Pinaka: Symbolic Execution Meets Incremental Solving

Eti Chaudhary  Saurabh Joshi

Department of Computer Science and Engineering
IIT Hyderabad, India

TOOLympics, TACAS, 2019
```c
int x,y,output;

if(x>0){
    output = x;
}
else{
    if(y>0){
        output = y;
    }
    else{
        output = 0;
    }
}
assert(output>=0);
```
Underlying Approach: Partial Incremental Mode

int x, y, output;

if ( x > 0 ){
    output = x;
} else {
    if ( y > 0 ){
        output = y;
    } else {
        output = 0;
    }
}

assert(output ≥ 0);
Underlying Approach: Partial Incremental Mode

```c
int x, y, output;

if ( x > 0 ){
    output = x;
} else {
    if ( y > 0 ){
        output = y;
    } else{
        output = 0;
    }
}

assert(output ≥ 0);
```
```c
int x, y, output;

if (x > 0) {
    output = x;
} else {
    if (y > 0) {
        output = y;
    } else {
        output = 0;
    }
}

assert(output ≥ 0);
```

\[(x_1 > 0) \land (output_1 == x_1)\]
int x, y, output;

if (x > 0) {
    output = x;
} else {
    if (y > 0) {
        output = y;
    } else {
        output = 0;
    }
}

assert(output >= 0);

\[(x_1 > 0) \land (output_1 == x_1) \land \neg(output_1 \geq 0)\]
Underlying Approach: Partial Incremental Mode

```c
int x,y,output;
if ( x > 0 ){
  output = x;
} else {
  if ( y > 0 ){
    output = y;
  } else{
    output = 0;
  }
}
assert(output \geq 0);
```

Diagram:
```
    s1
     \   /\n      v v
     b1  \   /
          \ v
         s2
```

(True, False) \& (True, False)

\((x_1 \leq 0)\)
int x, y, output;

if (x > 0) {
    output = x;
} else { 
    if (y > 0) {
        output = y;
    } else {
        output = 0;
    }
}

assert(output >= 0);

\[(x_1 \leq 0) \land (y_1 > 0) \land (output_2 == y_1)\]
```c
int x, y, output;
if (x > 0) {
    output = x;
} else {
    if (y > 0) {
        output = y;
    } else {
        output = 0;
    }
}
assert(output ≥ 0);
```

\[(x_1 ≤ 0) \land (y_1 > 0) \land (output_2 == y_1) \land \lnot(output_2 ≥ 0)\]
Underlying Approach: Handling Loops
Non-Terminating Case

Suppose by the loop is reached, we have \( x_1 = 8 \) and \( y_1 = 5 \).

```
1   while ( x < 10 )
2   {
3       y = y + 1;
4       if ( y < 5 )
5           {
6               x = x + 1;
7           }
8   }
```
Suppose by the loop is reached, we have $x_1 = 8$ and $y_1 = 5$. 

```
while ( x < 10 )
{
    y = y+1;
    if ( y < 5 )
    {
        x = x+1;
    }
}
```
Underlying Approach: Handling Loops

Non-Terminating Case

Suppose by the loop is reached, we have $x_1 = 8$ and $y_1 = 5$. 
Suppose by the loop is reached, we have $x_1 = 8$ and $y_1 = 5$. 

```
1 while ( x < 10 )
2 {
3     y = y+1;
4     if ( y < 5 )
5         {
6             x = x+1;
7         }
8 }
```
Underlying Approach: Handling Loops

Non-Terminating Case

While loop: 
1. while ($x < 10$) 
2. 
3. $y = y + 1$; 
4. if ($y < 5$) 
5. 
6. $x = x + 1$; 
7. 
8. 

Suppose by the loop is reached, we have $x_1 = 8$ and $y_1 = 5$.

Graph representation:

- $x_1 < 10$ ($8 < 10$)
- $x_1 \geq 10$
- $y_2 = y_1 + 1$ ($y_2 = 6$)
- $y_2 < 5$
- $y_2 \geq 5$
- $x_1 < 10$
- $x_1 \geq 10$
### Verification Outcomes

<table>
<thead>
<tr>
<th>Safe Program</th>
<th>Unsafe Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminating</td>
<td>Safe</td>
</tr>
<tr>
<td></td>
<td>Unsafe</td>
</tr>
<tr>
<td>Non-Terminating</td>
<td>Won’t Terminate</td>
</tr>
</tbody>
</table>

**Note:**
- For a safe program, if Pinaka terminates, it also ensures that the program is terminating.
- If Pinaka reports a bug, it will indeed be a bug.
- For an unsafe non-terminating program, Pinaka might not terminate if it takes the non-terminating path *first* along the search.
SVCOMP Configuration: DFS + PI

<table>
<thead>
<tr>
<th>Category</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReachSafety-Arrays</td>
<td>4</td>
</tr>
<tr>
<td><strong>ReachSafety-Floats</strong></td>
<td>2</td>
</tr>
<tr>
<td>ReachSafety-Heaps</td>
<td>9</td>
</tr>
<tr>
<td>ReachSafety-ProductLines</td>
<td>9</td>
</tr>
<tr>
<td>ReachSafety-Recursive</td>
<td>8</td>
</tr>
<tr>
<td>ReachSafety-BitVectors</td>
<td>10</td>
</tr>
<tr>
<td>ReachSafety-ControlFlow</td>
<td>11</td>
</tr>
<tr>
<td>ReachSafety-ECA</td>
<td>10</td>
</tr>
<tr>
<td>ReachSafety-Loops</td>
<td>14</td>
</tr>
</tbody>
</table>
Thank You!

Available at:
https://github.com/sbjoshi/Pinaka

Developed at:
Indian Institute of Technology Hyderabad

Funded by:
SERB, DST, India