Modern Symbolic Execution

Austin Cory Bart

CS-6304 Program Analysis

11/10/2015

Papers

- Cadar et al, "Symbolic Execution for Software Testing in Practice – a Preliminary Assessment", ICSE 2011.
- C. Cadar & K. Sen, "Symbolic Execution for Software Testing: Three Decades Later", CACM, Feb 2013, p 82-90

Cristian Cadar
PhD from Stanford
Now at Imperial College London
KLEE, EXE

Koushik Sen:
PhD from University of Illinois at Urbana-Champaign
Now at UC Berkley
DART, Latest, CUTE, jCUTE, Jalangi

Outline

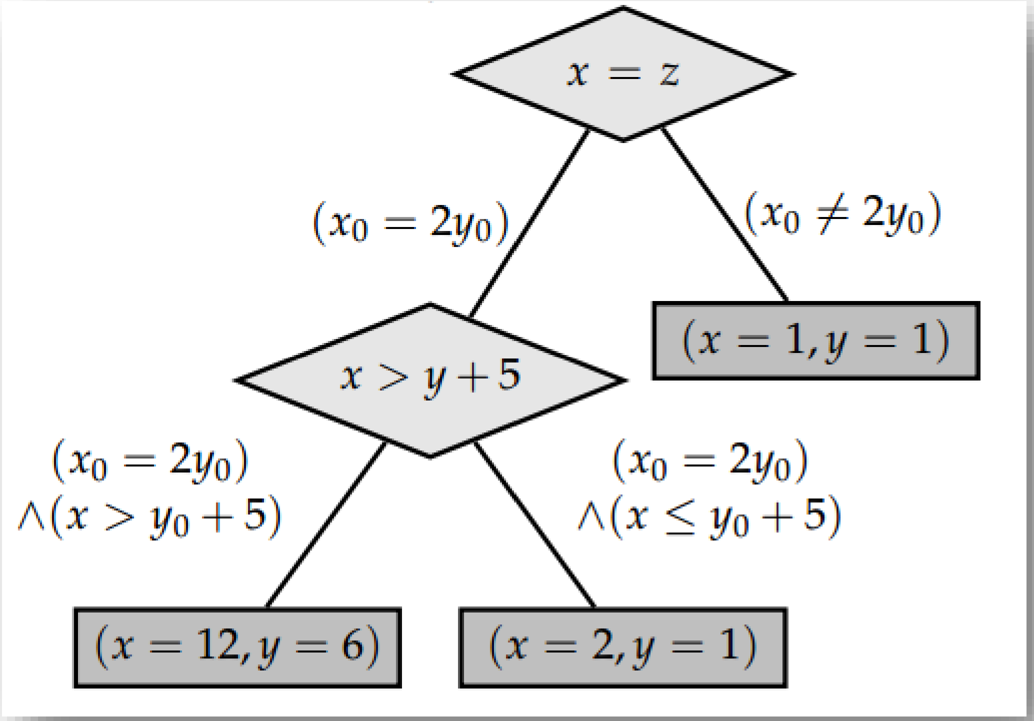
- Motivation
- Symbolic Execution Techniques
  - EGT
  - Concolic Testing
- Challenges
  - Path Explosion
  - Constraint Solving
  - Concurrency
- Tools

Why Care?

- Automatic Software testing
- Systems and above

Execution Tree

```
foo(int x,int y){
    z = 2*y;
    if (x == z){
        if (x > y + 5){
            //some error
        }
    }
}
```



Paqué, Daniel. "From Symbolic Execution to Concolic Testing." 2014
https://concurrency.cs.uni-kl.de/documents/Logics_Seminar_2014/SymbolicExecutionConcolicTesting.pdf

Concolic vs. EGT

Concolic

- Simultaneous Concrete and Symbolic
- Needs initial concrete values
- Multiple runs

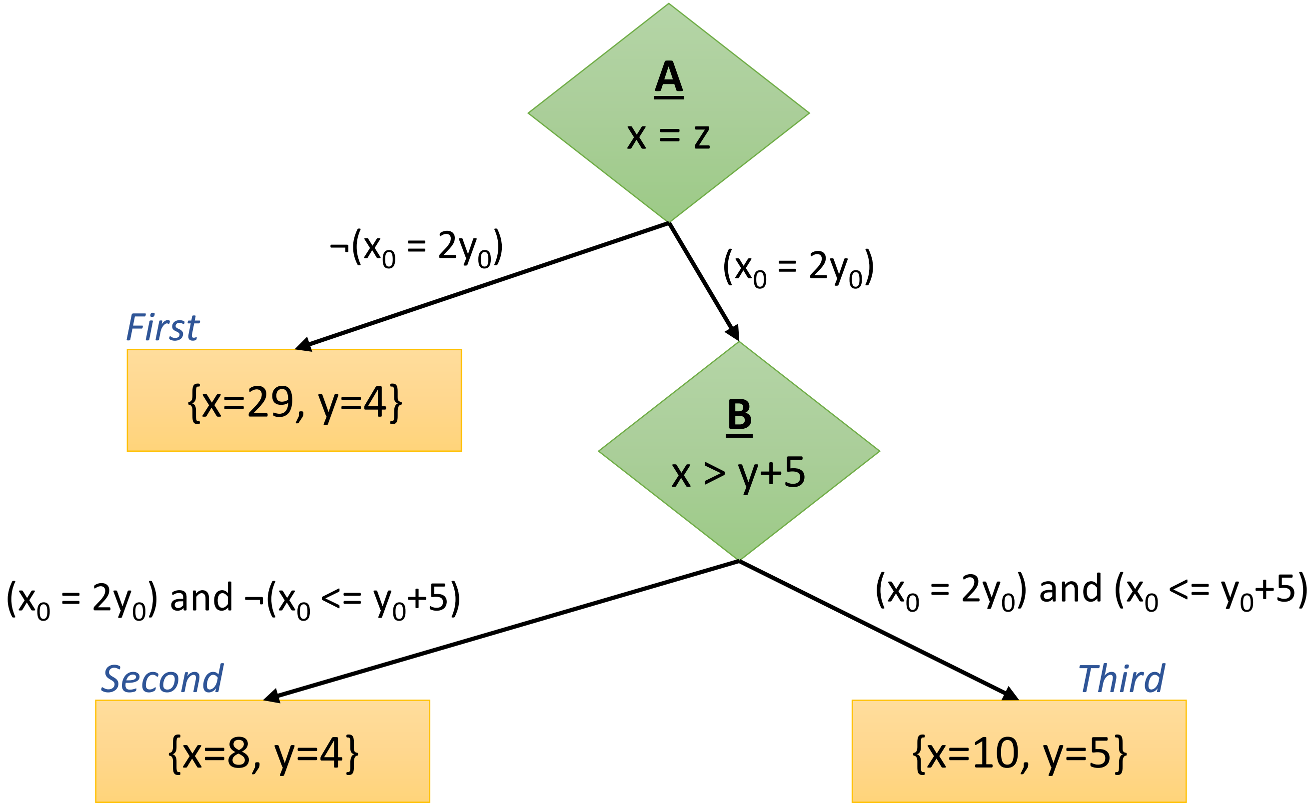
EGT

- Concrete values generated “on-demand”
- No initial concrete values
- Forking execution
- More similar to vanilla Symbolic Execution

Concolic Example #1

```
foo(int x,int y){
    z = 2*y;
    if (x == z){
        if (x > y + 5){
            //some error
        }
    }
}
```

- Initial random input: {x=29, y=4}
- First run, A[false]:
  - x_0 != $2y_0$
  - Negate conjunct, so x_0 == $2y_0$
  - New input: {x=8, y=4}
- Second Run, A[true] and B[false]:
  - (x_0 == $2y_0$) and (x_0 <= y_0+5)
  - Negate *new* conjunct, so (x_0 > y_0+5)
  - New input: {x=10, y=5}
- Third Run, A[true] and B[True]:
  - (x_0 == $2y_0$) and (x_0 > y_0+5)
  - All conjuncts tested. Complete.



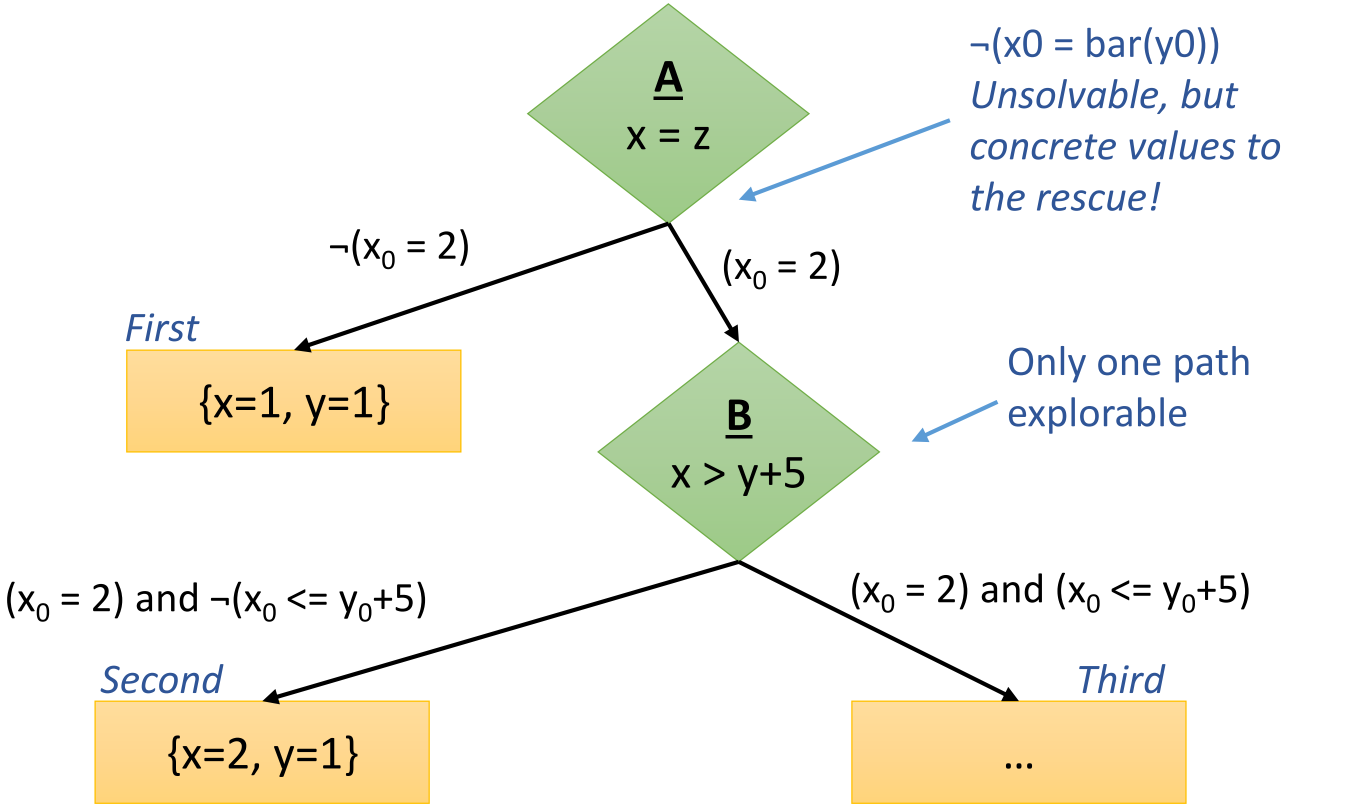
Concolic Example #2

```
foo(int x,int y){
    z = bar(y)
    if (x == z){
        if (x > y + 5){
            //some error
        }
    }
}
```

```
bar(int w){
    return 2*w;
}
```

(Executable, but source code not available)

- Initial random input: {x=1, y=1}
- First run, ***<u>Concrete Evaluation!</u>***, A[false]:
  - x_0 != 2
  - Negate conjunct, so x_0 == 2
  - New input: {x=2, y=1}
- …



EGT Example

```
foo3(int x){
  y = 2;
  z = 3*y;
  if (x == z){
    // ...
  } else {
    // ...
  }
}
```

concrete

concrete

Symbolic but simplified

Commonalities

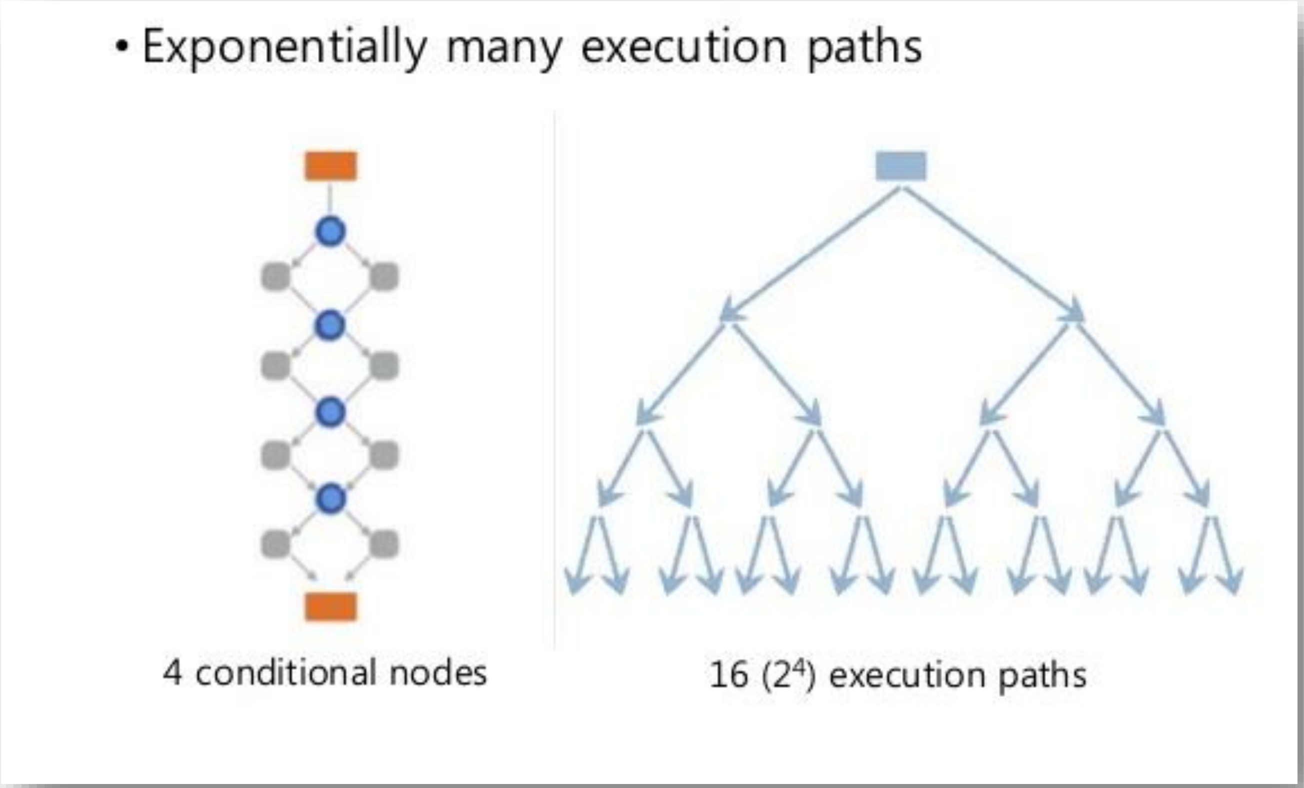
- Overcome
  - External code
  - Hardware imprecision (e.g., floating points)
  - Constraint Solver timeouts
- Sound, but not complete
- Automatic

Key Challenges

And their solutions

Problem 1: Path Explosion

```
void process(char input[3]) {
    int counter = 0;
    if (input[0] == 'a') counter++;
    if (input[1] == 'b') counter++;
    if (input[2] == 'c') counter++;
    if (counter >= 3) success();
    error();
}
```



Seo, Hyunmin, and Sunghun Kim. "How we get there: a context-guided search strategy in concolic testing." *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014.

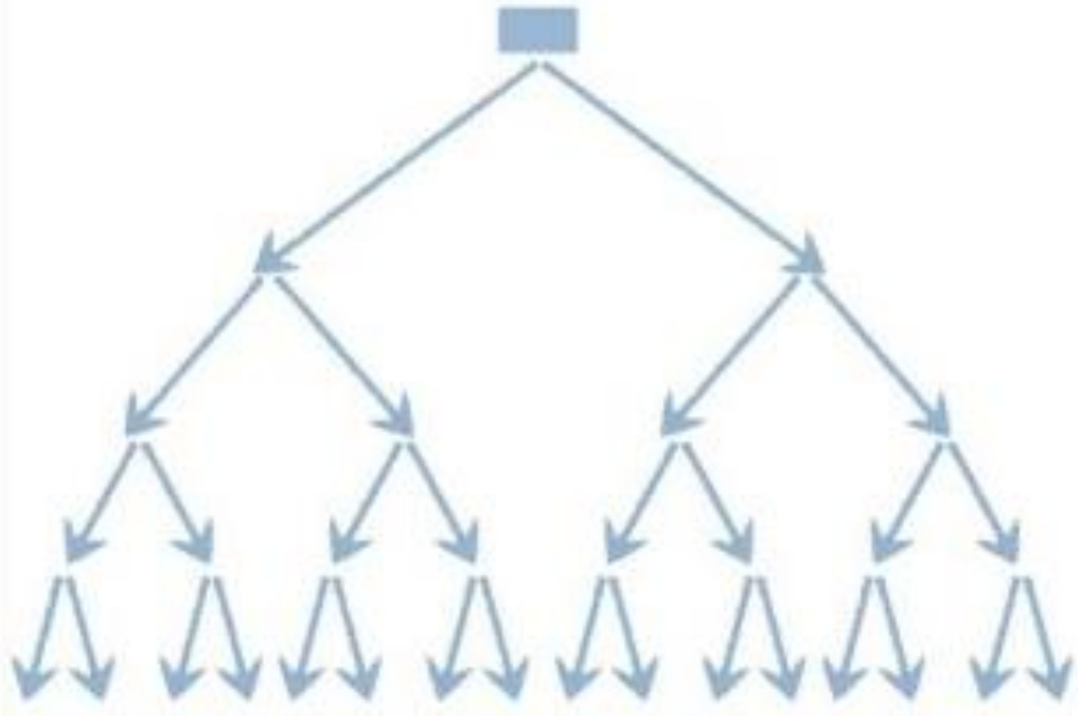
Infinite Execution Paths

```
void testme_inf () {
        int sum = 0;
        int N = sym_input();
        while (N > 0) {
            sum  = sum + N;
            N = sym_input();
        }
}
```

Figure 3. Simple example to illustrate infinite number of execution paths.

Solution 1.1: Heuristics

- Goal:
  - High statement coverage
  - High branch coverage
  - User-guided
- Examples
  - Distance (based on CFG)
  - Few Previous Runs
  - Randomness
  - Evolutionary search


16 (2^4) execution paths

Solution 1.2: Select Statements

- Merge If-conditions into Select statements
- Phi-node folding (if-conversion)
  - Static-single assignment (SSA)
  - Diamond-shaped if statements
  - Uncondtionally execute and select result
  - Side-effects can occur!
- Passes the buck to the Constraint Solver

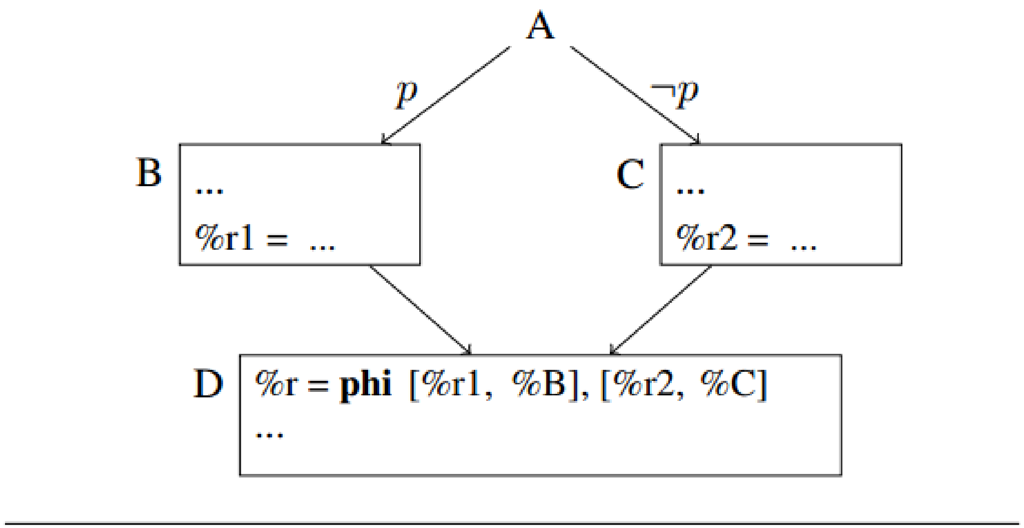


Figure 3. Diamond control flow pattern.

Collingbourne, Peter, Cristian Cadar, and Paul HJ Kelly. "Symbolic crosschecking of floating-point and SIMD code." *Proceedings of the sixth conference on Computer systems*. ACM, 2011.

Solutions 1.3, 1.4, 1.5: Other techniques

- Cache and reuse the analysis of lower-level functions
  - Pre-/post- condition summaries

- Lazy Test Generation
  - “The technique first explores, using dynamic symbolic execution, an abstraction of the function under test by replacing each called function with an unconstrained input.”
    - Strlen becomes a symbolic input that can represent any integer

- Prune redundant paths
  - Redundancy: same program path, same symbolic constraints

Problem 2: Constraint Solving

- NP Complete (although practical in practice)
- Dominates the runtime

Solution 2.1: Irrelevant Constraint Elimination

$$(x + y > 10) \wedge (z > 0) \wedge (y < 12) \wedge (z - x = 0)$$

Negated conjunct for new inputs

$$(x + y > 10) \wedge (z > 0) \wedge \neg(y < 12)$$

No relationship

$$(x + y > 10) \wedge \neg(y < 12)$$

Solution 2.2: Incremental Solving

- Cached constraint solutions

$$(x + y < 10) \wedge (x > 5) \Rightarrow \{x = 6, y = 3\}$$

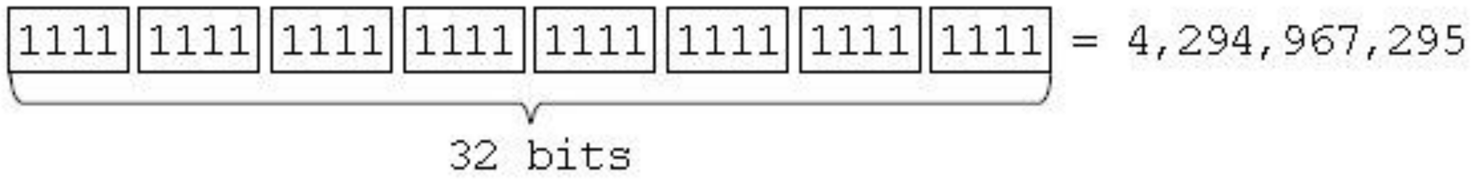
- Two situations:
  - Subset of a cached constraint: Easy, use the cached inputs!
  - Superset of a cached constraint: Test the inputs!

$$(x + y < 10) \wedge (x > 5) \wedge (y \geq 0)$$

- *"In practice, adding constraints often does not invalidate an existing solution"*

Problem 3: Memory Modelling

- 32-bit integer



- Pointers
  - b[7] vs. b[i] vs. a[b[i]]

Problem 4: Handling Concurrency

- Complex Data Inputs
- Distributed Systems
- GPGPU Programs

- Race conditions cause interleaving explosion

Solution 4.1: “Race Detection and Flipping Algorithm”

- Adaption of Concolic by Koushik Sen and Gul Agha
- Identify identical interleavings
  - Race conditions are collected during execution alongside path constraints
  - Race for two events if:
    - Stem from different threads
    - Both access the same memory location without locks
    - Order permutable by changing thread scheduling
  - Sequence of Triples: (thread, label, shared memory access type)
  - Types of race conditions: sequential, shared-memory access precedence, causal and race relation.
- Works by varying execution times
- Vector clocks (integer vectors) to record thread execution

Sen, Koushik, and Gul Agha. "A race-detection and flipping algorithm for automated testing of multi-threaded programs." *Hardware and Software, Verification and Testing*. Springer Berlin Heidelberg, 2007. 166-182.

Tool Rundown

- DART: First concolic testing (C)
- CUTE: Multi-threaded DART, dynamic data structures (C)
- jCUTE: CUTE for Java
- CREST: Concolic testing for experimenting with heuristics (C)
- EXE: EGT approach for Bit-level accuracy using STP (C)
- KLEE: Concurrent states, external data, heavily extended (LLVM)
- SAGE: Microsoft Windows, uses fuzzing (x86 binaries)
- PEX: Microsoft, focuses on more pure symbolic (.NET)

Conclusions

- Mixing symbolic and concrete is useful
- Many successes
- But still a lot left to do
  - Parallel
  - Constraint Solving
  - Memory models
  - Heuristics