

# Software Systems Verification and Validation

Vescan Andreea, PHD, Assoc. Prof.

---



Faculty of Mathematics and Computer Science  
Babeș-Bolyai University



2022-2023





# Software Systems Verification and Validation

---

"Tell me and I forget, teach me and I may remember, involve me and I learn."

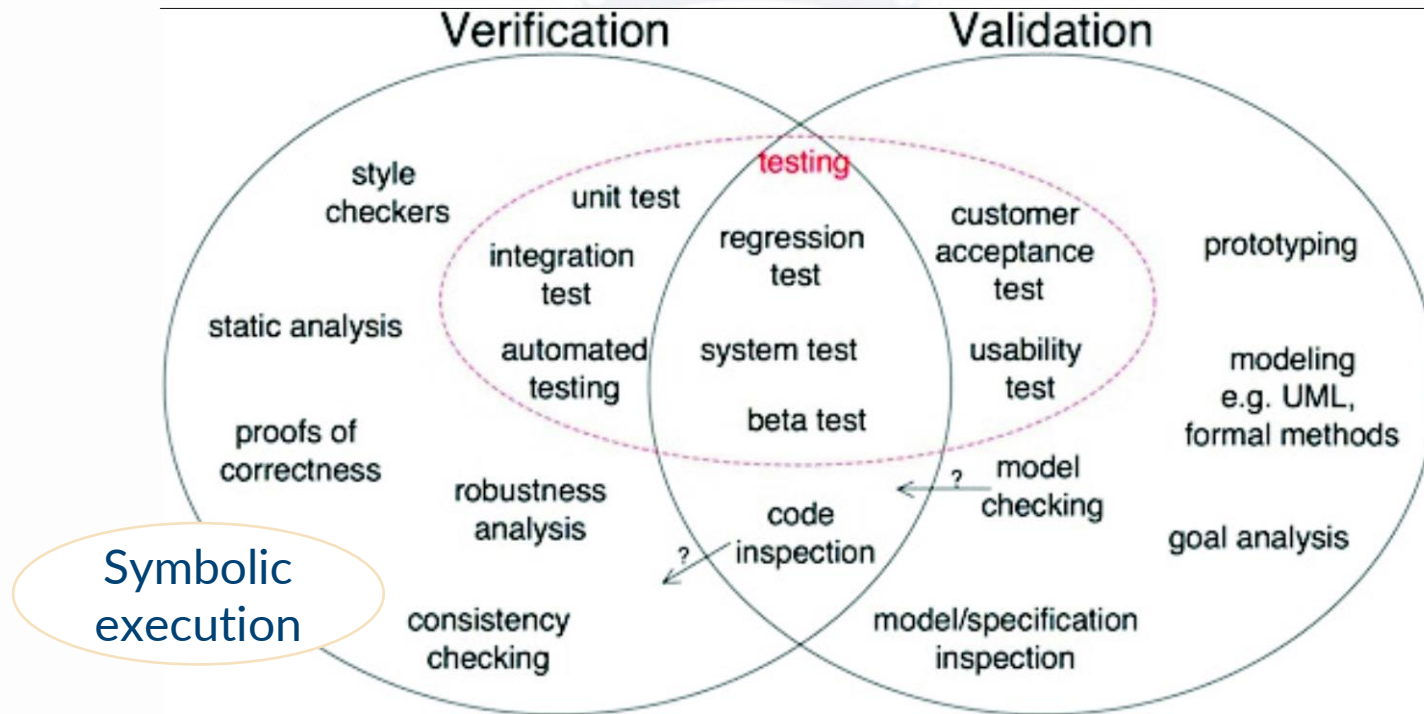
(Benjamin Franklin)

# (Next)/Today Lecture

- Symbolic execution
- Model checking



# What we will learn!



- <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/>



# Outline

- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
  - Sequential structure execution
  - Alternative structure execution
  - Repetitive structure execution
- Symbolic Execution Tree
  - Symbolic Execution Tree
  - Properties
- Questions
- Next lecture (still today)
  - Model checking

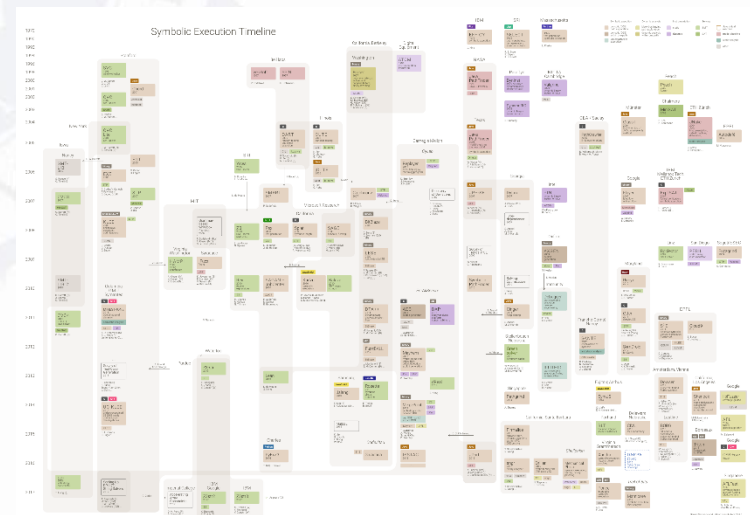
# Static analysis

## Symbolic execution

- Bugs that are missed by testing: rare features, rare circumstances, nondeterminism.
  - ➔ Static analysis
    - Can analyze all possible runs of a program
    - But can it find deep, difficult bugs?
      - Abstraction let us model all possible runs
      - Static analysis abstraction  $\leftrightarrow$  developer abstraction
- Testing works
  - reported bugs are real bugs, but each test only explores **one** possible execution.  
`assert (f(5)==6)`
  - We **hope** test cases generalize, but no guarantees!
  - ➔ Symbolic execution **generalizes** testing
    - ➔  $y=\alpha$ , `assert(f(y)==2*y+1)`
- **Remarks:**
  - symbolic execution is not meant to inspect the quality of the code.
    - static analysis deals with issues of path feasibility,
    - dynamic analysis tends to deal with path coverage.
  - Symbolic analysis is sort of in between and deals with state space explosion by logically forking the analysis at branches and solving for a set of satisfiable constraints.

# Symbolic execution - research

- 1976 - King [Kin76], Clarke [Cla76]
- ....
- 2005 - Microsoft: DART [God05]
- 2006 - Univ. Stanford: EXE, Univ. Illinois: CUTE and jCUTE [SA06]
- ...
- 2008 - KLEE (Stanford) [CDE08]
- ...
- 1999 - 2016 - NASA: Symbolic (Java) Path Finder [PV09], [CS13]
  - <http://javapathfinder.sourceforge.net/>
  - <http://babelfish.arc.nasa.gov/trac/jpf>
- Modern Symbolic Execution Techniques
  - mix concrete and symbolic execution
  - Concolic testing (DART – Directed Automated Random Testing)
  - EGT (Execution-Generated Testing)
- 2017 - Learn&Fuzz: Machine Learning for Input Fuzzing
  - <https://patricegodefroid.github.io/>
- 2018
  - Chopped Symbolic Execution (ICSE) (2006 - EXE)
  - Shadow Symbolic Execution for Testing Software Patches
  - <https://www.doc.ic.ac.uk/~cristic/>
- 2018 - Deep Reinforcement Fuzzing, Konstantin Böttinger, Patrice Godefroid, Rishabh Singh
- 2022 – SBSE conference - Fuzzing vs SBST Intersections and Differences
- SAGE (2005 -DART)
  - <https://patricegodefroid.github.io/>
  - <https://channel9.msdn.com/blogs/peli/automate-d-whitebox-fuzz-testing-with-sage> - **video (11 minutes)**
- <https://www.microsoft.com/en-us/security-risk-detection/>
- PEX
  - <https://www.microsoft.com/en-us/research/project/pex-and-moles-isolation-and-white-box-unit-testing-for-net/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Fpex%2F>
- Symbolic execution timeline



# Symbolic execution

## What is symbolic execution ?

- Symbolic execution
  - Execution of program with symbols as argument.
  - Symbolic execution supplies symbols (as input to a program) representing arbitrary values.
  - `int FunctionName(1, 2) → int FunctionName(a1 , a2)`
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- Symbolic execution
  - Produces a concrete input (a test case) on which the program will fail to meet the specification.
  - But it cannot, in general, prove the absence of errors
- Key idea
  - Evaluate the program on symbolic input values
  - Use an automated theorem prover to check whether there are corresponding concrete input values that make the program fail.



# Symbolic execution

## Symbolic state

- **Symbolic state**
  - Set of (particular) concrete states, yet not instantiated.
  - Symbolic states represent sets of concrete states.
- A **symbolic state** is described by:
  - **Variables**, i.e. symbolic values/expressions for variables;
  - **Path condition** - a conjunct of constraints on the symbolic input values;
  - **Program counter** - the statement that is executed.
- All paths in the program form its execution tree, in which some paths are feasible, and some are infeasible.
- Symbolic execution is a bug finding technique based on automated theorem proving:
  - Evaluates the program on symbolic inputs, and a solver finds concrete values for those inputs that lead to errors.

# Outline

- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
  - Sequential structure execution
  - Alternative structure execution
  - Repetitive structure execution
- Symbolic Execution Tree
  - Symbolic Execution Tree
  - Properties
- Questions
- Next lecture (still today)
  - Model checking

# Conventional vs Symbolic execution

## Conventional execution (CE)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:

	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9



# Conventional vs Symbolic execution

## Symbolic execution (SE)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:

	a	b	c	x	y	z
1	$\alpha$	$\beta$	$\gamma$	-	-	-
2	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	-	-
3	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	-
4	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$
5	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$

# Symbolic execution

## Symbolic execution for **sequential**, alternative, repetitive structures

- **Sequential structure execution**
  - path condition
    - condition to execute a statement;
  - when the symbolic execution starts, the value(pc) = true
  - the condition is updated from one statement to other
    - If  $\tau$  represents the condition to execute statement  $\langle I \rangle$  then
$$pc' = pc \wedge \tau(I)$$

# Symbolic execution

Symbolic execution for  
**sequential**, alternative, repetitive structures

Conventional

	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9

- Sequential execution -

```
1 : int Sum(int a, int b, int c)
2 :   int x := a + b;
3 :   int y := b + c;
4 :   int z := x + y - b;
5 : return z;
6 :
```

Symbolic

	a	b	c	x	y	z
1	$\alpha$	$\beta$	$\gamma$	-	-	-
2	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	-	-
3	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	-
4	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$
5	$\alpha$	$\beta$	$\gamma$	$\alpha+\beta$	$\beta+\gamma$	$\alpha+\beta+\gamma$



# Symbolic execution

## Symbolic execution for sequential, **alternative**, repetitive structures

- **Alternative structure execution**
- Symbolic execution of an IF statement
  - if ( $\eta$ ) then  
    A  
  else  
    B.
- **During symbolic execution**  $\rightarrow$  value( $\eta$ ) could be true, false, or some symbolic formula over the input symbols.
  - $\rightarrow$  “unresolved” execution of a conditional statement
- Path Condition (Initial value of pc is true)
  - $pc \rightarrow \eta$
  - $pc \rightarrow \neg\eta$

# Symbolic execution

Symbolic execution for  
sequential, **alternative**, repetitive structures

Conventional

- Alternative execution -

Symbolic

	x	b	If condition
1	6	-	-
2	6	False	-
3	6	False	6 modulo 2=0
4	6	True	6 modulo 2=0
6	6	True	6 modulo 2=0

```
1 : boolean IsEven(int x)
2 : boolean b := False;
3: If (x modulo 2 ==0) then
4:     b:=true;
   else
5:     b:=false;
6: IsEven:=b;
```

	x	b	Path condition
1	$\alpha$	-	True
2	$\alpha$	False	True
3	$\alpha$	False	$\alpha \text{ modulo } 2=0$
Case ( $\alpha \text{ modulo } 2=0$ ) is True			
3	$\alpha$	False	$\alpha \text{ modulo } 2=0$
4	$\alpha$	True	$\alpha \text{ modulo } 2=0$
6	$\alpha$	True	$\alpha \text{ modulo } 2=0$
Case ( $\text{not}(\alpha \text{ modulo } 2=0)$ ) is True			
5	$\alpha$	False	$\text{not}(\alpha \text{ modulo } 2=0)$

# Symbolic execution

## Symbolic execution for sequential, alternative, **repetitive** structures

- Symbolic execution of a WHILE statement  
    **while ( $\eta$ )**  
        **A**  
    **endWh;**  
    **B**
- During symbolic execution  $\rightarrow$  value( $\eta$ ) could be true, false, or some symbolic formula over the input symbols.
  - $\rightarrow$  “unresolved” execution of a conditional statement
- Condition to execute A: pc for executing “while” and  $\eta$ .
- Condition to execute B: pc for executing “while” and  $\neg \eta$ .



# Symbolic execution

Symbolic execution for  
sequential, alternative, **repetitive** structures

## Conventional

	x	y	z	u	While condition
1	5	3	-	-	
2	5	3	1	-	
3	5	3	1	1	
4	5	3	1	1	$1 \leq 3$
5	5	3	5	1	
6	5	3	5	2	
4	5	3	5	2	$2 \leq 3$
5	5	3	25	2	
6	5	3	25	3	
4	5	3	5	3	$3 \leq 3$
5	5	3	75	3	
6	5	3	75	4	
4	5	3	75	4	$\text{not } 4 \leq 3$
7					
8	5	3	75	4	

## - Repetitive execution -

```

1 : Power(int x, int y, int z)
2 :   z := 1;
3 :   u:=1
4 :   while(u ≤ y)
5 :       z:=z*x;
6 :       u:=u+1
7 :   endwh;
8 :

```

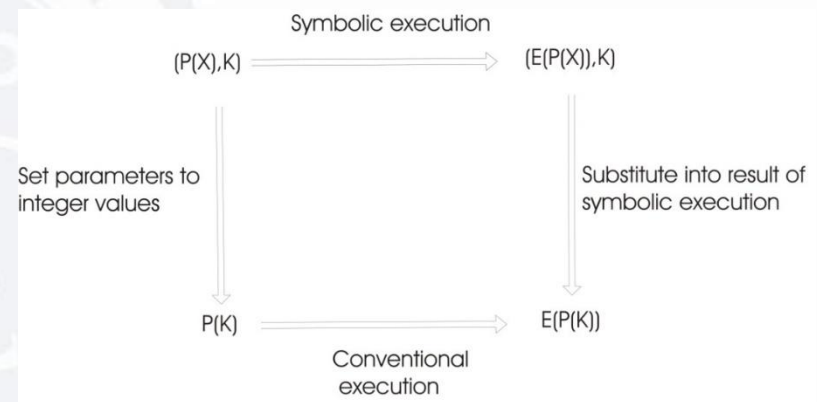
## Symbolic

	x	y	z	u	Path condition	Remarks
1	$\alpha$	$\beta$	-	-	True	
2	$\alpha$	$\beta$	1	-		
3	$\alpha$	$\beta$	1	1		
4	$\alpha$	$\beta$	1	1	$1 \leq \beta$	
Case $\text{not}(1 \leq \beta), \rightarrow 1 > \beta$						
4	$\alpha$	$\beta$	1	1	$1 > \beta$	
8	$\alpha$	$\beta$	1	1		$\beta=0$ and $z=1$
Case $(1 \leq \beta)$						
4	$\alpha$	$\beta$	1	1	$1 \leq \beta$	
5	$\alpha$	$\beta$	$\alpha$	1	$1 \leq \beta$	
6	$\alpha$	$\beta$	$\alpha$	2	$1 \leq \beta$	
7						
4	$\alpha$	$\beta$	$\alpha$	2	$2 \leq \beta$ and $1 \leq \beta$	
Case $\text{not}(2 \leq \beta)$ and $1 \leq \beta, \rightarrow 2 > \beta$ and $1 \leq \beta$						
4	$\alpha$	$\beta$	$\alpha$	2	$2 > \beta$ and $1 \leq \beta$	
8	$\alpha$	$\beta$	$\alpha$	2		$\beta=1$ and $z=\alpha$
Case $(2 \leq \beta)$ and $1 \leq \beta$						
4	$\alpha$	$\beta$	$\alpha$	2	$2 \leq \beta$ and $1 \leq \beta$	
5	$\alpha$	$\beta$	$\alpha^2$	2	$2 \leq \beta$ and $1 \leq \beta$	
6	$\alpha$	$\beta$	$\alpha^2$	3	$2 \leq \beta$ and $1 \leq \beta$	
7						
4	$\alpha$	$\beta$	$\alpha^2$	3	$3 \leq \beta$ and $2 \leq \beta$ and $1 \leq \beta$	
Case $\text{not}(3 \leq \beta)$ and $2 \leq \beta$ and $1 \leq \beta$ $\rightarrow 3 > \beta$ and $2 \leq \beta$ and $1 \leq \beta$						
4	$\alpha$	$\beta$	$\alpha^2$	3	$3 > \beta$ and $2 \leq \beta$ and $1 \leq \beta$	
8	$\alpha$	$\beta$	$\alpha^2$	3		$\beta=2$ and $z=\alpha^2$

# Symbolic execution

## Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
  - $\text{Sum}(a, b, c) \rightarrow \text{Sum}(1, 3, 5)$
  - $\text{Sum}(1, 3, 5) = 9$
- Symbolic execution (SE)
  - $\text{Sum}(a, b, c) = \alpha + \beta + \gamma$
  - Instantiate the symbolic result
    - $\rightarrow \alpha = 1, \beta = 3, \gamma = 5$
    - $\rightarrow 1+3+5=9$



# Outline

- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
  - Sequential structure execution
  - Alternative structure execution
  - Repetitive structure execution
- Symbolic Execution Tree
  - Symbolic Execution Tree
  - Properties
- Questions
- Next lecture (still today)
  - Model checking



# Symbolic execution

## Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
  - Associate a node with each statement executed.
  - Associate a directed arc connecting the associated nodes with each transition between statements.
  - For IF statement execution, the associated node has two arcs leaving the node which are labeled “T” and “F” for the true and false part, respectively.
  - Associate the complete current execution state, i.e. variable values, statement counter, and pc with each node.

# Symbolic execution

## Symbolic Execution Tree

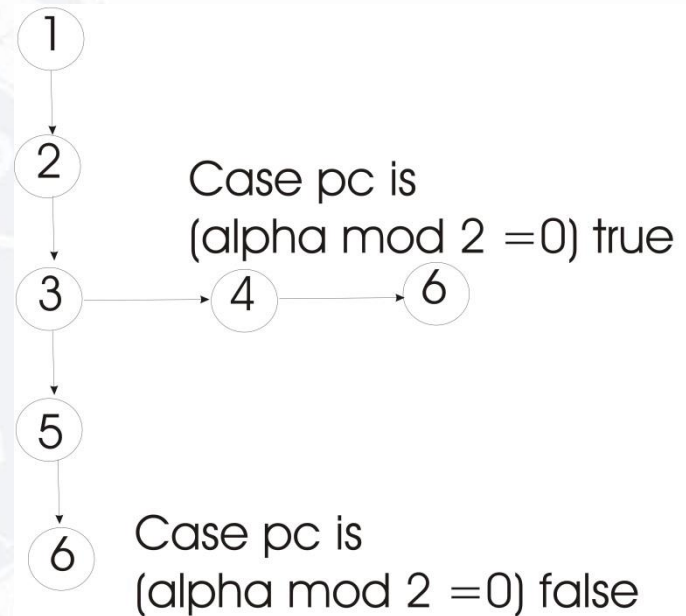
```
1 : int Sum(int a, int b, int c)
2 :   int x := a + b;
3 :   int y := b + c;
4 :   int z := x + y - b;
5 : return z;
6 :
```



# Symbolic execution

## Symbolic Execution Tree

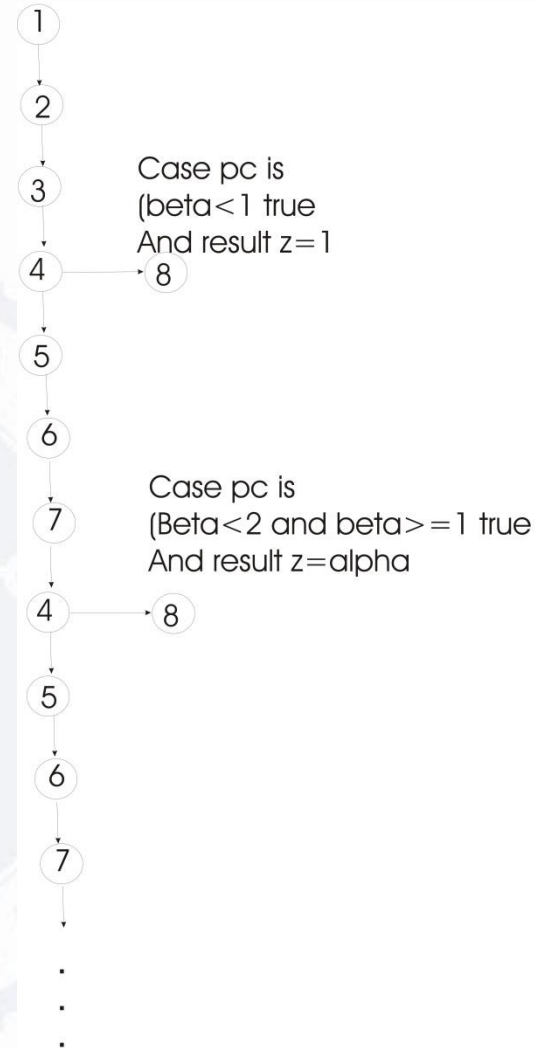
```
1 : boolean IsEven(int a)
2 : boolean b := False;
3: If (x modulo 2 ==0) then
4:     b:=true;
   else
5:     b:=false;
6: IsEven:=b;
```



# Symbolic execution

## Symbolic Execution Tree

```
1 : Power(int x, int y, int z)
2 :   z := 1;
3 :   u:=1
4 :   while(u ≤ y)
5 :     z:=z*x;
6 :     u:=u+1
7 :   endwh;
8 :
```



# Symbolic execution

## Properties of the Symbolic Execution Tree

- For each terminal leaf exists a particular non symbolic input.
- The pc associated with any two terminal leaves are distinct.
- Test case generation
  - to execute every statement at least once
  - to include execution of each branch both ways
  - finding input values to reach a particular point in a program

### Symbolic execution

- Symbolic variables for input variables
- Execute the program symbolically
- Collect symbolic path constraints
- Use constraint solver to generate test inputs for each execution path
- **Remaining problem** - to instantiate the pc with particular values.
- The **pc** specifies a **class of equivalent tests**, and any feasible solution to the constraints (represented by the pc) would be a representative member.
- The symbolic execution also provides expressions describing the program outputs for all inputs in this set.

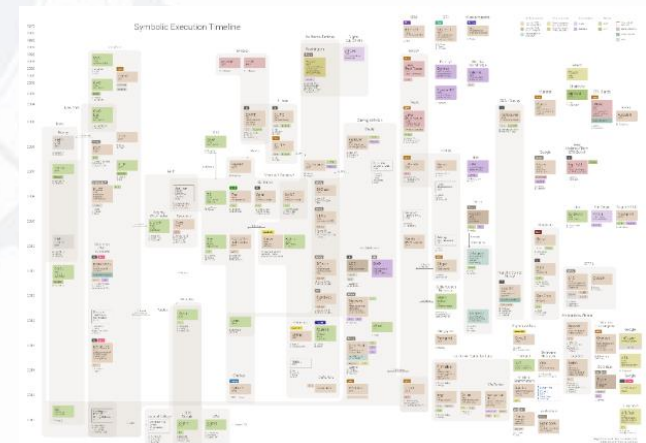


# Symbolic execution

## Symbolic execution – research- revisited

- 1976 - King [Kin76], Clarke [Cla76]
- ....
- 2005 - Microsoft: DART [God05]
- 2006 - Univ. Stanford: EXE, Univ. Illinois: CUTE and jCUTE [SA06]
- ...
- 2008 - KLEE (Stanford) [CDE08]
- ...
- 1999 - 2016 - NASA: Symbolic (Java) Path Finder [PV09], [CS13]
  - <http://javapathfinder.sourceforge.net/>
  - <http://babelfish.arc.nasa.gov/trac/jpf>
- Modern Symbolic Execution Techniques
  - mix concrete and symbolic execution
  - Concolic testing (DART – Directed Automated Random Testing)
  - EGT (Execution-Generated Testing)
- 2017 -Learn&Fuzz: Machine Learning for Input Fuzzing
  - <https://patricegodefroid.github.io/>
- 2018 -Chopped Symbolic Execution (ICSE) (2006 -EXE)
  - <https://www.doc.ic.ac.uk/~cristic/>
- 2018 -Deep Reinforcement Fuzzing, Konstantin Böttinger, Patrice Godefroid, Rishabh Singh
- 2022 – SBSE conference - Fuzzing vs SBST Intersections and Differences

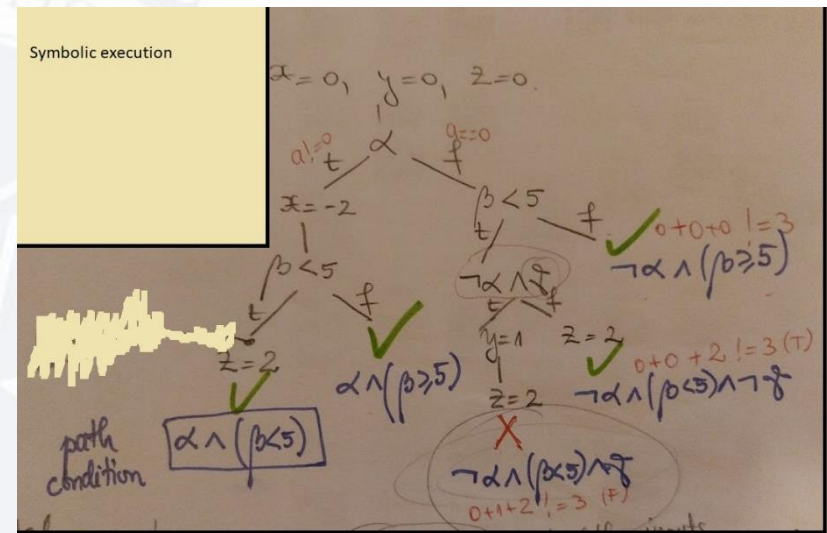
- SAGE (2005 -DART)
  - <https://patricegodefroid.github.io/>
  - <https://channel9.msdn.com/blogs/peli/automated-whitebox-fuzz-testing-with-sage-> **video**
- PEX
  - <https://www.microsoft.com/en-us/research/project/pex-and-moles-isolation-and-white-box-unit-testing-for-net/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Fpex%2F>
- Symbolic execution timeline



# Symbolic execution

Symbolic Execution – example -<http://klee.github.io/getting-started/>  
<http://klee.github.io/tutorials/testing-function/>

```
// Edit SymbolicExecutionExample.c
void SymbolicExecutionExample(int
a, int b, int c){
    int x=0, y=0, z=0;
    if (a!=0){
        x=-2;
    }
    if (b<5){
        if(!a && c){y=1;}
        z=2;
    }
    assert(x+y+z!=3);
}
```



- <http://klee.github.io/getting-started/>  
**KLEE web:** run tiny code examples in your browser

**Cristian Cadar**

<http://www.doc.ic.ac.uk/~cristic/>

Execution and test cases created  
<https://klee.github.io/tutorials/testing-function/>

# 2022 - Fuzzing vs SBST Intersections and Differences

The image shows a screenshot of the SBST 2022 website and a promotional poster. The website header includes navigation links like 'SEARCH-BASED SOFTWARE TESTING', 'ORGANISATION', 'TOOL COMPETITION', and 'SBST VS FUZZING PANEL'. The poster for the 'Discussion Panel SBST vs Fuzzing' is dated '9th of May, 2pm-4pm CEST' and features portraits of seven participants: Andreas Zeller, Annibale Panichella, Lionel Briand, Marcel Böhme, Mark Harman, Myra Cohen, and Paolo Tonella. A QR code and the URL 'https://sbst22.github.io' are also present on the poster.

sbst22.github.io/program/

SEARCH-BASED SOFTWARE TESTING

ORGANISATION

TOOL COMPETITION

SBST VS FUZZING PANEL

KEYNOTE & TUTORIAL

PROG

Luigi Libero Lucio Stavace, Andreea Rumpelhan, Sergio Di Martino

Rahana Ferdous, Chia-kang Hung, Fitsum Meshesha Kifetew, Davide Prandi and Angelo Susi

Wogan

Jarkko Peltomäki

## Workshop Format

SBST22 will be held virtually this year. The format of the workshop will be as follows:

**Authors and registered participants**

We will use Zoom for SBST2022 for registered participants. To minimize issues during the time of the workshop, we ask that author presentations be *pre-recorded*. Authors are still welcome to present live.

**Similar to last year, we will also live-stream SBST2022 via Twitch.** Feel free to join our stream and ask questions in the chat if you did not register.

**Recording**

Following the workshop, we will make recordings available online.

[https://www.youtube.com/watch?v=OS0-k7WQkus&t=2s&ab\\_channel=Search-BasedSoftwareTestingWorkshop%28SBST%29](https://www.youtube.com/watch?v=OS0-k7WQkus&t=2s&ab_channel=Search-BasedSoftwareTestingWorkshop%28SBST%29)

# Outline

- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
  - Sequential structure execution
  - Alternative structure execution
  - Repetitive structure execution
- Symbolic Execution Tree
  - Symbolic Execution Tree
  - Properties
- Questions
- Next lecture (still today)
  - Model checking

# References

- [Kin76] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.
- [Cla76] L. A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, SE-2(3):215–222, 1976.
- [God05] P. Godefroid. Dart: directed automated random testing. pages 213–223, 2005.
- [SA06] Koushik Sen and Gul Agha. Cute and jcute: Concolic unit testing and explicit path model-checking tools. In *Proceedings of the 18th International Conference on Computer Aided Verification*, pages 419–423, 2006.
- [CDE08] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pages 209–224, 2008.
- [PV09] Corina S. Pasareanu and Willem Visser. A survey of new trends in symbolic execution for software testing and analysis. *Int. J. Softw. Tools Technol. Transf.*, 11(4):339–353, 2009.
- [CS13] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, 2013.





# Software Systems Verification and Validation

---

"Tell me and I forget, teach me and I may remember, involve me and I learn."

(Benjamin Franklin)