

Design of a Modified Concolic Testing Algorithm with Smaller Constraints

Yavuz Koroglu Alper Sen

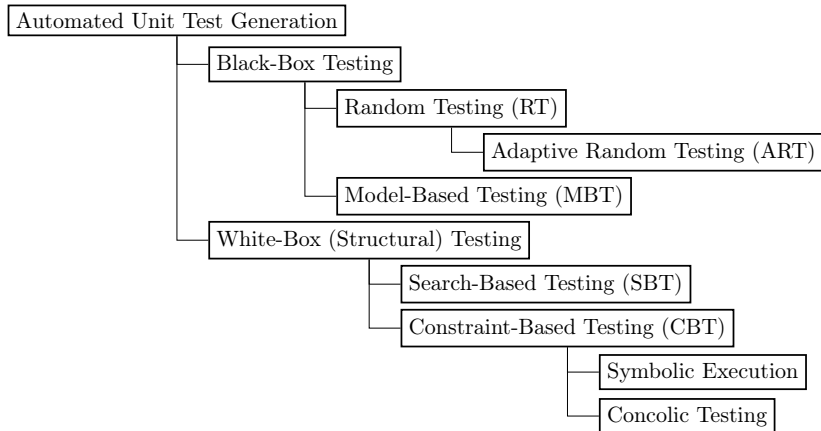
Department of Computer Engineering
Bogazici University, Turkey
yavuz.koroglu@boun.edu.tr
depend.cmpe.boun.edu.tr

International workshop on Constraints in Software Testing,
Verification and Analysis CSTVA@ISSTA 2016

Constraint-Based Testing (CBT)

Definition

If a testing technique uses a **constraint solver** to generate test cases, it is called **Constraint-Based Testing (CBT)**.



Constraint-Based Testing (Overview)

- The term coined in 1991 by Offut and DeMillo.
- **Symbolic Execution** (dates back to 1975),
 - Considered **impractical**, lack of powerful constraint solvers.
- **Revival** in the last two decades,
 - Availability of powerful constraint solvers (Yices, Z3 etc.),
 - **Concolic testing** is proposed.
- **Constraint solving bottleneck**,
 - **Scalability** issues.
 - Constraint solving optimizations (Concolic Unit Testing Engine (CUTE) offers **three** optimizations).
 - Did not completely solve the issue.

Our Motivation

What did we aim?

Design a modification on the current constraint solving methodology which

- **Decreases the burden** on the constraint solver,
- Still gets the **same coverage** as the previous CBT approaches and
- Allows **new heuristics and optimizations** to be implemented.

Our Motivation

What did we see?

CBT approaches make **few large queries** to the constraint solver.

- Instead, make **thousands of small queries**.
- In model checking domain, IC3 uses this strategy.
(SAT-Based Model Checking Without Unrolling, Aaron R. Bradley, VMCAI2011)
- Can we better utilize constraint solvers in CBT?

Concolic Testing

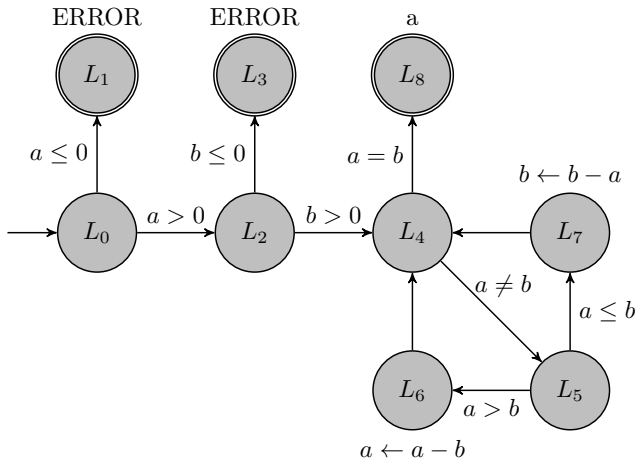
- Also called **Dynamic Symbolic Execution (DSE)**.
- Combines **concrete** and **symbolic** execution.
- The idea dates back to **2005** (CUTE and DART).
- We implement our approach on top of Concolic Testing.

Example: Greatest Common Divisor (GCD)

```
1  int gcd(int a, int b) {  
2      if (a <= 0) {          // L0  
3          return ERROR;    // L1  
4      }  
5      if (b <= 0) {          // L2  
6          return ERROR;    // L3  
7      }  
8      while (a != b) {       // L4  
9          if (a > b) {       // L5  
10             a = a - b;    // L6  
11         } else {  
12             b = b - a;    // L7  
13         }  
14     }  
15     return a;              // L8  
16 }
```

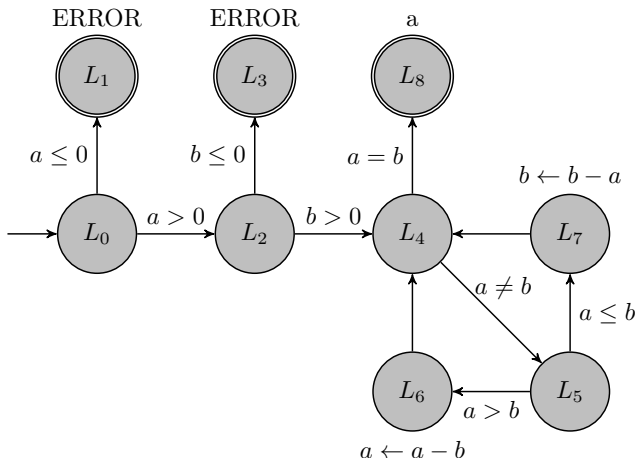
Test GCD using Concolic Testing

1) Generate random inputs: let $a = 4$, $b = 0$.



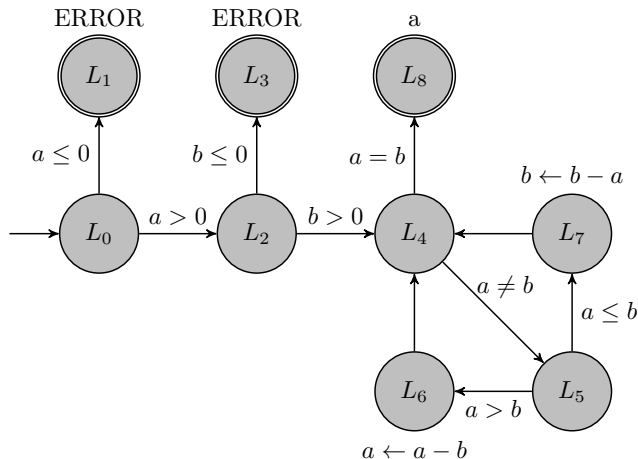
Test GCD using Concolic Testing

2) $\text{gcd}(4,0)$ traverses the following execution path: $L_0 \rightarrow L_2 \rightarrow L_3$.



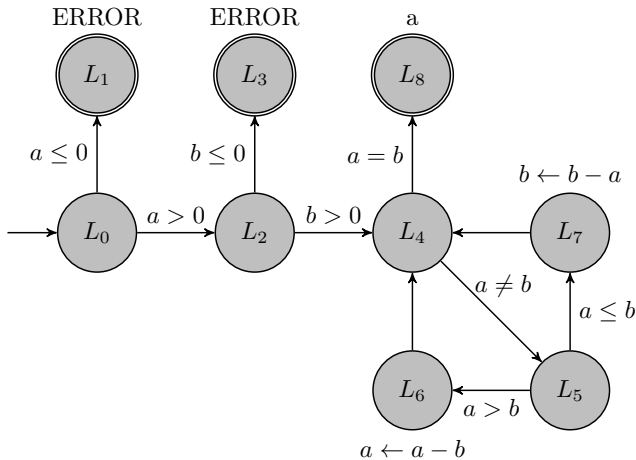
Test GCD using Concolic Testing

3) Gather $\pi_0 = (a > 0) \wedge (b \leq 0)$ during execution.



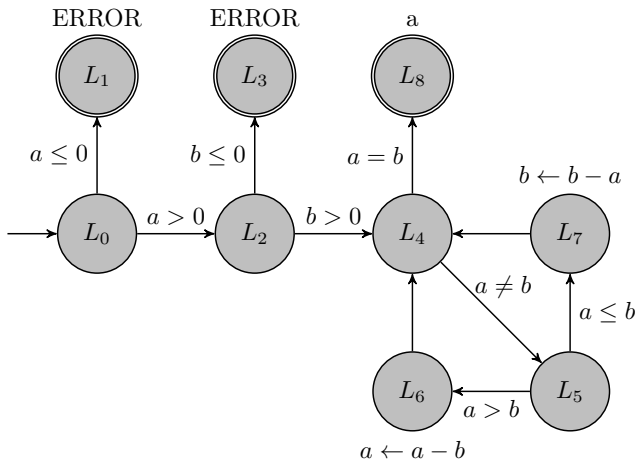
Test GCD using Concolic Testing

4) π_0 is a **full path constraint**.



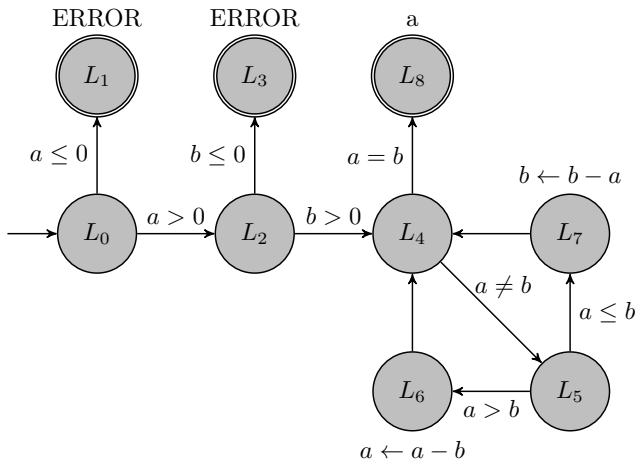
Test GCD using Concolic Testing

5) Full path constraint is the conjunction of all path conditions on an execution path.



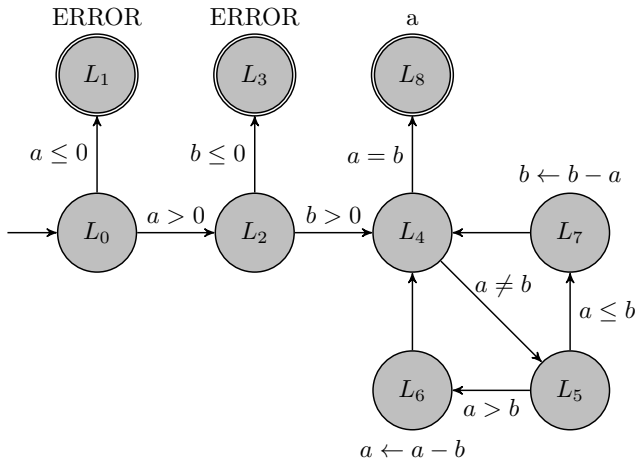
Test GCD using Concolic Testing

6) Generate $\phi_1 = (a > 0) \wedge (b > 0)$.



Test GCD using Concolic Testing

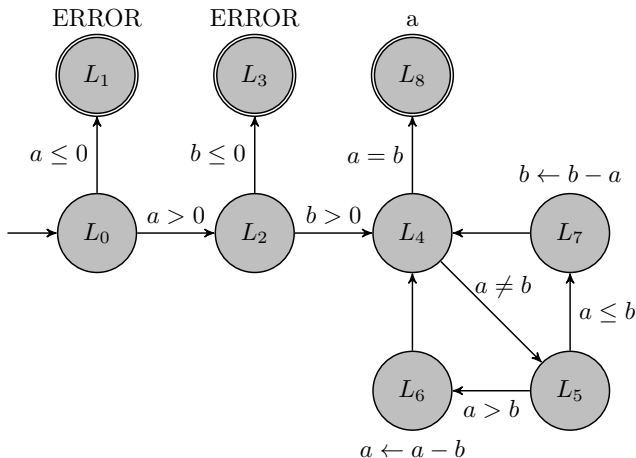
7) Let $CS(\phi_1)$ be $a = 4$ and $b = 6$.



Test GCD using Concolic Testing

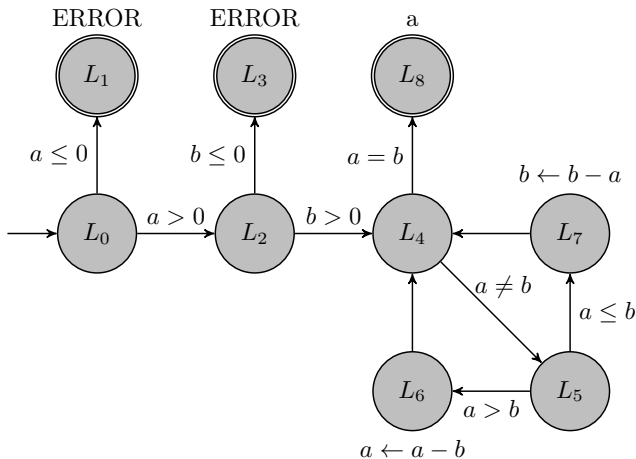
8) $\text{gcd}(4,6)$ traverses

$L_0 \rightarrow L_2 \rightarrow L_4 \rightarrow L_5 \rightarrow L_7 \rightarrow L_4 \rightarrow L_5 \rightarrow L_6 \rightarrow L_4 \rightarrow L_8$.



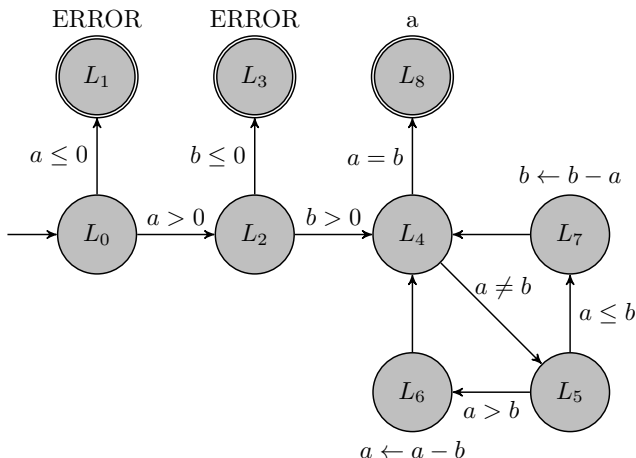
Test GCD using Concolic Testing

9) Gather $\pi_1 = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] = b - a)$.



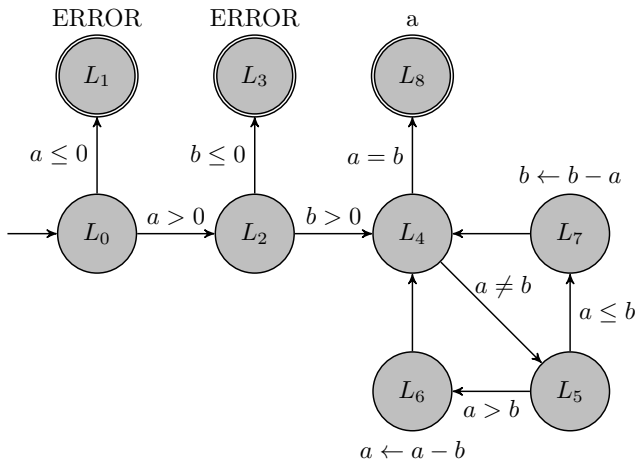
Test GCD using Concolic Testing

10) Solved only a small constraint (ϕ_1) to get an input which satisfies a large constraint (π_1).



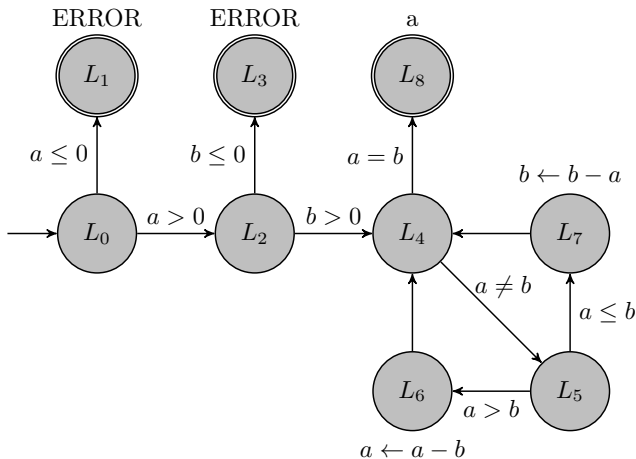
Test GCD using Concolic Testing

11) After a few iterations constraints get **very large**.



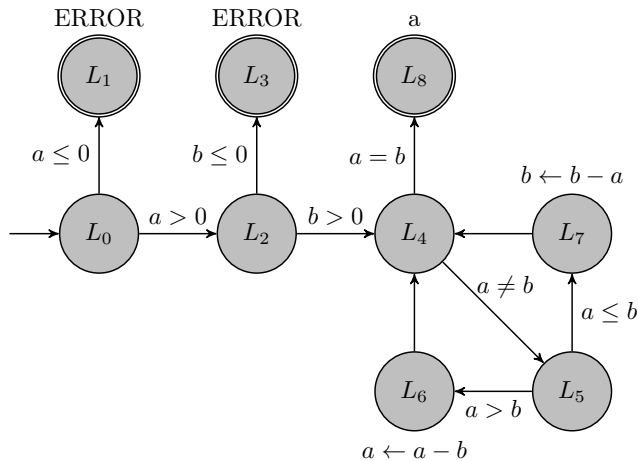
Test GCD using Concolic Testing

12) Generate $\phi_2 = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$.



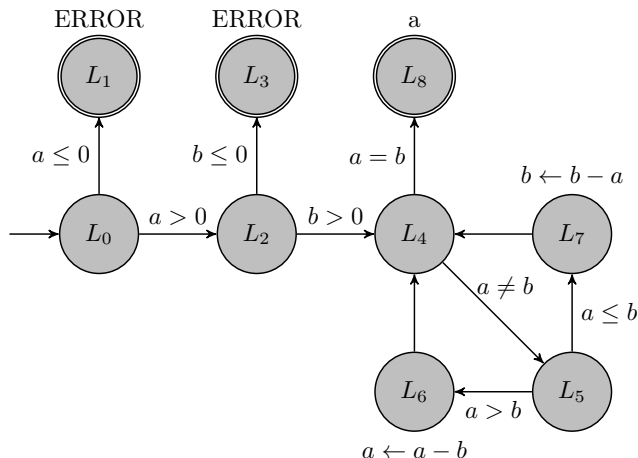
Test GCD using Concolic Testing

13) Let $\text{CS}(\phi_2) = a = 5, b = 6$.



Test GCD using Concolic Testing

14) $\text{gcd}(5,6)$ traverses $L_0 \rightarrow L_2 \rightarrow L_4 \rightarrow L_5 \rightarrow L_7 \rightarrow L_4 \rightarrow L_5 \rightarrow L_6 \rightarrow L_4 \rightarrow L_5 \rightarrow L_6 \rightarrow L_4 \rightarrow \dots$



Previous Constraint Solving Optimizations

- One of the first concolic testers, CUTE,
 - Proposes **three** optimizations for **constraint solving**:
 - 1) Fast Unsatisfiability Check.
 - 2) Common Sub-Constraints Elimination.
 - 3) Incremental Solving.

(OPT1) Fast Unsatisfiability Check

Main Idea

- Check if a path condition is **syntactically the negation** of any preceding ones in the full path constraint.
e.g. $\pi = \dots \wedge (a = b) \wedge \dots \wedge (a \neq b) \wedge \dots$
- If it is, the full path constraint is decided to be **infeasible** without solving.

OPT1,

- Reduces the number of constraint solver queries by 60-95% in general.
- Reduction in the GCD example: 0%.

(OPT2) Common Sub-Constraints Elimination

Main Idea

- Identify and eliminate common sub-constraints.

OPT2,

- Reduces common sub-constraints by 64-90% in general.
- Reduction in the GCD example: 0%.

(OPT3) Incremental Solving

Main Idea

- Remember that $\pi_0 = (a > 0) \wedge (b \leq 0)$ and $\phi_1 = (a > 0) \wedge (b > 0)$ from the GCD example.
- π_0 and ϕ_1 only differ by one condition.
- Let the conjunction of all conditions on ϕ_1 that depend on $(b > 0)$ be $\phi_1' = (b > 0)$.
- Let the solver fix a to its previous value and find a solution for ϕ_1' instead of ϕ_1 .

OPT3,

- On average, $|\phi'| \approx |\phi|/8$ in general.
- On the GCD example: No significant improvement.

Partial Path Constraints (ϕ)

Definition

Any **overapproximation** of the Full Path Constraint π is called a Partial Path Constraint (ϕ).

Example

- Let $\pi = (a > 0) \wedge (b \leq 0)$.
- Then, the possible partial path constraints are
 - $\phi_0 = T$,
 - $\phi_1 = (a > 0)$,
 - $\phi_2 = (b \leq 0)$ and
 - $\phi_3 = (a > 0) \wedge (b \leq 0)$.

Motivation of Partial Path Constraints

- There are **subsumed** path conditions.
- 1 In the GCD example, ϕ_2 contains both $p = (a \neq b - a)$ and $q = (a > b - a)$.
- 2 Trivially, $q \rightarrow p$.
- 3 So, p is **redundant**.
- 4 We should **eliminate** redundant path conditions.

Partial Path Constraints Cont'd

- Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a = b)$.
- Let $\phi = (a = b)$.
- Probability of $\text{CS}(\phi)$ also satisfies π is 0.25.
- For $\phi' = (b > 0) \wedge (a = b)$, probability becomes 0.50.

Danger!

- Usage of partial path constraints may cause **path divergence**.
- Therefore, some feasible execution paths may not get executed (**incompleteness**).

Incremental Partial Path Constraints (IPPC)

Main Idea

- **Same** as concolic testing.
- We **replace** the constraint solver call with IPPC.
- IPPC tries a **small** partial path constraint.
- Learns **larger** ϕ and tries again until the answer is found.

Incremental Partial Path Constraints (IPPC)

Algorithm

- 1 Start from a partial path constraint ϕ where $\pi \rightarrow \phi$.
- 2 Generate test input i that satisfy ϕ .
- 3 If ϕ is infeasible, then π must be infeasible.
- 4 Else if i satisfies π , return i .
- 5 Find out the first path condition c_d which i does not satisfy.
- 6 Let $\phi \leftarrow \phi \wedge c_d$.
- 7 Goto 2.

- c_d is called the **Cause of Divergence**.
- Steps 5-6-7 occurs only if generated i causes a **path divergence**.

Determining Initial ϕ : Most Basic Strategy

Motivation

We negate **only one condition** on the previously satisfied full path constraint.

Approach

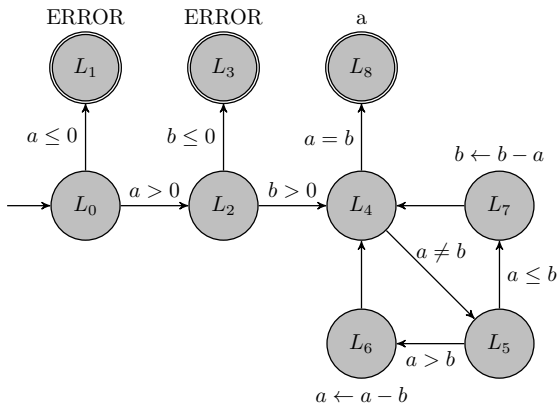
Take the **negated condition** as the initial ϕ .

Advantage

Incremental Solving optimization (OPT3) has more chance to satisfy π by fixing some of the inputs.

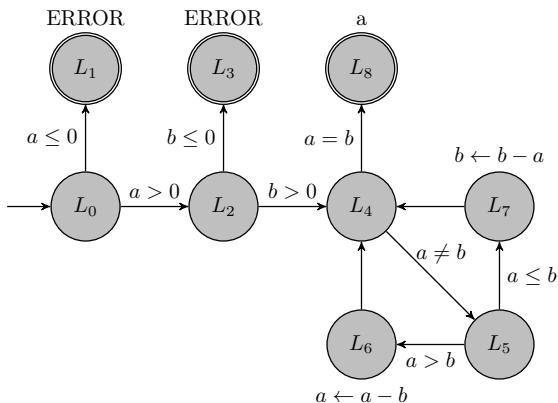
Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).



Example Returned: GCD

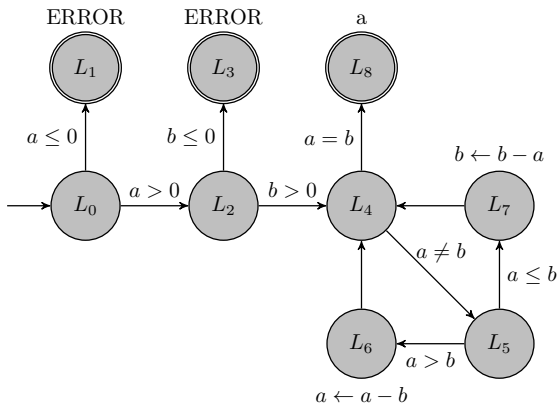
- 1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).
- 2) Let us solve this constraint using IPPC instead of a CS call.



Example Returned: GCD

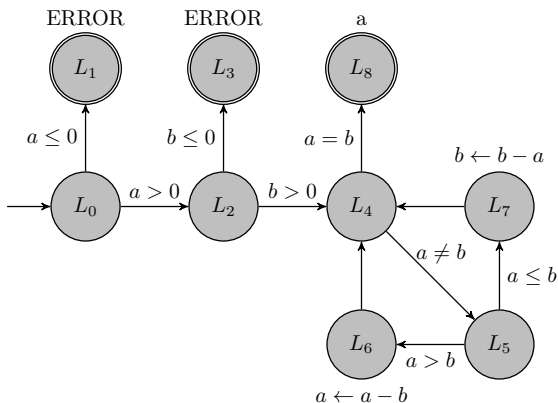
1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

3) $\phi^1 = (a - [b - a] \neq b - a)$.



Example Returned: GCD

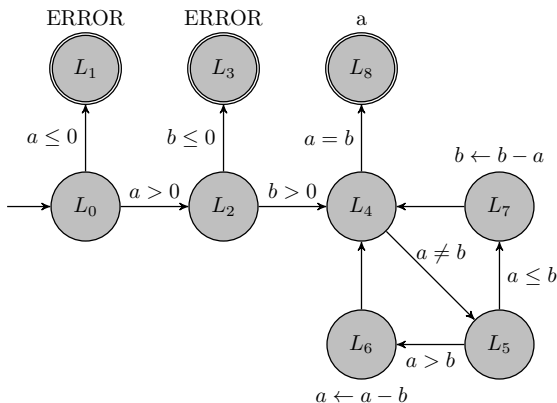
- 1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).
- 4) Yices in incremental mode generates $a = 2, b = 3$ for $\text{CS}(\phi^1)$.



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

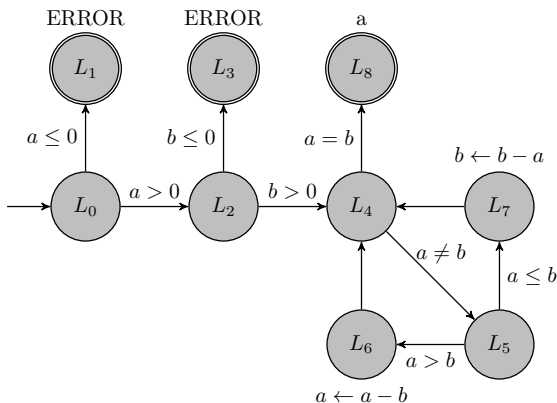
5) $(a, b) = (2, 3)$ does **NOT** satisfy π due to $c_d^1 = (a \neq b - a)$.



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

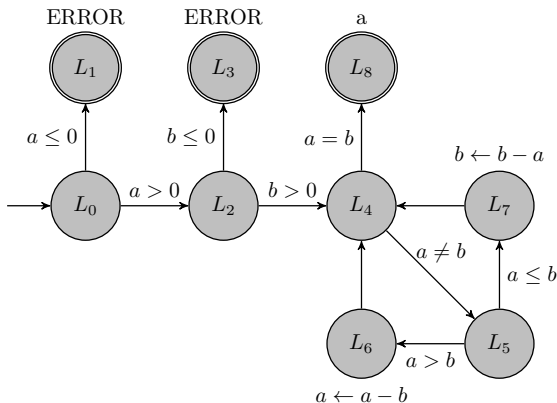
6) $\phi^2 = \phi^1 \wedge c_d^1 = (a - [b - a] \neq b - a) \wedge (a \neq b - a)$.



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

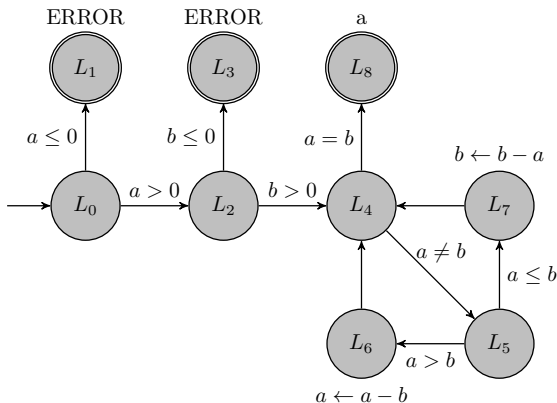
7) Yices generates $a = 4, b = 6$ for $\text{CS}(\phi^2)$ which satisfies the π .



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

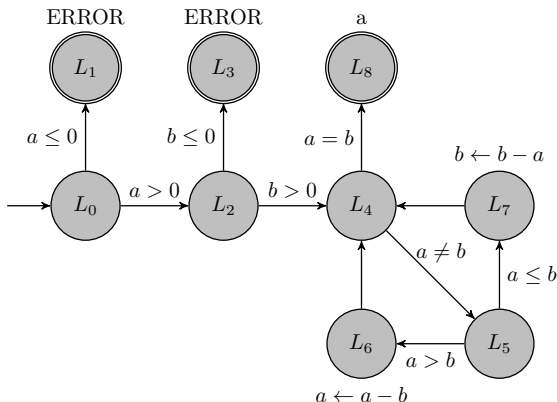
8) Standard concolic tester solves 1 path constraint of size 7.



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

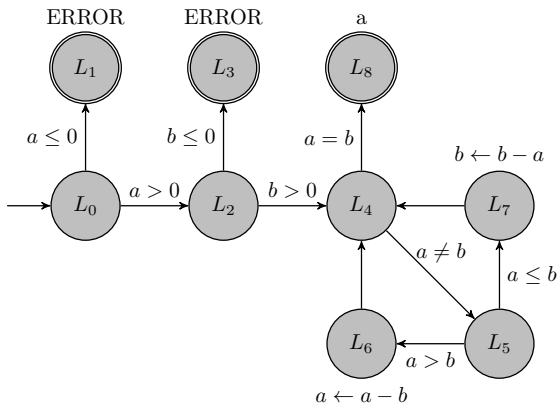
9) IPPC solves 2 path constraints of sizes 1 and 2.



Example Returned: GCD

1) Consider $\pi = (a > 0) \wedge (b > 0) \wedge (a \neq b) \wedge (a \leq b) \wedge (a \neq b - a) \wedge (a > b - a) \wedge (a - [b - a] \neq b - a)$ (ϕ_2 of the previous example).

10) More smaller queries vs. Few larger queries



Experimental Environment

The Environment

- Virtual Linux guest with 1024MB memory and one CPU,
- MacBook Pro host with an Intel Core i7 2.9 GHz GPU and 8GB Memory.

The Framework

CREST, is a known concolic testing framework developed by J. Burnim.

- **Source code** available.
- It uses **Yices**.
- It implements **different** concolic testing **strategies**.

Benchmarks

List of benchmarks used in the experiments are as follows:

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

Benchmarks

All conditions are guaranteed to be correctly solvable by Yices.

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

Benchmarks

Benchmarks are in different sizes.

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

Benchmarks

We made 10 executions for each configuration.

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

Benchmarks

We measured branch coverage via a script which uses gcov.

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

Benchmarks

Standard Concolic Testing and IPPC achieves the same coverage in N iterations.

UUT	KLOC	#vars
gcd	0.05	2
bsort	0.05	30
sqrt	0.06	1
prime	0.1	1
factor	0.2	1
replace	0.5	20
ptokens	0.6	40
grep	15	10

IPPC Speedup over Standard Concolic Testing (DFS)

IPPC has smaller constraints by a factor of 60.

UUT	Avg Const. Size Ratio (DFS / IPPC)	Speedup (t_{DFS}/t_{IPPC})
replace	4.4	0.6x
bsort	20.8	0.79x
sqrt	21.3	1.25x
grep	31.8	0.83x
ptokens	48.4	1.7x
gcd	97.5	2.77x
prime	115.6	9.1x
factor	137.3	9.8x
avg	59.6	3.35x

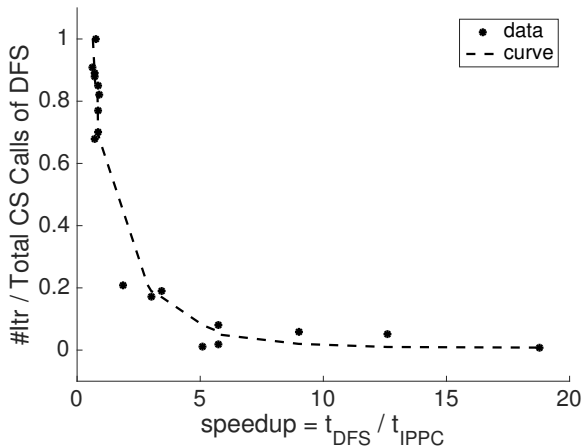
IPPC Speedup over Standard Concolic Testing (DFS)

IPPC has a speedup of 3.35 on average.

UUT	Avg Const. Size Ratio (DFS / IPPC)	Speedup (t_{DFS}/t_{IPPC})
replace	4.4	0.6x
bsort	20.8	0.79x
sqrt	21.3	1.25x
grep	31.8	0.83x
ptokens	48.4	1.7x
gcd	97.5	2.77x
prime	115.6	9.1x
factor	137.3	9.8x
avg	59.6	3.35x

Relationship btw. Infeasible Constraints and Speedup

IPPC has better speedup when the UUT has more infeasibilities.



Conclusion

In this work, we designed a modification which,

- Eliminates the need for solving large constraints,
 - Largest path constraint sizes found during the experiments:
 - 1 IPPC: 157
 - 2 DFS: 2922
- Works better if the UUT has many infeasible paths and
- Is **flexible**.

We also,

- Gave motivational examples and background for our work.
- Strongly suggested a relationship between speedup and infeasibility.
- Did experiments on the benchmarks.

Caching

- KLEE utilizes caching as a performance improving optimization.
- We use partial path constraints,
 - Therefore we can have **both-way** caching:
 - Inputs have a corresponding full path constraint (input $\rightarrow$ path constraint, reduces UUT execution)
 - Full path constraints are mapped to inputs. (path constraint $\rightarrow$ input, reduces CS execution)

Future Work

Independent Path Conditions as the Initial ϕ

- Using a greedy algorithm, find a set of independent path conditions.
- Conjoin all the independent conditions to get the Initial ϕ .
- Maybe we can decrease the total CS calls if we use this initial ϕ .

Future Work

Implementation on Different Frameworks and More Benchmarks

- We should find more benchmarks (currently there are 8 benchmarks),
- We should implement IPPC on top of different CBT approaches,

Thank You. Any Questions?