≥3 ∧ y≤0
  7,8   x≥3 ∧ y≤0 ∧ x-y < 0
procedure EXAMPLE is
  X, Y, Z : integer;
  1   read X, Y;
  2     if X ≥ 3 then
  3       Z := X+Y;
  4     else
  5       Z := 0;
  6     endif;
  7     if Y > 0 then
  8       Y := Y + 5;
  9     endif;
 10    if X - Y < 0 then
 11      write Z;
 12    else
 13      write Y;
 14    endif
end EXAMPLE

Stmts    PV          PC
           1       X← x        true
           Y ← y
           2,3      Z ← x+y      true ∧ x≥3 = x≥3
           5,7      x≥3 ∧ y≤0
           7,8      x≥3 ∧ y≤0 ∧ x-y < 0

P = 1, 2, 3, 5, 7, 8
D[P] = { (x,y) | x≥3 ∧ y≤0 ∧ x-y<0}
infeasible path!
Results (infeasible path)

\[ y \leq 0 \]

\[ x \geq 3 \]

\[ (x-y) < 0 \]
what about loops?

• Symbolic evaluation requires a full path description

• Example Paths
  • P = 1, 2, 3, 5
  • P = 1, 2, 3, 4, 2, 3, 5
  • P = 1, 2, 3, 4, 2, 3, 4, 2, 3, 5
  • Etc.
Symbolic Testing

• Path Computation provides [concise] functional representation of behavior for entire Path Domain

• Examination of Path Domain and Computation often useful for detecting program errors

• Particularly beneficial for scientific applications or applications w/o oracles
**Simple Symbolic Evaluation**

- Provides symbolic representations given path $\Pi$
  - path condition $PC =$
  - path domain $D[\Pi] = \{(x_1, x_1, \ldots, x_1) \mid pc \text{ true}\}$
  - path values $PV.X_1 =$
  - path computation $C[\Pi] =$

\[
P = 1, 2, 3, 5, 6, 7, 9
\]
\[
D[P] = \{(x, y) \mid x \geq 3 \land y > 0 \land x - y \geq 5\}
\]
\[
C[P] = PV.Y = y + 5
\]
Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation
Simplification

• Reduces path condition to a canonical form

• Simplifier often determines consistency

\[ PC = (x \geq 5) \text{ and } (x < 0) \]

• May want to display path computation in simplified and unsimplified form

\[ PV.X = x + (x + 1) + (x + 2) + (x + 3) = 4 \times x + 6 \]
Path Condition Consistency

• strategy = solve a system of constraints
  • theorem prover
    • consistency
  • algebraic, e.g., linear programming
    • consistency and find solutions
  • solution is an example of automatically generated test data

... but, in general we cannot solve an arbitrary system of constraints!
Fault Detection

• Implicit fault conditions
  • E.g. Subscript value out of bounds
  • E.g. Division by zero e.g., Q := N/D
• Create assertion to represent the fault and conjoin with the pc
  • Division by zero assert(divisor ≠ 0)
  • Determine consistency
    \( PC_p \) and \( (PV\text{.divisor} = 0) \)
  • if consistent then error possible
• Must check the assertion at the point in the path where the construct occurs
Checking user-defined assertions

• example
  • Assert \((A > B)\)
  • \(PC \text{ and } (PV.A) \leq PV.B)\)
  • if consistent then assertion not valid
Comparing Fault Detection Approaches

- assertions can be inserted as executable instructions and checked during execution
  - dependent on test data selected
    
    \textit{(dynamic testing)}

- use symbolic evaluation to evaluate consistency
  - dependent on path, but not on the test data
  - looks for violating data in the path domain
Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation
Path Selection

- User selected

- Automated selection to satisfy some criteria
  - e.g., exercise all statements at least once

- Because of infeasible paths, best if path selection done incrementally
**Incremental Path Selection**

- PC and PV maintained for partial path
- Inconsistent partial path can often be salvaged

\[
\begin{align*}
T & \quad X > 0 \\
F &
\end{align*}
\]

\[
\begin{align*}
T & \quad X > 0 \\
F & \quad pc' = pc \land (x \leq 0)
\end{align*}
\]

\[
\begin{align*}
T & \quad X > 3 \\
F & \quad pc'' = pc' \land (x \leq 0) \land (x > 3)
\end{align*}
\]

\[
\begin{align*}
T & \quad pc''' = pc' \land (x > 3) \\
F & \quad \text{infeasible path}
\end{align*}
\]

\[
\begin{align*}
T & \quad pc'' = pc' \land (x \leq 0) \land (x \leq 3) \\
F & \quad \text{CONSISTENT [if pc' is consistent]}
\end{align*}
\]
Path Selection (continued)

Can be used in conjunction with other static analysis techniques to determine path feasibility

- Testing criteria generates a path that needs to be tested
- Symbolic evaluation determines if the path is feasible
  - Can eliminate some paths from consideration
Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation
Test Data Generation

• Simple test date selection: Select test data that satisfies the path condition $pc$

• Error based test date selection
  • Try to select test cases that will help reveal faults
  • Use information about the path domain and path values to select test data
    • e.g., $PV.X = a * (b + 2)$;
      
      $a = 1$ combined with min and max values of $b$
      $b = -1$ combined with min and max values for $a$
Enhanced Symbolic Evaluation Capabilities

- Creates symbolic representations of the Path Domains and Computations
  - “Symbolic Testing”
- Determine if paths are feasible
- Automatic fault detection
  - system defined
  - user assertions
- Automatic path selection
- Automatic Test Data Generation
An Enhanced Symbolic Evaluation System

- **User input**
  - **Path Selection**
  - **Symbolic Execution**
    - **Fault conditions**
      - **Detect inconsistency**
    - **Path condition**
    - **Path values**
      - **Detect inconsistency**
  - **Simplifier**
    - **Inequality Solver**
      - **Simplified path values**
    - **Fault report**
      - **Detect inconsistency**
    - **Path domain**
    - **Test data**
    - **Path computation**
Problems

• Information explosion

• Impracticality of all paths

• Path condition consistency

• Aliasing
  • elements of a compound type e.g., arrays and records
  • pointers
Alias Problem

A(2) := 5

read I, A(I)

X := A(2)

I > 2

Y := A(I)  Z := A(I)

Indeterminate subscript

constraints on subscript value due to path condition
**Escalating problem**

- Read I
- $X := A[I] \quad PV.X = \text{unknown}$
- $Y := X + Z \quad PV.Y = \text{unknown} + PV.Z$
  \[= \text{unknown} \]
Can often determine array element

\[
\begin{align*}
I &:= 0 \\
\text{if } I &\leq 3 \\
Y &:= A(I) \\
I &:= I + 1
\end{align*}
\]

. subscript value is constant
Symbolic Evaluation Approaches

- symbolic evaluation
  - With some enhancements
  - Data independent
  - Path dependent

- dynamic symbolic evaluation
  - Data dependent --> path dependent

- global symbolic evaluation
  - Data independent
  - Path independent
Dynamic Symbolic Execution

- Data dependent
- Provided information
  - Actual value:
    \[ X := 25.5 \]
  - Symbolic expression:
    \[ X := Y \times (A + 1.9) ; \]
  - Derived expression:
Dynamic Analysis combined with Symbolic Execution

- Actual output values
- Symbolic representations for each path executed
  - path domain
  - path computation
- Fault detection
  - data dependent
  - path dependent (if accuracy is available)
Dynamic Symbolic Execution

- **Advantages**
  - No path condition consistency determination
  - No path selection problem
  - No aliasing problem (e.g., array subscripts)

- **Disadvantages**
  - Test data selection (path selection) left to user
  - Fault detection is often data dependent

- **Applications**
  - Debugging
  - Symbolic representations used to support path and data selection
**Symbolic Evaluation Approaches**

- simple symbolic evaluation
- dynamic symbolic evaluation
- **global symbolic evaluation**
  - Data and path independent
  - Loop analysis technique classifies paths that differ only by loop iterations
  - Provides global symbolic representation for each class of paths
Global Symbolic Evaluation

• Loop Analysis
  • creates recurrence relations for variables and loop exit condition
  • solution is a closed form expression representing the loop
  • then, loop expression evaluated as a single node
Global Symbolic Evaluation

2 classes of paths:
\[ P_1: (s, (1,2), 4, (5, (6,7), 8), f) \]
\[ P_2: (s, 3, 4, (5, (6,7), 8), f) \]

global analysis

\[
\begin{align*}
\text{case} \\
D[P_1]: & \ C[P_1] \\
D[P_2]: & \ C[P_2]
\end{align*}
\]

Endcase

- analyze the loops first
- consider all partial paths up to a node
Loop analysis example

read A, B
Area := 0
X := A

X ≤ B
f
write AREA

AREA := AREA + A
X := X + 1
t
Loop Analysis Example

• Recurrence Relations
  \[ \text{AREA}_k = \text{AREA}_{k-1} + A_0 \]
  \[ X_k = X_{k-1} + 1 \]

• Loop Exit Condition
  \[ \text{lec}(k) = (X_k > B_0) \]
Loop Analysis Example (continued)

- solved recurrence relations
  \[ \text{AREA}(k) = \text{AREA}_0 + \sum_{i = x_0}^{x_0 + k - 1} \text{A}_0 \]
  \[ \text{X}(k) = X_0 + k \]

- solved loop exit condition
  \[ \text{lec}(k) = (X_0 + k > B_0) \]

- loop expression
  \[ k_e = \min \{k \mid X_0 + k > B_0 \text{ and } k \geq 0\} \]

\[ \text{AREA} : = \text{AREA}_0 + \sum_{i = x_0}^{x_0 + k_e - 1} \text{A}_0 \]
\[ \text{X} : = X_0 + k_e \]
• loop expression
  \[ k_e = \min \{k \mid X_0 + k > B_0 \text{ and } k \geq 0\} \]

\[
\begin{align*}
AREA & : = AREA_0 + X_0 + k_e \\
X & : = X_0 + k_e \\
\sum_{i=X_0}^{X_0+k_e-1} A_0 & \\
\end{align*}
\]

• global representation for input \((a,b)\)
  \[ X_0 = a, \ A_0=a, \ B_0 = b, \ AREA_0 = 0 \]
  \[ a + k_e > b \implies k_e > b - a \]
  \[ K_e = b - a +1 \]
  \[ X = a + (b-a+1) = b+1 \]
  \[ AREA = \sum_{i=a}^{b} a \]
  \[ \text{read } A,B \]
  \[ \text{AREA } :=0 \]
  \[ X := A \]
  \[ \text{write } \text{AREA} \]
Loop analysis example

1. read A, B
2. Area := 0
3. X := A
4. If X ≤ B then
   5. write AREA
   6. AREA := AREA + A
   7. X := X + 1
   8. Goto 4 else
   9. X := X + 1
Find path computation and path domain for all classes of paths

- $P1 = (1, 2, 3, 4, 7)$
- $D[P1] = a > b$
- $C[P1] = (\text{AREA}=0) \text{ and } (X=a)$
Find path computation and path domain for all classes of paths

• \( P2 = (1, 2, 3, 4, (5, 6), 7) \)
  • \( D[P2] = (b>a) \)
  • \( C[P2] = (\text{AREA} = (b-a+1) a) \)
  \( k_e = b - a + 1 \)
  \( X := b + 1 \)

\[ X_0 = a \]
\[ B_0 = b \]
\[ A_0 = a \]
\[ K_e = b - a + 1 \]
\[ X = b + 1 \]
\[ \text{AREA} = (b-a+1) a \]
Example

procedure RECTANGLE (A,B: in real; H: in real range -1.0 .. 1.0; 
F: in array [0..2] of real; AREA: out real; ERROR: out boolean) is 
    -- RECTANGLE approximates the area under the quadratic equation
    -- F[0] + F[1]*X + F[2]*X**2 From X=A to X=B in increments of H.
    X,Y: real;
begin
    --check for valid input
    if H > B - A then
        ERROR := true;
    else
        ERROR := false;
        X := A;
        AREA := F[0] + F[1]*X + F[2]*X**2;
        while X + H ≤ B loop
            X := X + H;
            Y := F[0] + F[1]*X + F[2]*X**2;
            AREA := AREA + Y;
        end loop;
        AREA := AREA*H;
    endif;
end RECTANGLE
H > B - A

ERROR := true;

ERROR := false;

X := A;

AREA := F[0] + F[1]*X + F[2]*X**2;

X + H ≤ B

X := X + H;

Y := F[0] + F[1]*X + F[2]*X**2;

AREA := AREA + Y;

AREA := AREA*H

f
## Symbolic Representation of Rectangle

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>(s,1,2,f)</td>
</tr>
<tr>
<td>D[P₁]</td>
<td>(a - b + h &gt; 0.0)</td>
</tr>
</tbody>
</table>
| C[P₁] | AREA = ?  
ERROR = true |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P₂</td>
<td>(s,1,3,4,5,6,10,f)</td>
</tr>
</tbody>
</table>
| D[P₂] | (a - b + h <= 0.0) and (a - b + h > 0.0)  
= = false  
*** infeasible path *** |
|   |   |
| P₃ | (s,1,3,4,5,6,7,8,9,10,11,f) |
| D[P₃] | (a-b+h <= 0.0) |
| C[P₃] | AREA = a*f[1]*h + 2.0*a*f[2]*h + f[0]*h  
+ sum < i :=1 ... int (-a/h+b/h) |  
(a*f[1]*h + a**2*f[2]*h  
+ 2.0*a*f[2]*h**2 + f[0]*h  
+ f[1]*h**2 + f[2]*h**3 + i**2) >  
ERROR = false |

```
D[P₁] (a - b + h > 0.0)
C[P₁] AREA = ?
ERROR = true

D[P₂] (a - b + h <= 0.0) and (a - b + h > 0.0)  
= = false  
*** infeasible path ***

D[P₃] (a-b+h <= 0.0)
C[P₃] AREA = a*f[1]*h + 2.0*a*f[2]*h + f[0]*h  
+ sum < i :=1 ... int (-a/h+b/h) |  
(a*f[1]*h + a**2*f[2]*h  
+ 2.0*a*f[2]*h**2 + f[0]*h  
+ f[1]*h**2 + f[2]*h**3 + i**2) >  
ERROR = false
```
Global Symbolic Evaluation

• Advantages
  • global representation of routine
  • no path selection problem

• Disadvantages
  • has all problems of
    • Symbolic Execution PLUS
      • inability to solve recurrence relations
        • (interdependencies, conditionals)

• Applications
  • has all applications of
    • Symbolic Execution plus
      • Verification
      • Program Optimization
Why hasn't symbolic evaluation become widely used?

- expensive to create representations
- expensive to reason about expressions
- imprecision of results
  - current computing power and better user interface capabilities may make it worth reconsidering
Partial Evaluation

• Similar to (Dynamic) Symbolic Evaluation
• Provide some of the input values
  • If input is $x$ and $y$, provide a value for $x$
• Create a representation that incorporates those values and that is equivalent to the original representation if it were given the same values as the preset values
  • $P(x, y) = P'(x', y)$
Partial Evaluator
Why is partial evaluation useful?

• In compilers
  • May create a faster representation
  • E.g., if you know the maximum size for a platform or domain, hardcode that into the system
  • More than just constant propagation
    • Do symbolic manipulations with the computations
Example with Ackermann’s function

- $A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } n = 0 \\ A(m-1, A(m,n-1)) & \text{else} \end{cases}$

- $A_0(n) = n+1$
- $A_1(n) = \begin{cases} A_0(1) & \text{if } n = 0 \\ A_0(A_1(n-1)) & \text{else} \end{cases}$
- $A_2(n) = \begin{cases} A_1(1) & \text{if } n = 0 \\ A_1(A_2(n-1)) & \text{else} \end{cases}$
Specialization using partial evaluation

\[ Y := A(I) \]
\[ Z := A(2) \]
\[ I > 2 \]
\[ \text{read } I, A(I) \]
\[ A(2) := 5 \]
\[ I = 2 \]
\[ I < 2 \]
\[ Z := \text{eval}(A(2)) \]
Why is Partial Evaluation Useful in Analysis

• Often can not reason about dynamic information
  • Instantiates a particular configuration of the system that is easier to reason about
  • E.g., the number of tasks in a concurrent system; the maximum size of a vector

• Look at several configurations and try to generalize results
  • Induction
  • Often done informally
Reference on Partial Evaluation