Abstracting Symbolic Execution with String Analysis

Daryl Shannon
Sukant Hajra
Alison Lee
Daiqian Zhan
Sarfraz Khurshid

(The University of Texas at Austin)

Outline

- **Example**
- Background
- Approach
- Conclusion

Example

```
ResultSet getOrders(Statement stmt, String userName) {

   if (userName.contains("'")) {
         userName.replace("'", "''");
   }

   String sql = "SELECT * FROM orders "
              + "WHERE userName = '"
              + userName + "';";

   return stmt.executeQuery(sql);

}
```

Outline

- Example
- **Background**
- Approach
- Conclusion

“...symbolic execution for testing programs is a more exploitable technique in the short term than the more general one of program verification”

James King
CACM 19:7, 1976

Forward Symbolic Execution

technique for executing a program on **symbolic input values**

symbolic operations on program variables

explore program paths

- for each path, build a **path condition**
- check **satisfiability** of path condition

various applications

- test generation
- program verification

traditional use

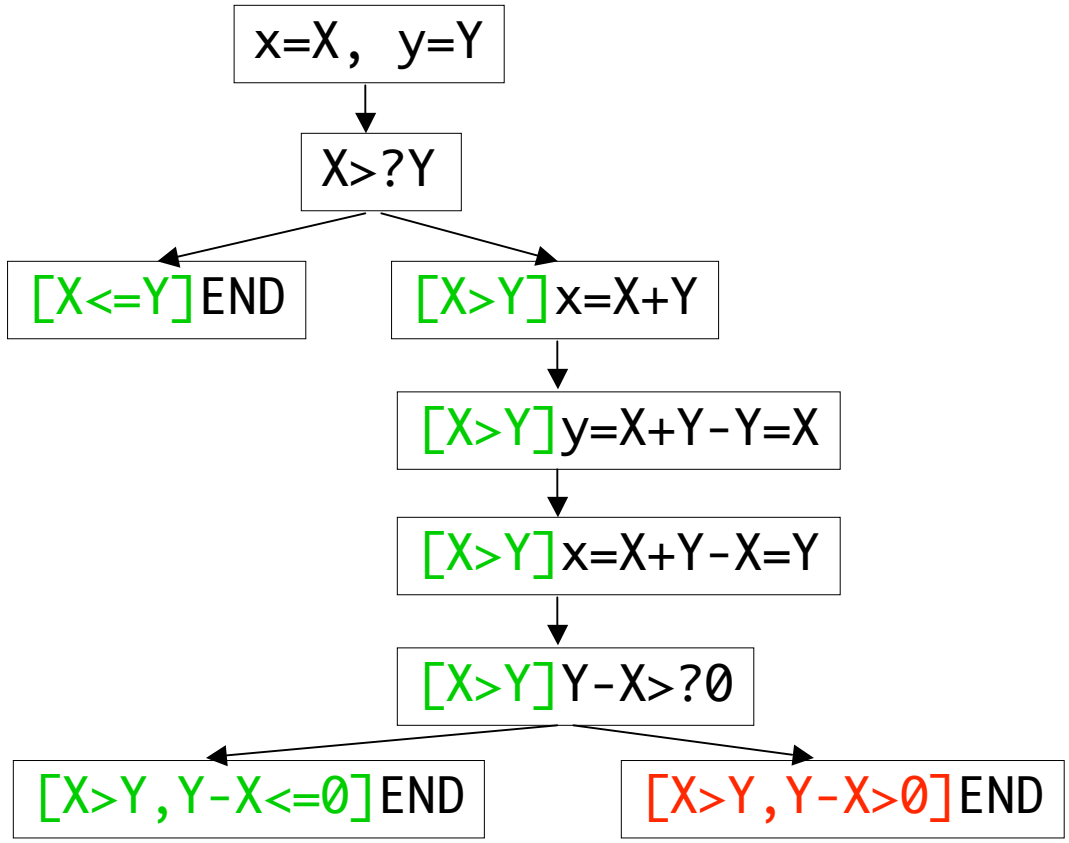
- programs with fixed number of int variables

Concrete Execution Path (example)

```
int x, y;                          x = 1, y = 0

if (x > y) {                       1 >? 0

   x = x + y;                      x = 1 + 0 = 1

   y = x - y;                      y = 1 - 0 = 1

   x = x - y;                      x = 1 - 1 = 0

   if (x - y > 0)                  0 - 1 >? 0

       assert(false);

}
```

Symbolic Execution Tree (example)

```
int x, y;

if (x > y) {

   x = x + y;

   y = x - y;

   x = x - y;

   if (x - y > 0)

       assert(false);

}
```



Symbolic References

object references are handled at the representation level

- references begin uninitialized
- on first-access non-deterministically initialize references to one of
  - null
  - an object created during a previous initialization
  - a new object

Abstracting Library Classes

problem with traditional symbolic execution: it does not scale

proposed solution: try not to perform it fully symbolically

- provide direct support for symbolic execution of certain (commonly used) classes
  - give semantics for symbolic manipulations of objects and solve constraints in ensuing path conditions
  - alleviate the need to symbolically execute intricate implementations of library code
    - prevent path conditions from becoming too complex and choking underlying solvers

Outline

- Example
- Background
- **Approach**
  - Solver
  - Checking Properties
  - Generating Test Cases
- Conclusion

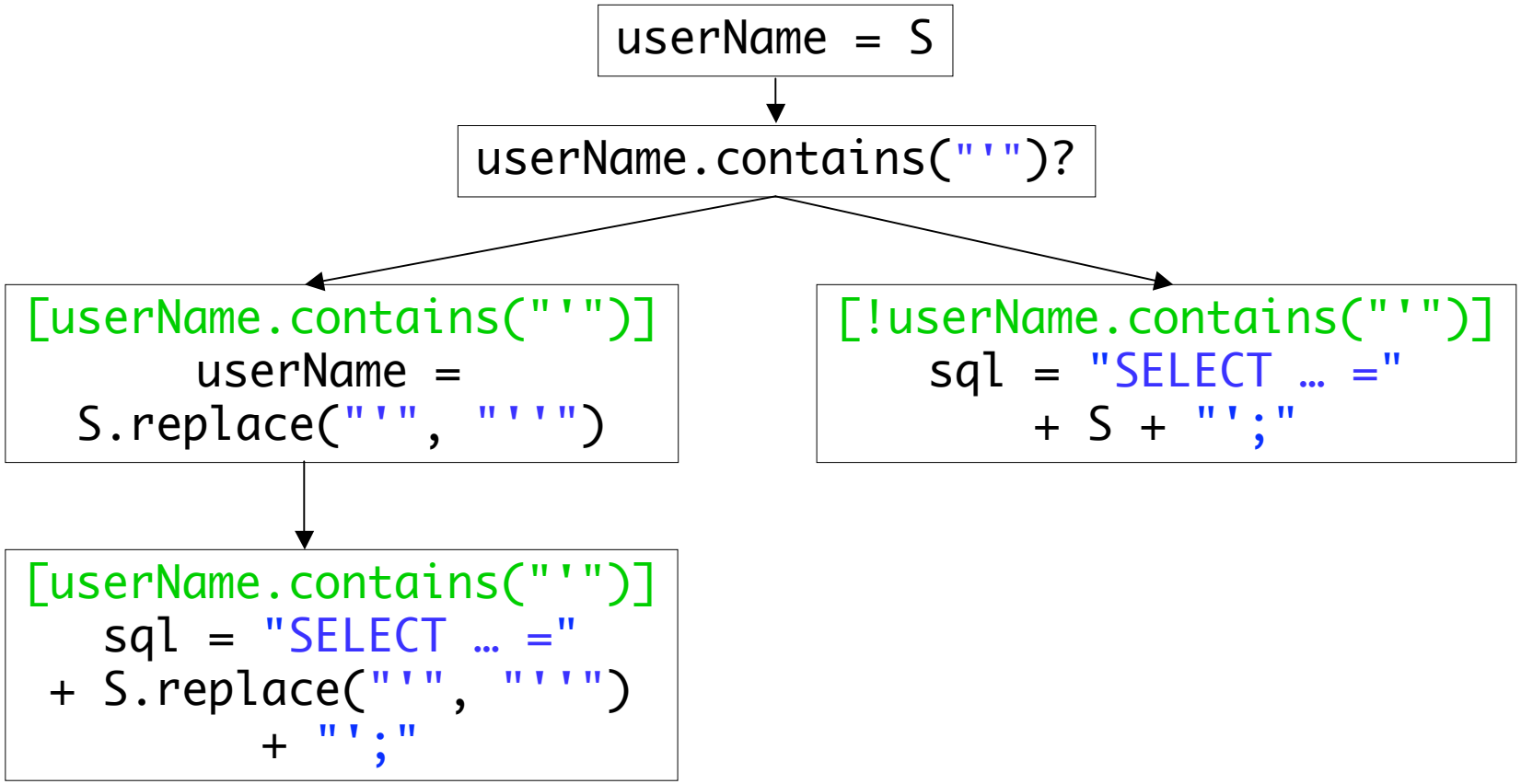
Solver Algorithm

- for each symbolic string value we have an automaton
  - the automata for input variables start out accepting all strings
  - automata for variables generated from string operations are based on the operands' automata
- for each constraint in the path condition we refine the automata to accept fewer strings
- if at any point an automaton accepts no strings then the path is infeasible

Example

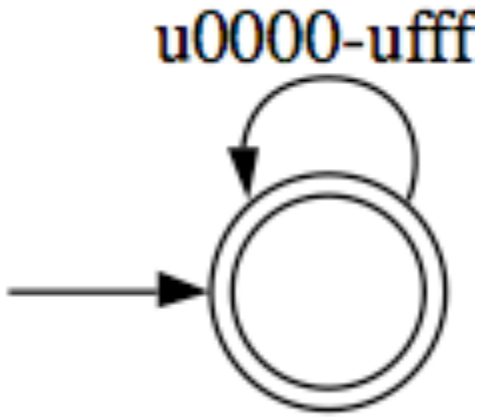
```
ResultSet getOrders(Statement stmt, String userName) {

   if (userName.contains("'")) {
         userName.replace("'", "''");
   }

   String sql = "SELECT * FROM orders "
              + "WHERE userName = '"
              + userName + "';";

   return stmt.executeQuery(sql);

}
```

Symbolic Execution Tree



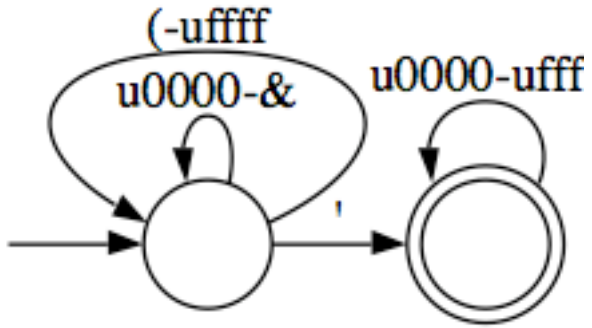
Input Variables

```
userName = S;
```

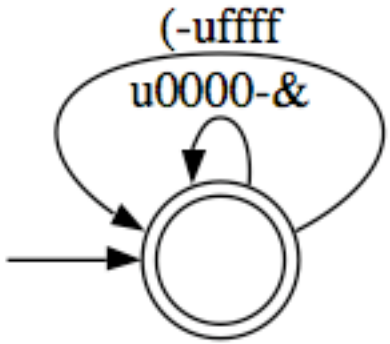


String Constraints

```
userName.contains("'")
```

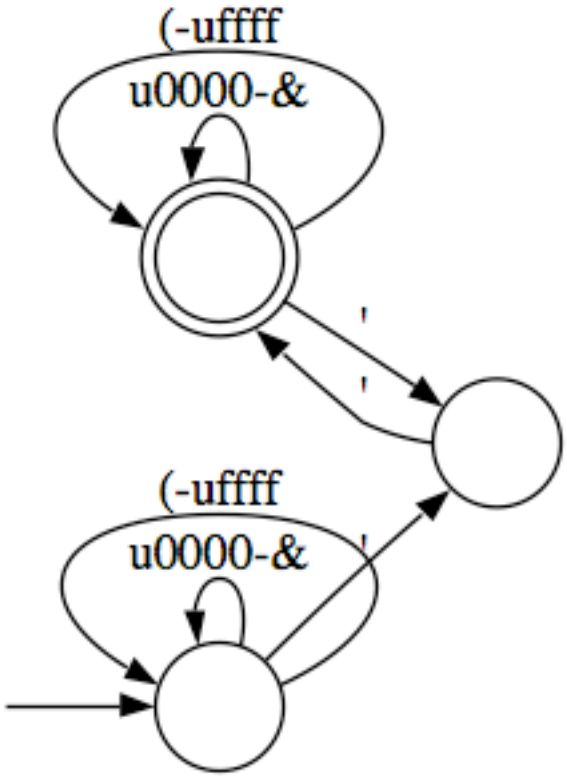



```
!userName.contains("'")
```



String Operations

```
userName = S.replace("'", "''");
```



Checking Properties

- if the path condition is feasible, then we can check properties on string values
- an automaton can represent a property (such as valid SQL grammar)
- if the strings accepted by the value's automaton is a subset of the strings accepted by the property's automaton then the property will always hold for that path

SQL Injection

```
[userName.contains("'")]

"SELECT * FROM orders WHERE userName = '"
        + S.replace("'", "''") + "';"
```

SQL Injection

```
userName
  \'; DROP TABLE x;--


SELECT * FROM orders WHERE userName
  = '\''; DROP TABLE x;--';
```

Generating Test Cases

any string accepted by an input variable's starting symbolic value's automaton can be use as a test case

Outline

- Example
- Background
- Approach
- **Conclusion**

Conclusions

- technique to abstract over higher level data types in symbolic execution
- implement abstraction of strings using finite state automata
  - allows checking string properties, such as conforming to valid SQL grammar

?

dshannon@ece.utexas.edu