Symbolic Analysis

Xiangyu Zhang

What is Symbolic Analysis

- Static analysis considers all paths are feasible
- Dynamic considers one path or a number of paths
- Symbolic analysis reasons about path feasibility
  - Much more precise
  - Scalability is an issue
- A lot of applications
  - Input generation
  - Vulnerability detection/Fuzzing
  - Verification
  - Many many others

An Example

```
int buf[10];
x=input()
y=2*x-1
z=x+5
if (z-y>1 )
   if (x%2==0)
       buf[8+x]=…
   else
        p=1/(x-1);
```

Basic Idea

- Explore individual paths in the program; models the conditions and the symbolic values along a path to a symbolic constraint; a path is feasible if the corresponding constraint is satisfiable (SAT)
- Similar to our per-path static analysis, a worklist is used to maintain the paths being explored
- Upon a function invocation, the current worklist is pushed to a stack and a new worklist is initialized for path exploration within the callee
- Upon a return, the symbolic value of the return variable is passed back to the caller

Another Example

```
x=input()
if (x>0)
   y=…;
else
   y=…;
t= f (x)
if (t>0)
  z=y
```

```
f (k) {
  if (k<=-10)
        return  k+10;
  else
        return k;
```

Design Symbolic Analysis

- Abstract domain and transfer function
  - Symbolic expression
- It is a per-path analysis, hence no loss by path aggregation
- Is termination a problem?
  - Loop unrolling

A More Realistic Example

```
int readData(char type){
  int sum=0;
  if('F'==type){
     Scanner cin=new Scanner(FileReader("input"));
     while(cin.hasNext()){     // cin.hasNext()*
       sum += cin.nextInt();   // cin.nextInt() *
     }
     cin.close();
  }else{
     Socket s=new Socket("1.1.1.1");
     while(s.hasNext()){      // hasNext() *
        sum += s.nextInt();   // nextInt() *
     }
     s.close();
  }
  return sum;
}
int readAndNoti(){
    int type=readUserInput(); //readUserInput() *
    int s= readData(type);
    if( s>=0 )
          print("Zero or Positive");
     else
           print("negative");
    return s;
}
void main(){
  int  rawInput=readAndNoti();
  assert(rawInput>=0);
  ...
}
```

Constraints

$$
\begin{aligned}
C1&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0, \\
C2&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1, \\
C3&: \ 'F' = type \wedge x_1 \wedge x_2 \wedge \neg x_3 \wedge RET = y_1 + y_2, \\
C4&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0, \\
C5&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1, \\
C6&: \ 'F' \neq type \wedge w_1 \wedge w_2 \wedge \neg w_3 \wedge RET = z_1 + z_2.
\end{aligned}
$$

$$
\begin{aligned}
C7&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 < 0, \\
C8&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1 \wedge RET = RET2 \wedge RET2 < 0, \\
C9&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 \geq 0, \\
C10&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1 \wedge RET = RET2 \wedge RET2 \geq 0, \\
C11&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 < 0, \\
C12&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1 \wedge RET = RET2 \wedge RET2 < 0, \\
C13&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 \geq 0, \\
C14&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1 \wedge RET = RET2 \wedge RET2 \geq 0.
\end{aligned}
$$

Technical Challenges

- How to solve constraints
  - Propositional logic and SAT/SMT solving

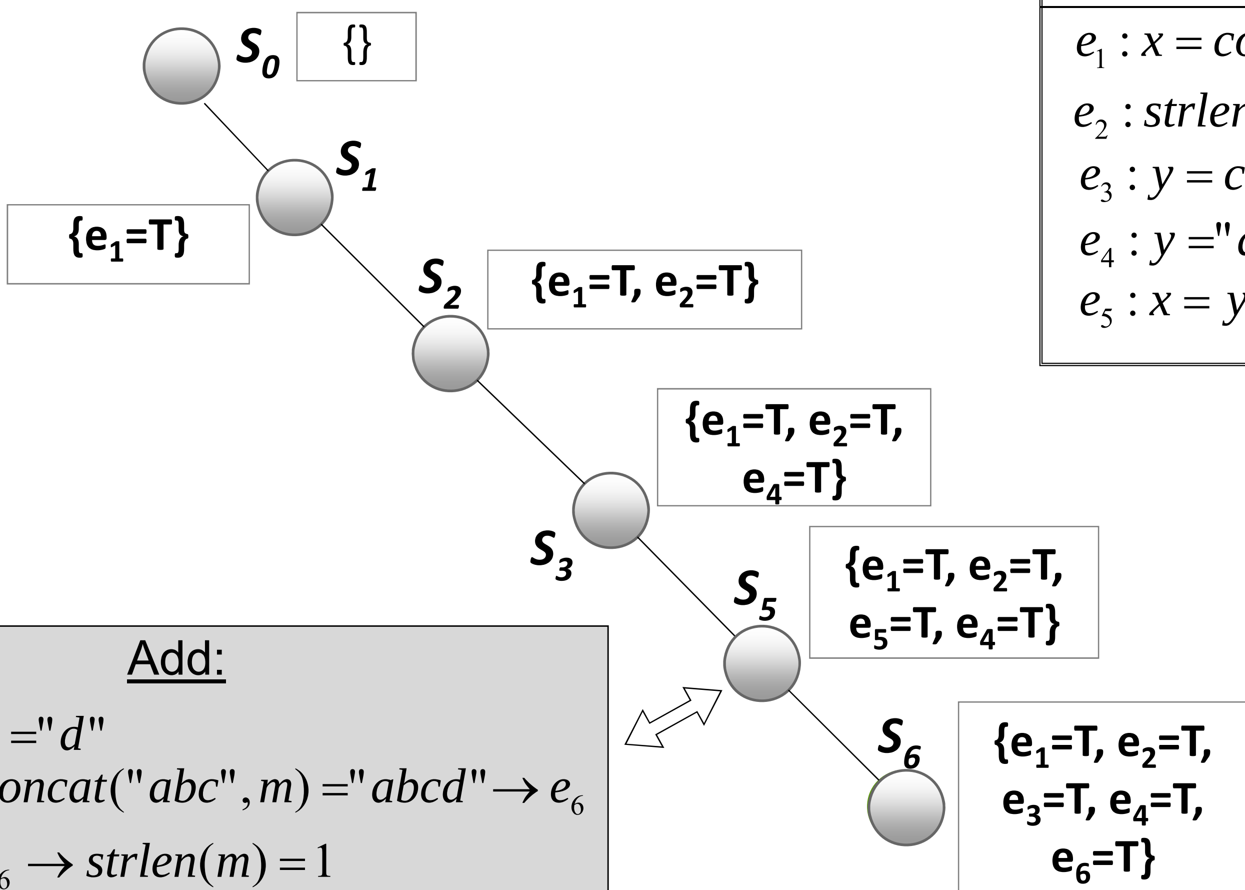
An Example of Symbolic Analysis and DPLL(T)

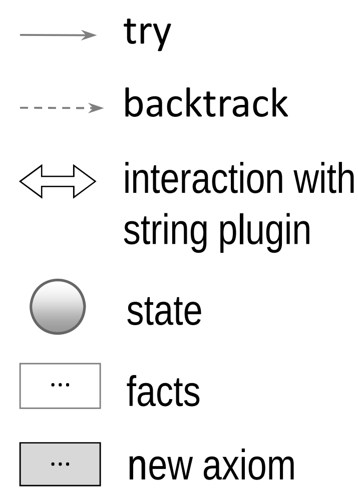
1. m=getstr();
2. n=getstr();
3. i=getint();
4. x=strcat("abc",m)
5. if (strlen(m)+i>5)
6. y="abcd"
7. else
8. y=strcat("efg",n);
9. if (x==y) …

$Path1 : assert(e_1 \wedge !e_2)$
$e_1 : x = concat("abc", m)$
$e_2 : strlen(m) + i > 5$
$e_3 : y = concat("efg", n)$
$e_4 : y = "abcd"$
$e_5 : x = y$

An Example of Symbolic Analysis and DPLL(T)

$Path\,2 : assert(e_1 \wedge e_2 \wedge e_4 \wedge e_5)$
$e_1 : x = concat("abc", m)$ $e_2 : strlen(m) + i > 5$ $e_3 : y = concat("efg", n)$ $e_4 : y = "abcd"$ $e_5 : x = y$





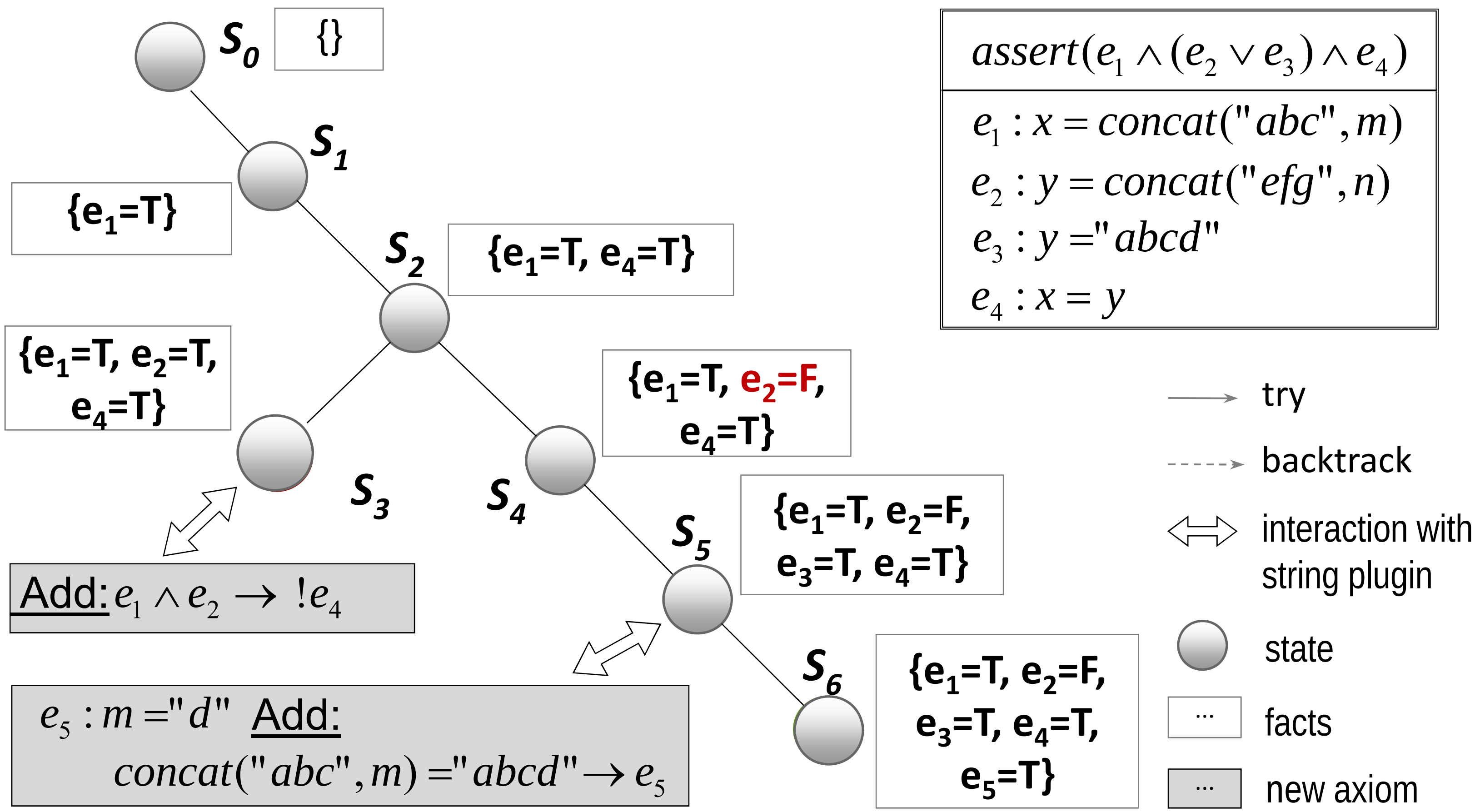
Add:

$e_6 : m = "d"$

$concat("abc", m) = "abcd" \rightarrow e_6$

$e_6 \rightarrow strlen(m) = 1$

An Example of DPLL(T)



Two Challenges/Extensions

- Path exploration
  - Depth first search + negating the last unvisited branch
- Difficult to solve constraints
  - Concolic execution
- Encode multiple paths

Concolic Execution

```
void test_me(int x,int y){
        z = x*x*x + 3*x*x + 9;
        if(z != y){
            printf("Good branch");
        } else {
            printf("Bad branch");
            abort();
        }
}
```

Initially it starts with a concrete input x=-3, y=7

Encoding Multiple Paths

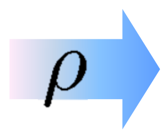
1. Convert statements into Single Static Assignment (SSA)
2. Convert SSA into equations
3. Unwind loops
4. SMT solving

SSA

Program

```
if (v)
  x = y;
else
  x = z;

w = x;
```



SSA Program

```
if (v_0)
  x_0 = y_0;
else
  x_1 = z_0;
x_2 = v_0 ? x_0 : x_1;
w_1 = x_2
```

For each join point, add new variables with selectors

```
int buf[10];
x=input()
y=2*x-1
z=x+5
if (z-y>1 )
   x=x+8;
buf[8]=…
```

Loop Unwinding

- **All loops are unwound**
  - **can use different unwinding bounds for different loops**
  - **to check whether unwinding is sufficient special "unwinding assertion" claims are added**

- **If a program satisfies all of its claims and all unwinding assertions then it is correct!**

- **Same for backward `goto` jumps and recursive functions**

Loop Unwinding

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
   ...
   if(cond) {
     Body;
     if(cond) {
       Body;
       while(cond) {
         Body;
       }
     }
   }
   Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Example: Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 2)
    j = j + 1;
  Remainder;
}
```

–unwind = 3

```
void f(...) {
  j = 1
  if(j <= 2) {
    j = j + 1;
    if(j <= 2) {
      j = j + 1;
      if(j <= 2) {
        j = j + 1;
        assert(!(j <= 2));
      }
    }
  }
  Remainder;
}
```

Symbolic Analysis of Neural Network Model


```
for (L=1 to 2) {
   for (i=0 to 1) {
      a[L,i]=0;
      for (j=0 to 1) {
         a[L,i]=a[L,i]+w[L,i,j]*a[L-1,j];
      }
      a[L,i]=a[L,i]+b[i];
      if (a[L,i]<0)
         a[L,i]=0;
    }
}
if (a[2,0]>a[2,1]) print (“class 0”);
else print (“class 1”)


w[0]= { {2,4}, {1,3}}   w[1]={ {3,2}, {7,1}}
b[0]=10                  b[1]=-20
```

Testing

Outline

- Black box testing
- White-box testing
  - Coverage
- Regression testing
- Fuzzing

Black-box Testing

- Given the function spec, analyze the NL doc, and come up with test cases
- Input + expected output
  - The oracle problem
- Equivalence class partitioning, boundary value analysis, and combinatorial testing

White-box Testing

- Statement coverage
- Edge coverage
- Path coverage
- Decision and condition coverage
- Mutation coverage

Note: the establishment of coverage: associating fault detection capabilities with coverage

Regression Testing

- Idea
  - When you find a bug,
  - Write a test that exhibits the bug,
  - And always run that test when the code changes,
  - So that the bug doesn't reappear
- Without regression testing, it is surprising how often old bugs reoccur
- Test selection, prioritization, repair

Fuzzing

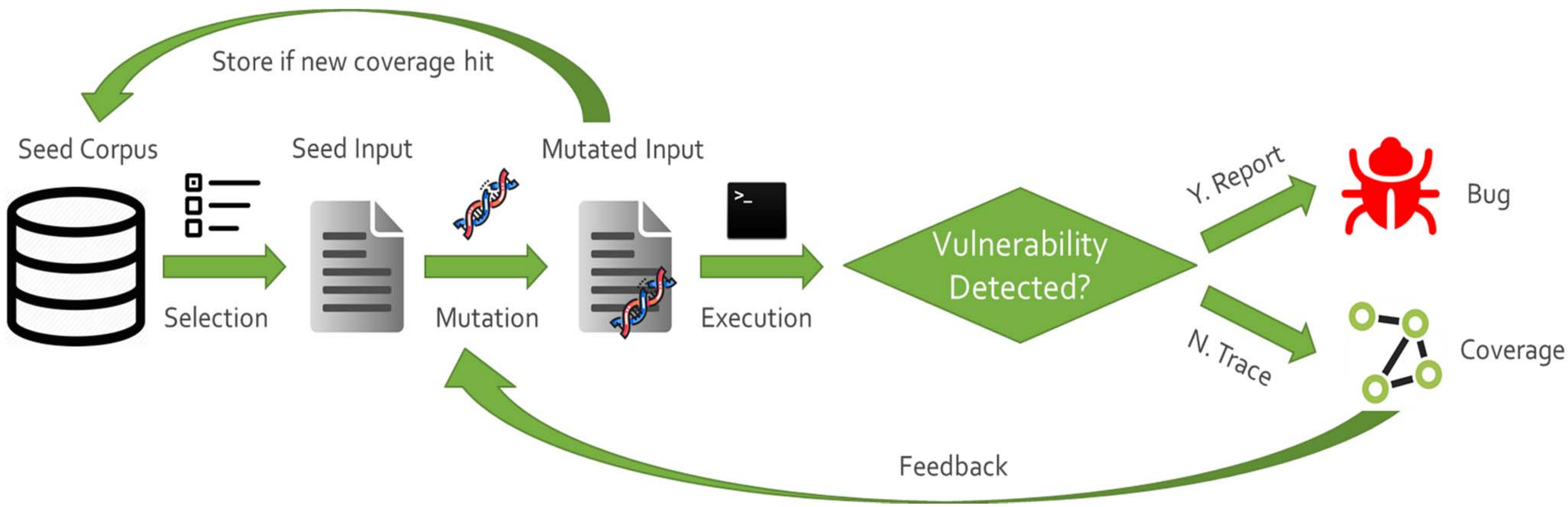
- Fuzzing is the most effective way for vulnerability discovery
  - most vulnerabilities are discovered by fuzzing
  - have academic and economic values

CVE-2014-0160

CVE-2015-3864

CVE-2017-2636

Fuzzing


Store if new coverage hit
Seed Corpus
Seed Input
Mutated Input
Selection
Mutation
Execution
Vulnerability Detected?
Y. Report
Bug
N. Trace
Coverage
Feedback

Fuzzing

- Neural model driven fuzzing
- Fuzzing DNN
- Gradient based fuzzing