FIELD FAILURE REPRODUCTION USING SYMBOLIC EXECUTION AND GENETIC PROGRAMMING

Alessandro (Alex) Orso
School of Computer Science – College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR

DSE SBST

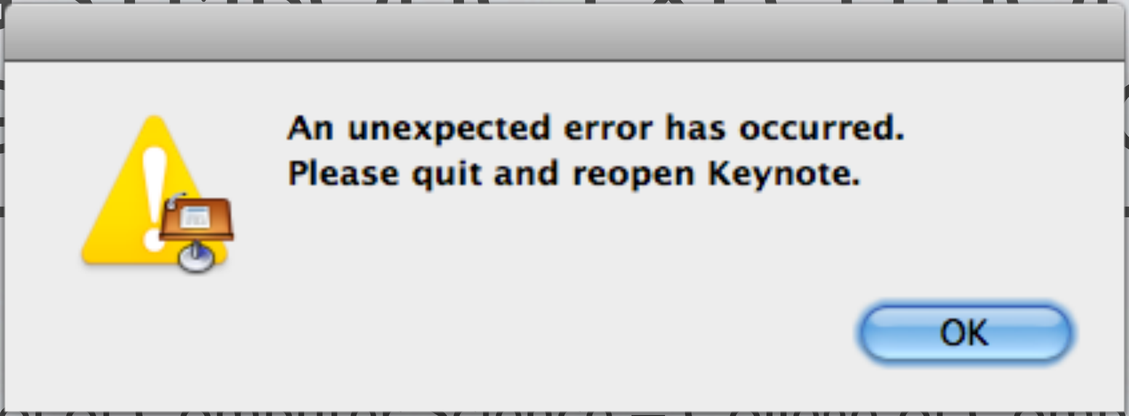
FIELD FAILURE REPRODUCTION USING SYMBOLIC EXECUTION AND GENETIC PROGRAMMING

Alessandro (Alex) Orso
School of Computer Science – College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR

DSE SBST

FIELD FAILURE REPRODUCTION USING SYMBOLIC EXECUTION AND G G



An unexpected error has occurred.
Please quit and reopen Keynote.

School of Computer Science – College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR

DSE

SBST

FIELD FAILURE REPRODUCTION USING SYMBOLIC EXECU AND

Field failures are unavoidable!

S Computing

Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR

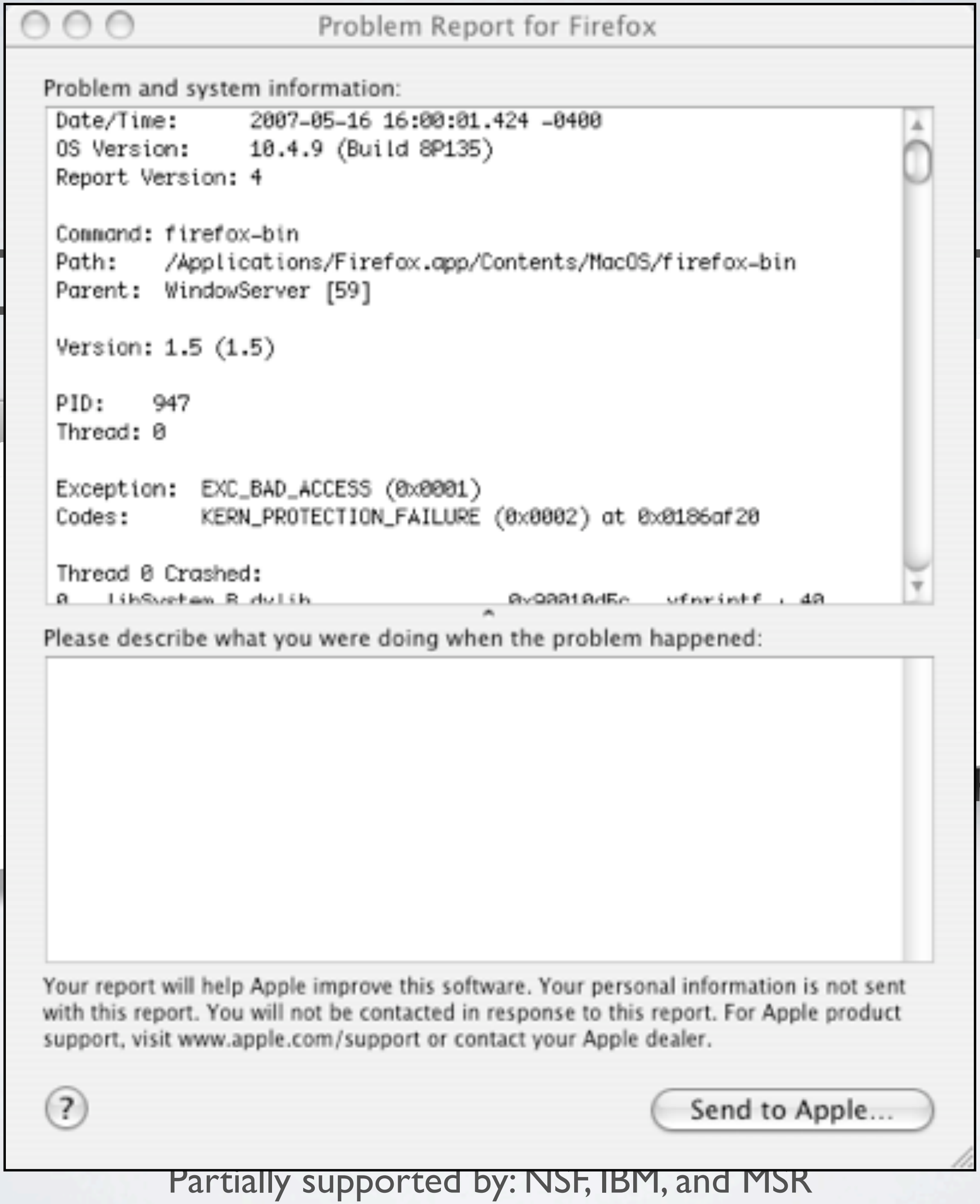
DSE

SBST

FIELD TION

USING AND

S puting



Problem Report for Firefox

Problem and system information:

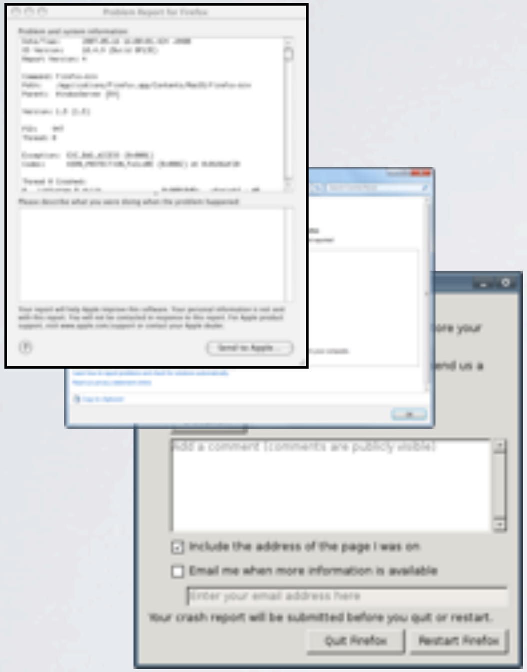
```
Date/Time:      2007-05-16 16:00:01.424 -0400
OS Version:     10.4.9 (Build 8P135)
Report Version: 4

Command: firefox-bin
Path:    /Applications/Firefox.app/Contents/MacOS/firefox-bin
Parent:  WindowServer [59]

Version: 1.5 (1.5)

PID:    947
Thread: 0

Exception:  EXC_BAD_ACCESS (0x0001)
Codes:      KERN_PROTECTION_FAILURE (0x0002) at 0x0186af20

Thread 0 Crashed:
0   libSystem.B.dylib               0x90010d5c   vfprintf + 40
```

Please describe what you were doing when the problem happened:

Your report will help Apple improve this software. Your personal information is not sent with this report. You will not be contacted in response to this report. For Apple product support, visit www.apple.com/support or contact your Apple dealer.

Partially supported by: NSF, IBM, and MSR

TYPICAL DEBUGGING PROCESS



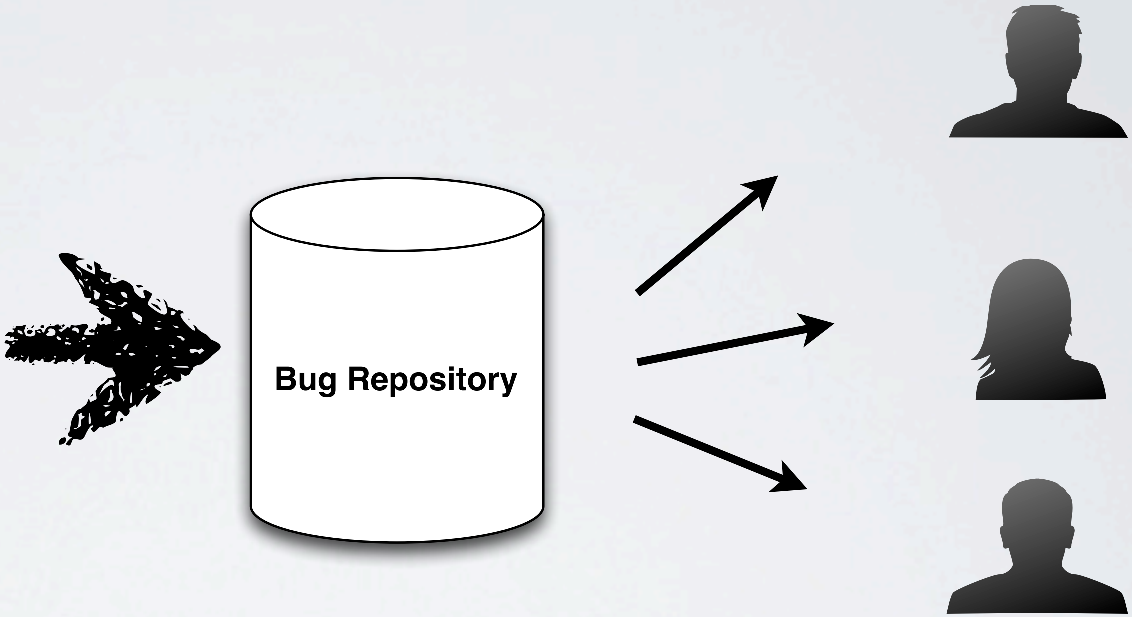
Add a comment (comments are publicly visible)

☑ Include the address of the page I was on

□ Email me when more information is available

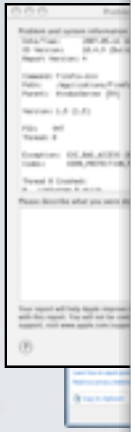
Enter your email address here

Your crash report will be submitted before you quit or restart.



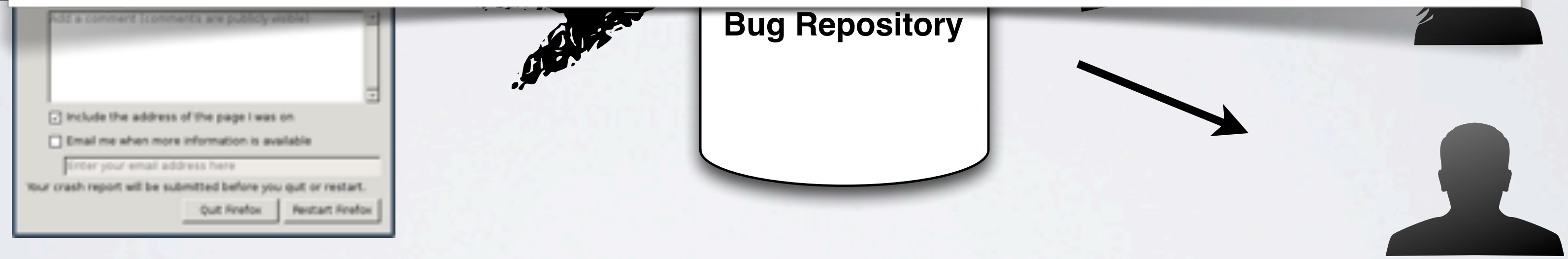
Very hard to
(1) reproduce
(2) debug

TYPICAL DEBUGGING PROCESS



Recent survey of
Apache, Eclipse, and Mozilla developers:

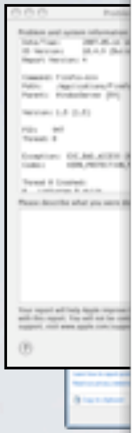
Information on *how to reproduce field failures* is the most valuable, and difficult to obtain, piece of information for investigating such failures.
[Zimmermann10]



Add a comment (comments are publicly visible)
☑ Include the address of the page I was on
☐ Email me when more information is available
Enter your email address here
Your crash report will be submitted before you quit or restart.
Quit Firefox
Restart Firefox
Bug Repository

Very hard to
(1) reproduce
(2) debug

TYPICAL DEBUGGING PROCESS



Recent survey of
Apache, Eclipse, and Mozilla developers:

Information on *how to reproduce field failures* is the most valuable, and difficult to obtain, piece of information for investigating such failures.
[Zimmermann10]

OVERARCHING GOAL: help developers
(1) *investigate* field failures,
(2) *understand* their causes, and
(3) *eliminate* such causes.

OUR WORK SO FAR

Recording and replaying executions

[icsm 2007, icse 2007]

Input minimization

[woda 2006, icse 2007]

Input anonymization

[icse 2011]

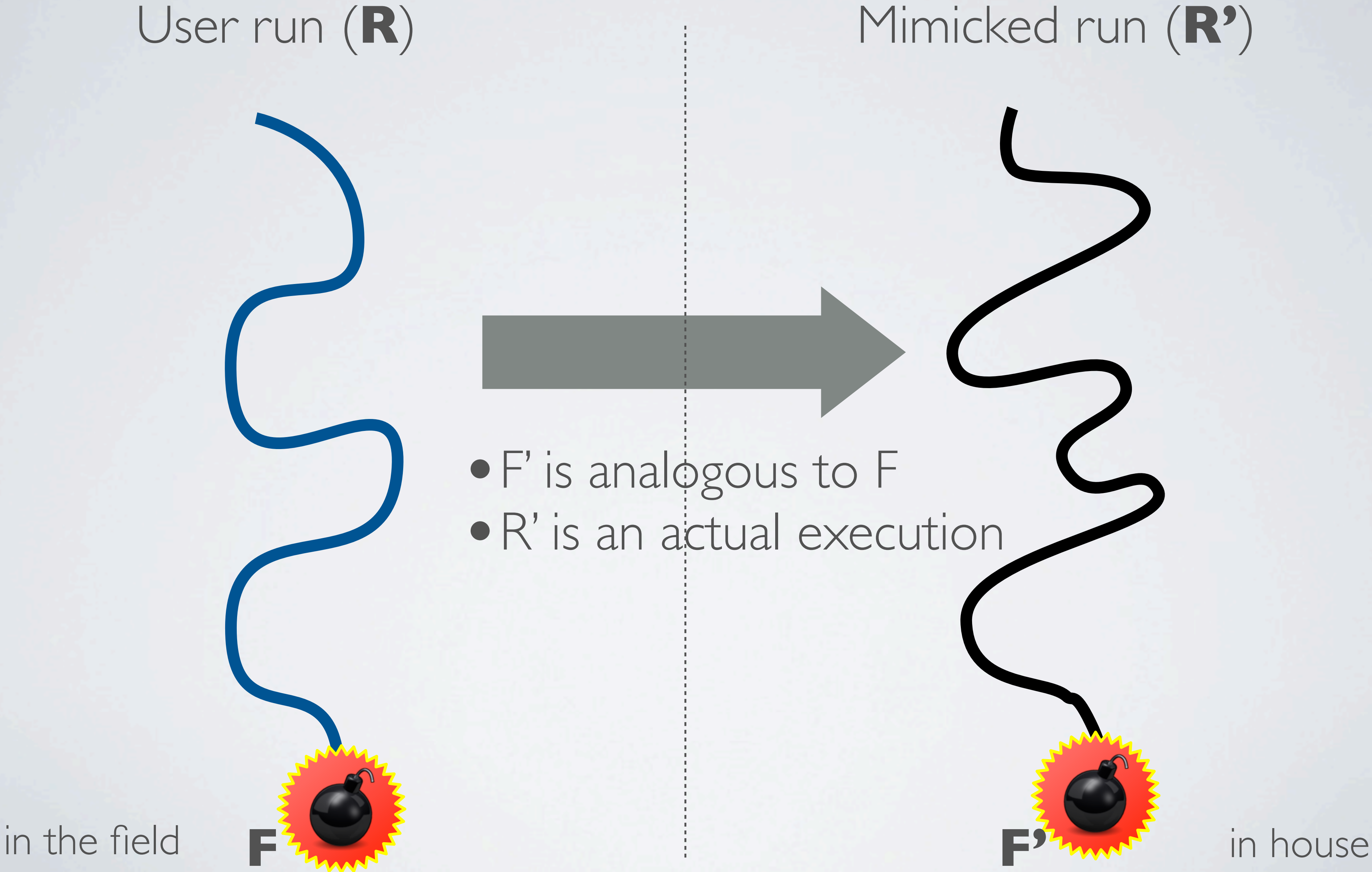
Mimicking field failures

[icse 2012, icst 2014]

Explaining field failures

[issta 2013, TR]

MIMICKING FIELD FAILURES



User run (**R**)

Mimicked run (**R'**)

- F' is analogous to F
- R' is an actual execution

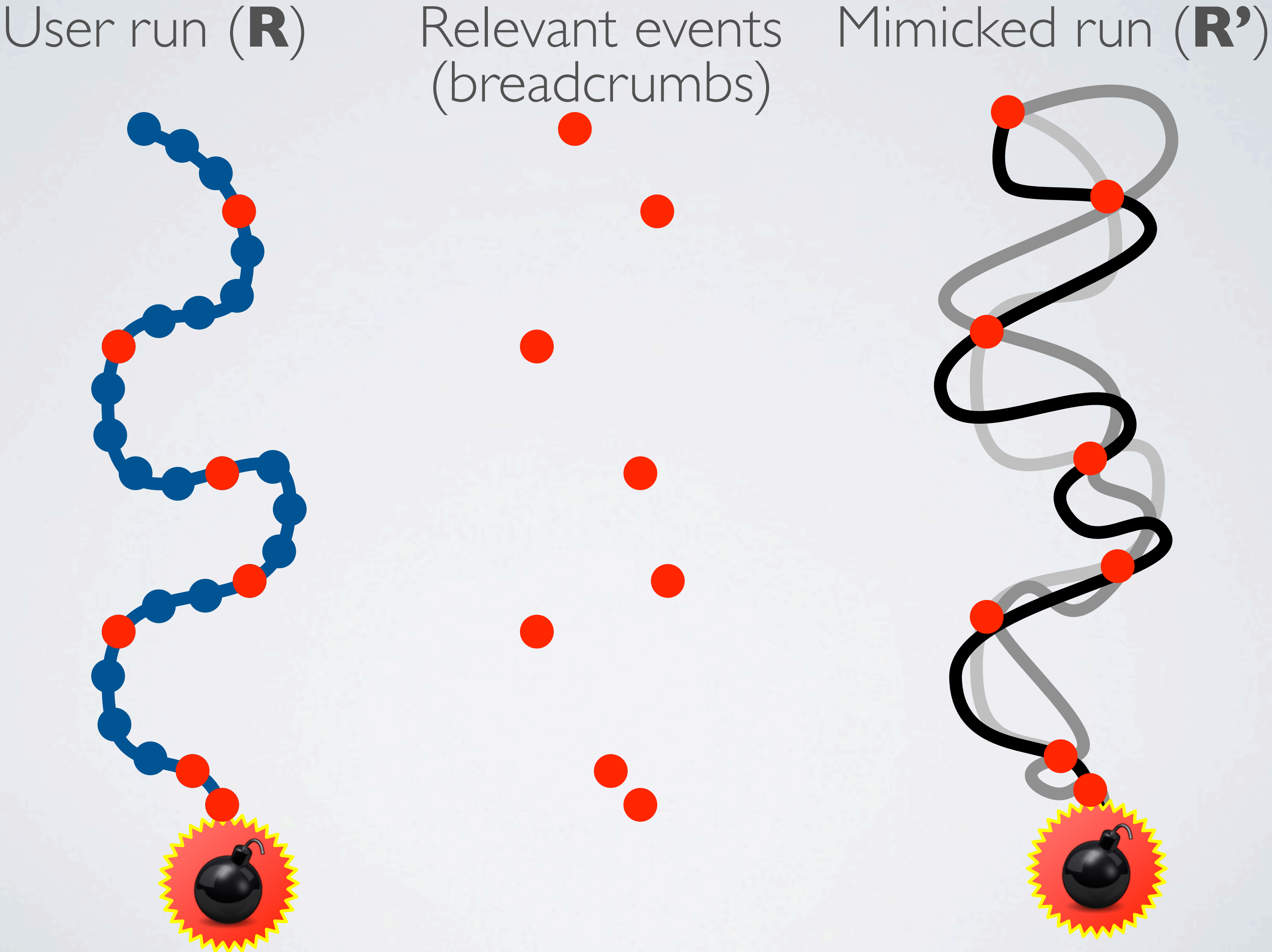
in the field

F

F'

in house

MIMICKING FIELD FAILURES

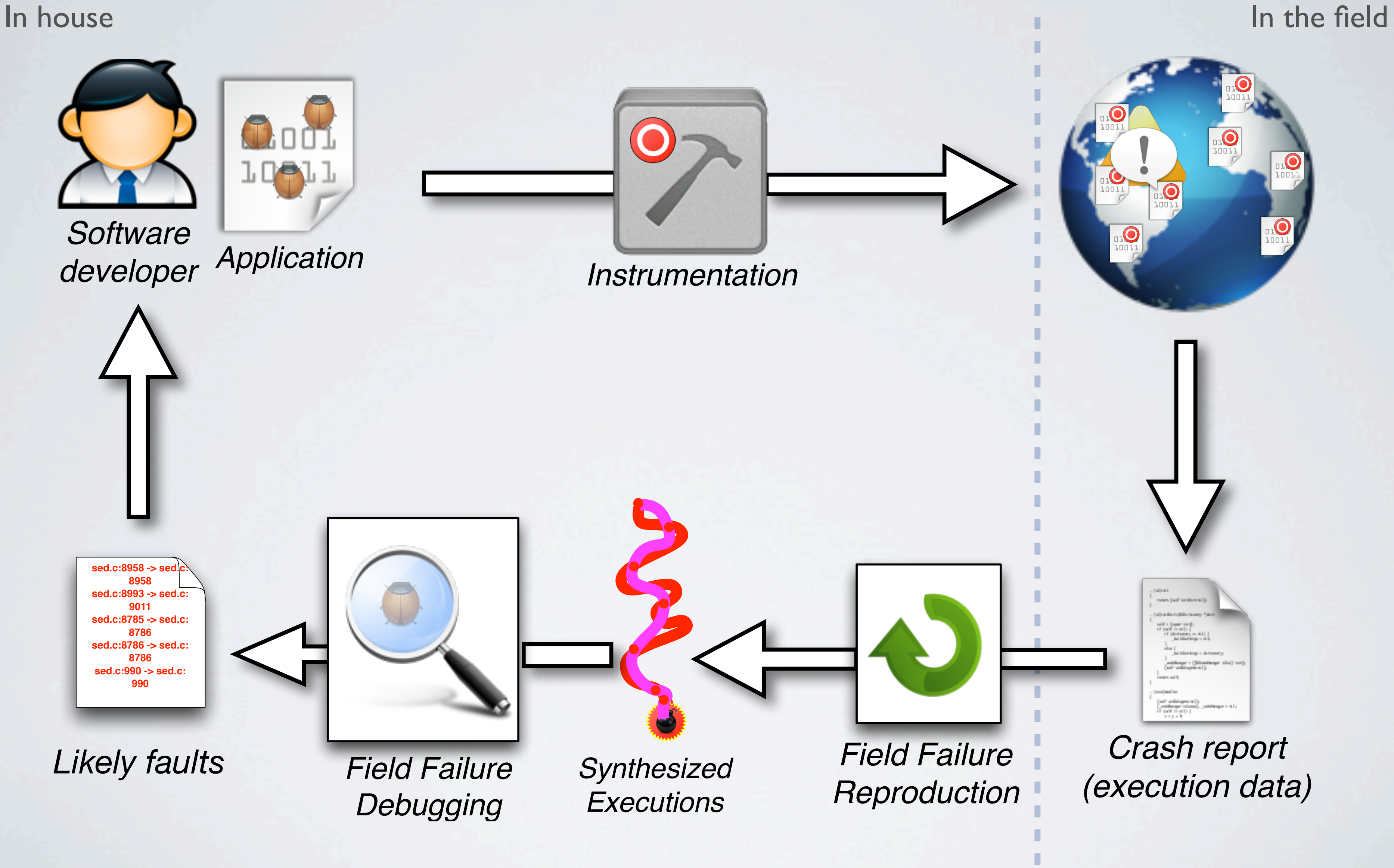


User run (**R**)

Relevant events (breadcrumbs)

Mimicked run (**R’**)

OVERALL VISION



In house

In the field

Software developer

Application

Instrumentation

Crash report (execution data)

Field Failure Reproduction

Synthesized Executions

Field Failure Debugging

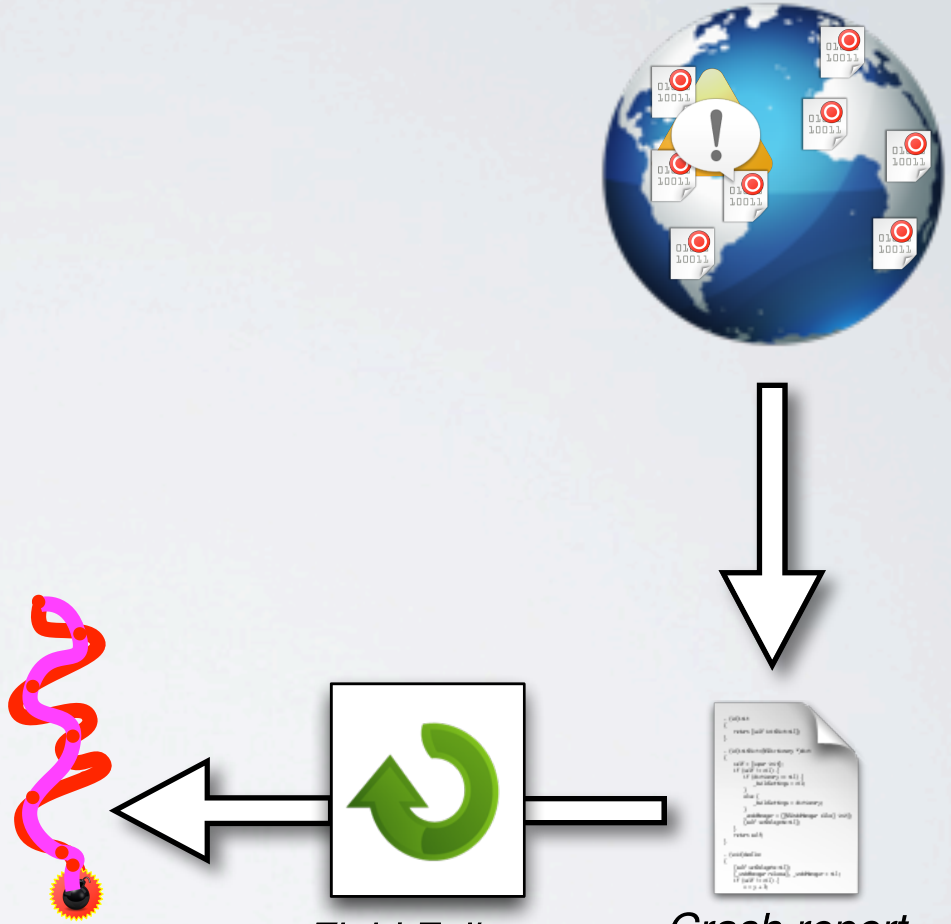
Likely faults

sed.c:8958 -> sed.c:8958
sed.c:8993 -> sed.c:9011
sed.c:8785 -> sed.c:8786
sed.c:8786 -> sed.c:8786
sed.c:990 -> sed.c:990

DSE

BUGREDUX/SBFR

SBST



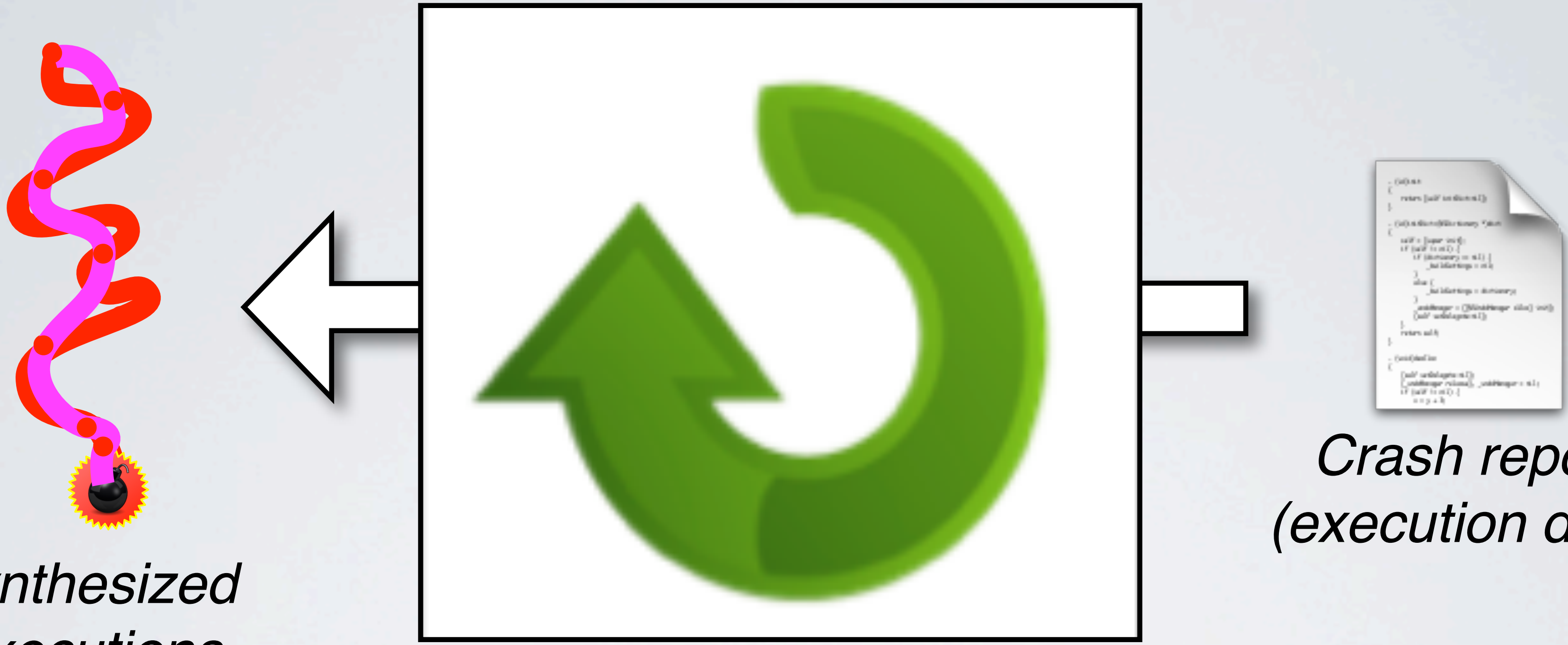
Synthesized Executions

Field Failure Reproduction

Crash report (execution data)

Joint work with Wei Jin

BUGREDUX

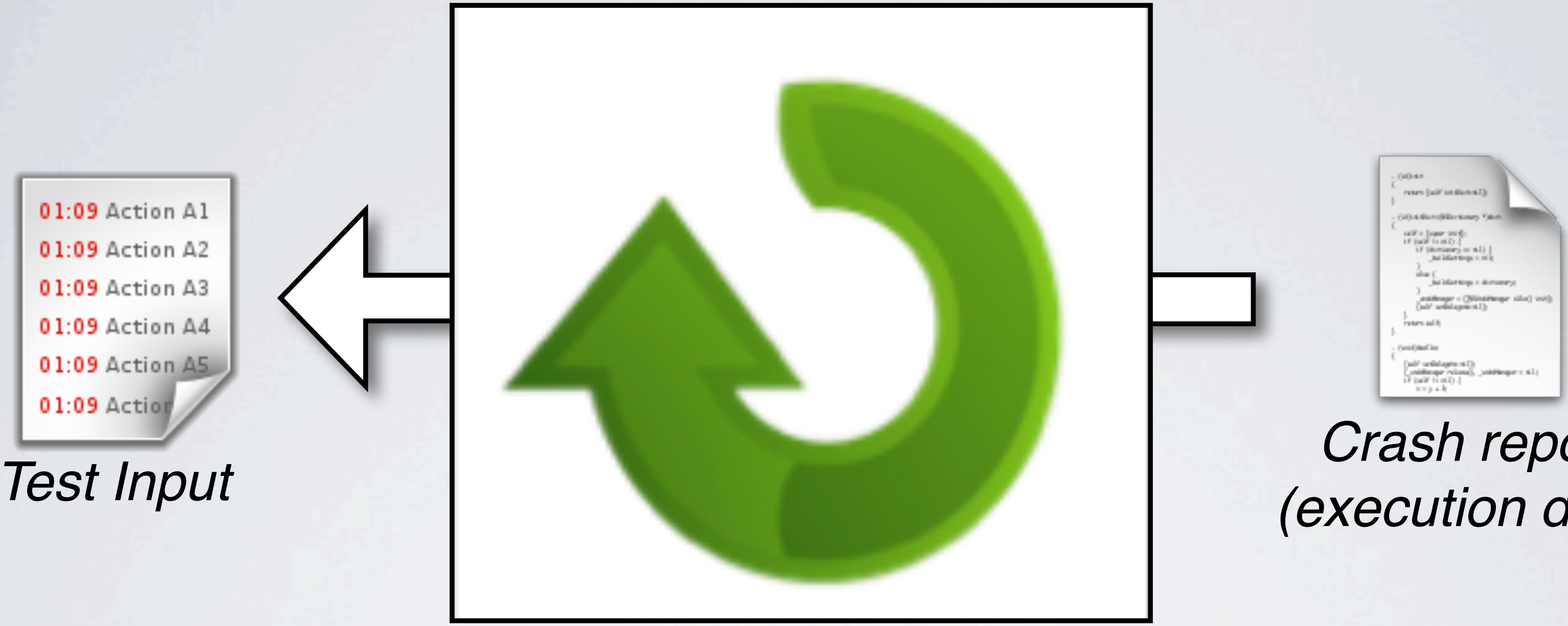


Crash report (execution data)

Synthesized Executions

Joint work with Wei Jin

BUGREDUX

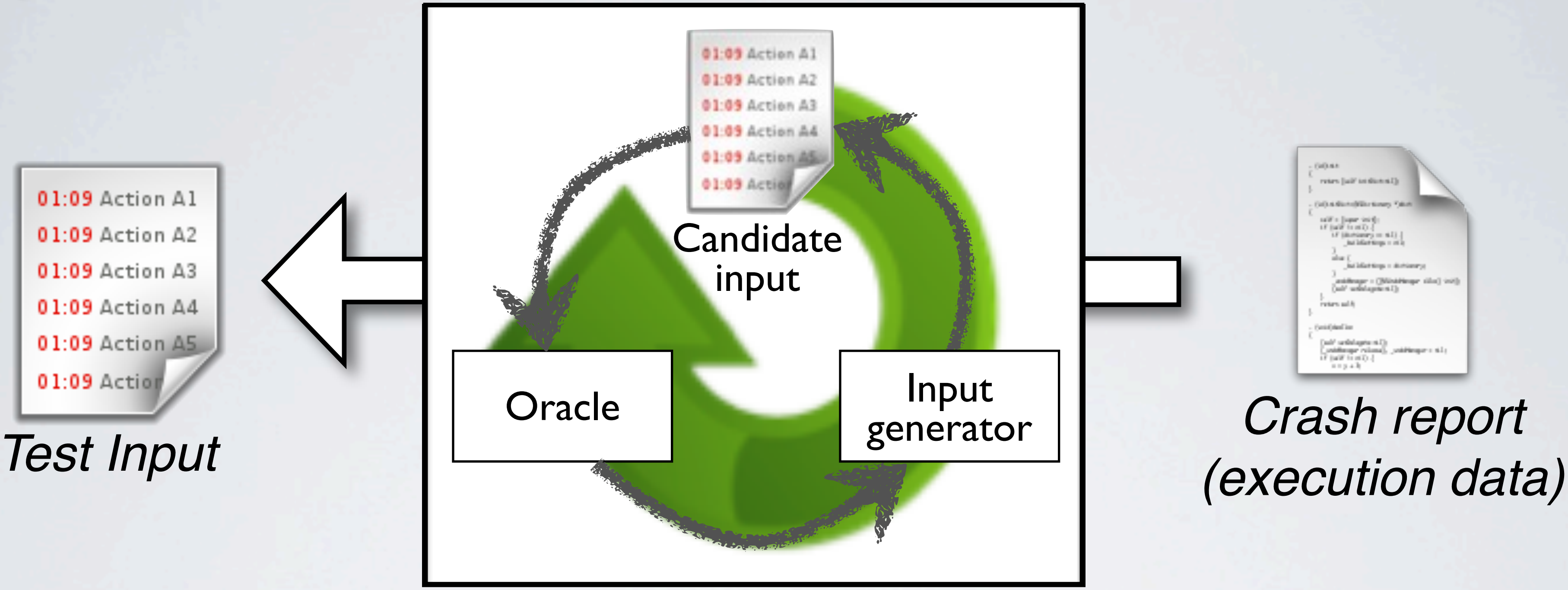


Test Input

Crash report (execution data)

Joint work with Wei Jin

BUGREDUX



- **Execution data**
  - Point of failure (POF)
  - Failure call stack
  - Call sequence
  - Complete trace
- **Input generation technique**
  - Guided symbolic execution

ALGORITHM (SIMPLIFIED)

Input

icfg for P

goals (list of code locations)

Output

I_f (candidate input)

Main algorithm

```
init; currGoal = first(goals)
repeat
  currState = SelNextState()
  if (!currState) backtrack or fail
  if (currState.cl == currGoal)
    if (currGoal == last(goals))
      return solve(currState.pc)
    else
      currGoal = next(goals)
      currState.goal = currGoal
  symbolicallyExec(currState)
```

statesSet= {<cl, pc, ss, goal>}

SelNextState

```
minDis = ∞
retState = null

foreach state in statesSet
  if (state.goal = currGoal)
    if (state.cl can reach currGoal)
      d = |shortest path state.cl, currGoal|
      if d < minDis
        minDis = d
        retState = state
return retState
```

ALGORITHM (SIMPLIFIED)

Input
icfg for P
goals (list of code locations)

Output
I_f (candidate input)

Main algorithm

```
ir
re

  i
  i                              )
    if (currGoal == last(goals))
      return solve(currState.pc)
    else
      currGoal = next(goals)
      currState.goal = currGoal
  symbolicallyExec(currState)
```

statesSet= {<cl, pc, ss, goal>}

```
                    tatesSet
  if (state.goal = currGoal)
    if (state.cl can reach currGoal)
      d = |shortest path state.cl, currGoal|
      if d < minDis
        minDis = d
        retState = state
return retState
```

Optimizations/Heuristics

Dynamic tainting to reduce the symbolic input space
Program analysis information to prune the search space
Some randomness in the shortest path computation

BUGREDUX EVALUATION – FAILURES CONSIDERED

Name	Repository	Size(KLOC)	# Faults
sed	SIR	14	2
grep	SIR	10	1
gzip	SIR	5	2
ncompress	BugBench	2	1
polymorph	BugBench	1	1
aeon	exploit-db	3	1
glftpd	exploit-db	6	1
htget	exploit-db	3	1
socat	exploit-db	35	1
tipxd	exploit-db	7	1
aspell	exploit-db	0.5	1
exim	exploit-db	241	1
rsync	exploit-db	67	1
xmail	exploit-db	1	1

BUGREDUX EVALUATION – FAILURES CONSIDERED

Name	Repository	Size(KLOC)	# Faults
sed	SIR	14	2
grep	SIR	10	1
gzip	SIR	5	2
ncompress	BugBench	2	1
polymorph	BugBench	1	1
aeon	exploit-db	[illegible]	[illegible]
glftpd	[illegible]	[illegible]	[illegible]
htget	[illegible]	[illegible]	1
socat	exploit-db	35	1
tipxd	exploit-db	7	1
aspell	exploit-db	0.5	1
exim	exploit-db	241	1
rsync	exploit-db	67	1
xmail	exploit-db	1	1

None of these faults can be discovered by a vanilla KLEE with a timeout of 72 hours

BUGREDUX EVALUATION – RESULTS

Name	POF	Call Stack	Call Seq.	Compl. Trace
sed #1				
sed #2				
grep				
gzip #1				
gzip #2				
ncompress				
polymorph				
aeon				
rsync				
glftpd				
htget				
socat				
tipxd				
aspell				
xmail				
exim				

One of three outcomes:

✘: fail

∼: synthesize

✔: (synthesize and) mimic

BUGRE V I E

Synth.: 9/16
Mimic: 6/16

Synth.: 10/16
Mimic: 6/16

Synth.: 16/16
Mimic: 16/16

Synth.: 2/16
Mimic: 2/16

Name	POF	Call Stack	Call Seq.	Compl. Trace
sed #1	✗	✗	✓	✗
sed #2	✗	✗	✓	✗
grep	✗	~	✓	✗
gzip #1	✓	✓	✓	✗
gzip #2	~	~	✓	✗
ncompress	✓	✓	✓	✗
polymorph	✓	✓	✓	✗
aeon	✓	✓	✓	✓
rsync	✗	✗	✓	✗
glftpd	✓	✓	✓	✗
htget	~	~	✓	✗
socat	✗	✗	✓	✗
tipxd	✓	✓	✓	✗
aspell	~	~	✓	✗
xmail	✗	✗	✓	✗
exim	✗	✗	✓	✓

BUGRE V I E

	Synth.: 9/16 Mimic: 6/16	Synth.: 10/16 Mimic: 6/16	Synth.: 16/16 Mimic: 16/16	Synth.: 2/16 Mimic: 2/16
Name	**POF**	**Call Stack**	**Call Seq.**	**Compl. Trace**
sed #1	✗	✗	✓	✗
sed #2	✗	✗	✓	✗
grep			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✓
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
	✗	✗	✓	✗
exim	✗	✗	✓	✓

Observations:

- Faults can be distant from the failure points: => POFs and call stacks unlikely to help
- More information is not always better
- Symbolic execution can be a limiting factor

BUGRE V IC E

Synth.: 9/16
Mimic: 6/16

Synth.: 10/16
Mimic: 6/16

Synth.: 16/16
Mimic: 16/16

Synth.: 2/16
Mimic: 2/16

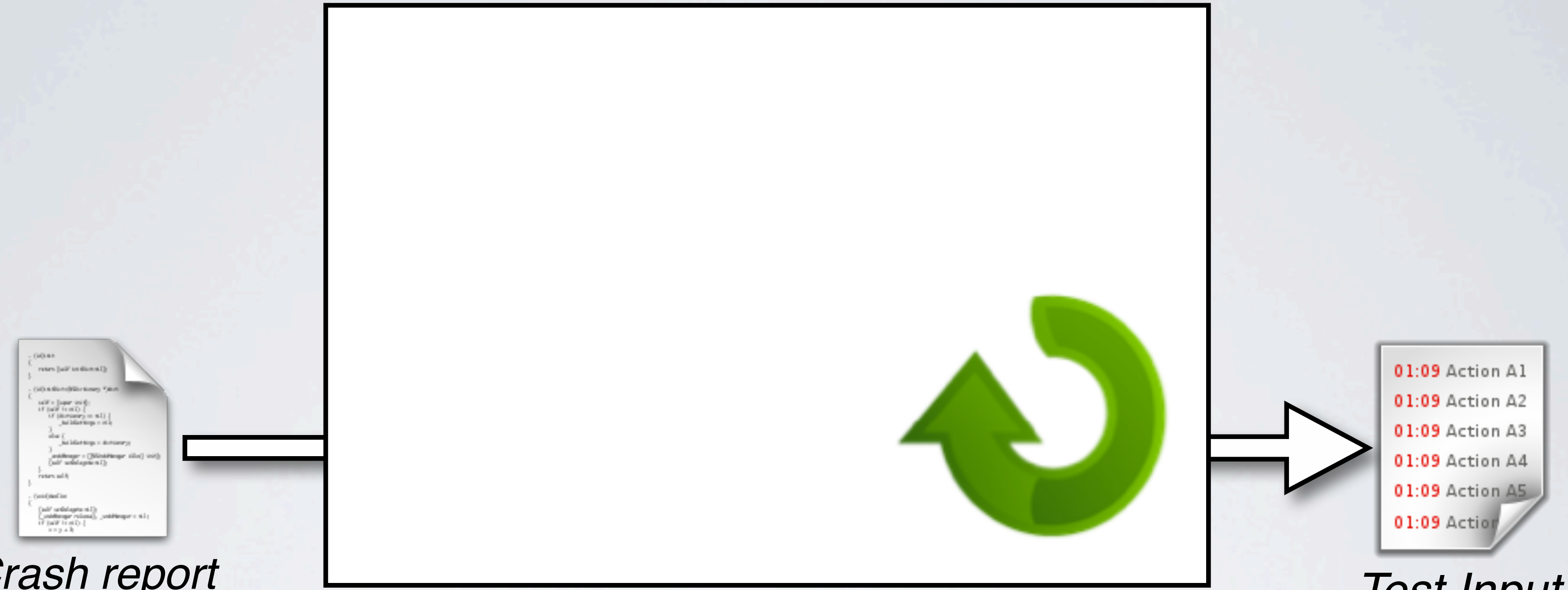
Name	POF	Call Stack	Call Seq.	Compl. Trace
sed #1	✗	✗	✓	✗
sed #2	✗	✗	✓	✗
grep			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✓
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
			✓	✗
	✗	✗	✓	✗
exim	✗	✗	✓	✓

Symbolic execution can be ineffective for

- programs with highly structured inputs
- programs that interact with external libraries
- large complex programs in general

SBFR

Joint work with
Kifetew, Jin, Tiella, Tonella



01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Actio

Crash report (execution data)

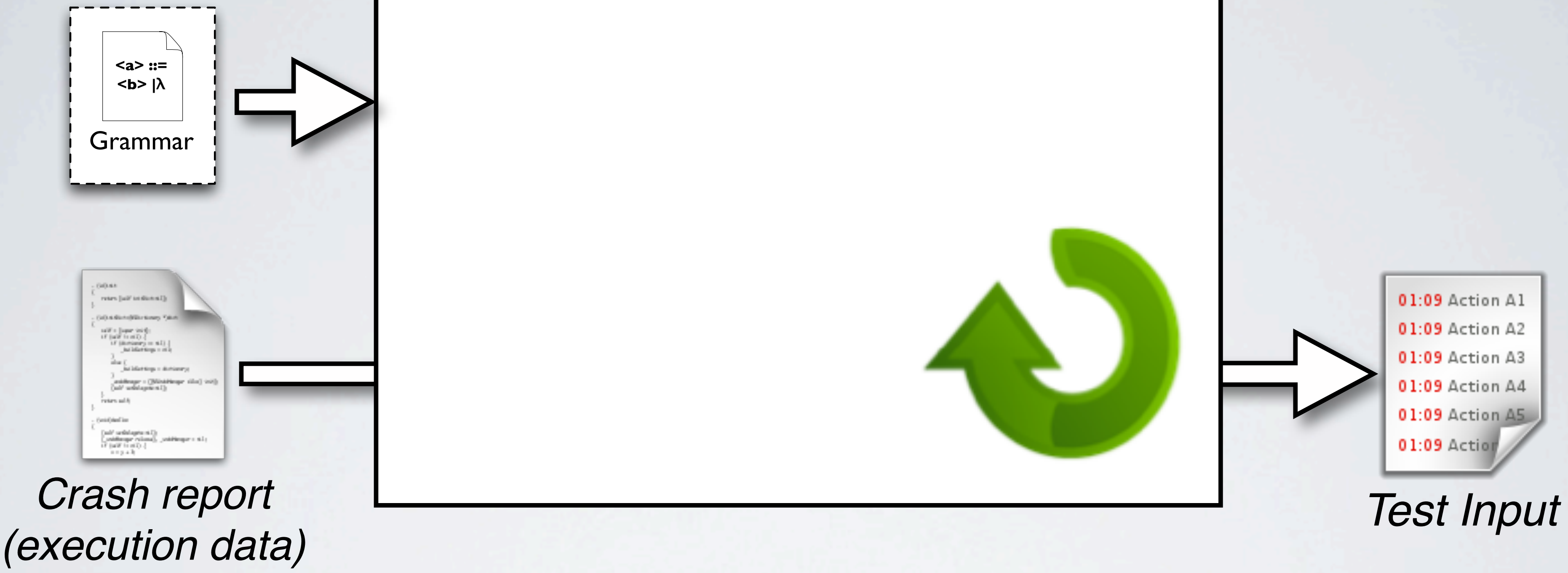
Test Input

- **Execution data**
  - Call sequence

- **Input generation technique**
  - Genetic Programming

Joint work with
Kifetew, Jin, Tiella, Tonella

SBFR



<a> ::=
<b> |λ

Grammar

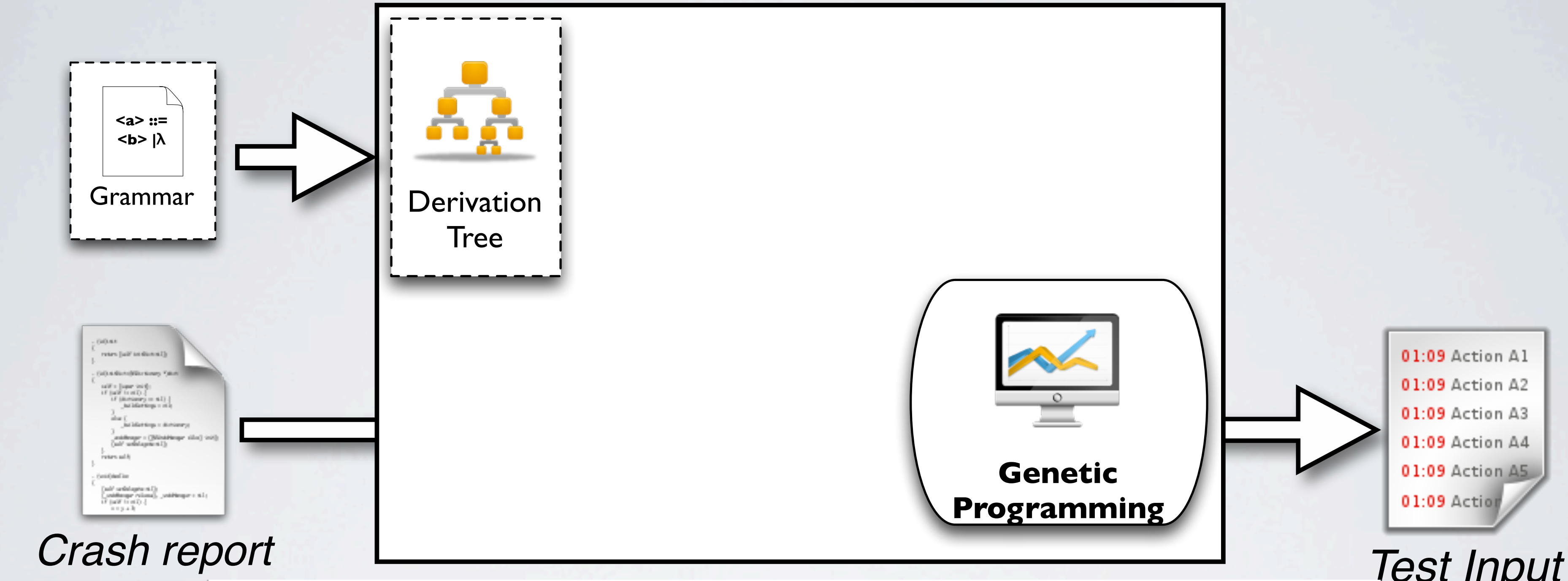
Crash report
(execution data)

01:09 Action A1
01:09 Action A2
01:09 Action A3
01:09 Action A4
01:09 Action A5
01:09 Actio

Test Input

SBFR

Joint work with
Kifetew, Jin, Tiella, Tonella



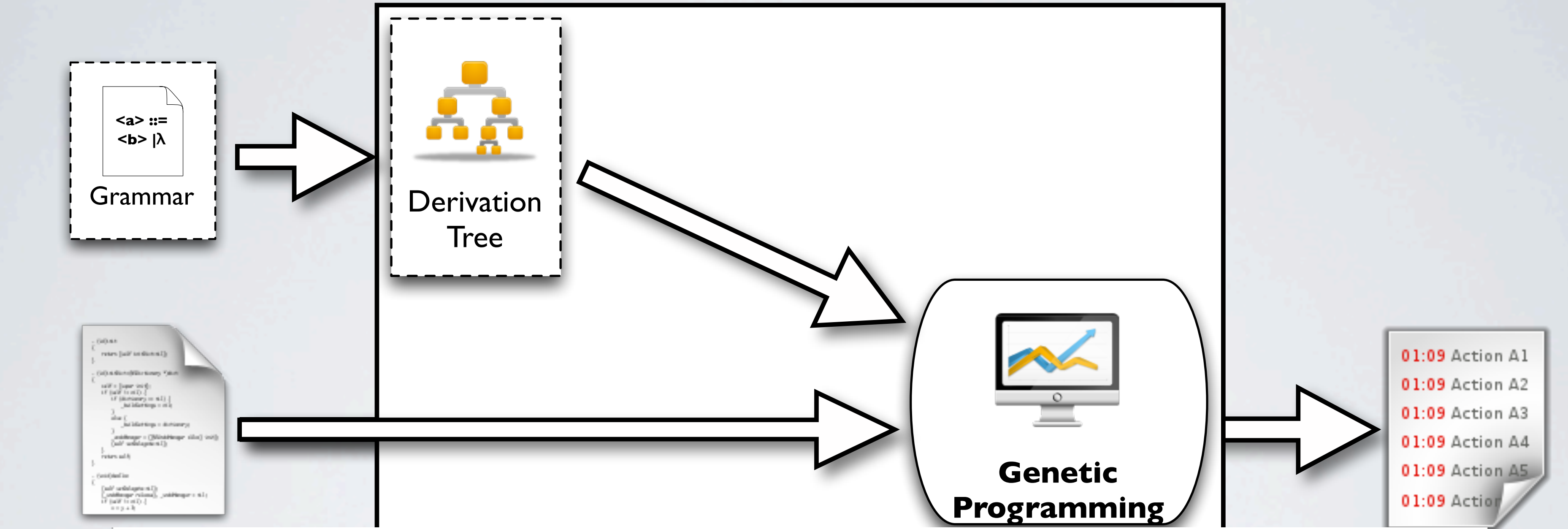
Sentence derivation from the grammar:

Random application of grammar rules

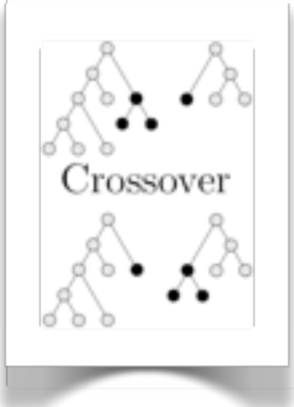
- Uniform
- 80/20
- Stochastic (from a corpus)

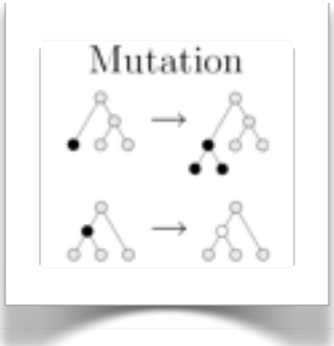
Joint work with
Kifetew, Jin, Tiella, Tonella

SBFR



Evolution:



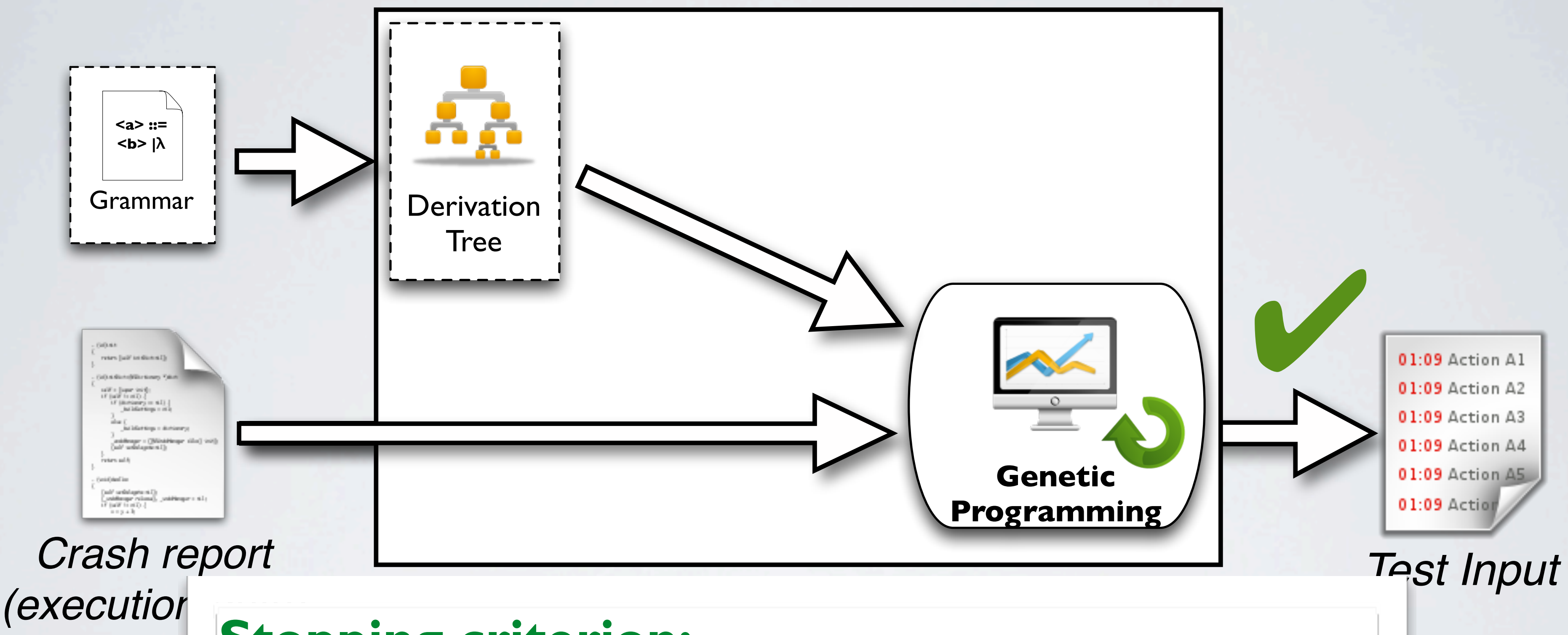


Fitness function:

Distance b/w execution traces (candidate–actual failure)

Joint work with
Kifetew, Jin, Tiella, Tonella

SBFR



Stopping criterion:

- Success
  - I_c reaches the point of failure
  - The program fails "in the same way"
- Search budget exhausted

SBFR EVALUATION – FAILURES CONSIDERED

Name	Language	Size(KLOC)	# Productions	# Faults
calc	Java	2	38	2
bc	C	12	80	1
MSDL	Java	13	140	5
PicoC	C	11	194	1
Lua	C	17	106	2

SBFR EVALUATION – FAILURES CONSIDERED

Name	Language	Size(KLOC)	# Productions	# Faults
calc	Java	2	38	2
bc	C	12	[illegible]	1
MSDL	[illegible]	[illegible]	[illegible]	5
PicoC	C	11	194	1
Lua	C	17	106	2

BugRedux was unable to reproduce any of these failures with a timeout of 72 hours

SBFR EVALUATION – RESULTS

Name	FRP (SBFR)
calc bug 1	
calc bug 2	
bc	
MSDL bug 1	
MSDL bug 2	
MSDL bug 3	
MSDL bug 4	
MSDL bug 5	
PicoC	
Lua bug 1	
Lua bug 2	

- Parameters:
  - Population: 500
  - Budget: 10,000 unique fitness evaluations
- Performed 10 runs
- Measured failure reproduction probability
- Used both 80/20 and stochastic derivations

SBFR EVALUATION – RESULTS

Name	FRP (SBFR)
calc bug 1	0.6
calc bug 2	0.8
bc	1.0
MSDL bug 1	1.0
MSDL bug 2	1.0
MSDL bug 3	1.0
MSDL bug 4	1.0
MSDL bug 5	1.0
PicoC	0.8
Lua bug 1	0.0
Lua bug 2	0.5

SBFR EVALUATION – RESULTS

Name	FRP (SBFR)	FRP (Random)
calc bug 1	0.6	0.0
calc bug 2	0.8	0.0
bc	1.0	0.0
MSDL bug 1	1.0	0.0
MSDL bug 2	1.0	0.0
MSDL bug 3	1.0	1.0
MSDL bug 4	1.0	0.0
MSDL bug 5	1.0	0.0
PicoC	0.8	0.1
Lua bug 1	0.0	0.0
Lua bug 2	0.5	0.0

SBFR EVALUATION – RESULTS

Name	FRP (SBFR)	FRP (Random)
calc bug 1	0.6	0.0
calc bug 2	0.8	0.0
bc	1.0	
MSDL		
MS		
MS		
MS		
MSD		
F		0.1
Lua bug 1	0.0	0.0
Lua bug 2	0.5	0.0

Example: failure in bc

segmentation fault triggered by an instruction sequence that allocates at least 32 arrays and declares a number of variables higher than the number of allocated arrays

SBFR EVALUATION – RESULTS

Name	FRP (SBFR)	FRP (Random)
calc bug 1	0.6	0.0
calc bug 2	0.8	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	0.1
Lua bug 1	0.0	0.0
Lua bug 2	0.5	0.0

Observations:

- Search-based approaches can be effective in cases that symbolic execution cannot handle
- Stochastic grammars are effective
- SBST more scalable, but less directed

=> SBST and DSE are complementary, rather than alternative techniques

FUTURE WORK / FOOD FOR THOUGHTS

- Relevant execution data identification
  - Which types?
  - Which specific ones?
- Failure explanation
  - Reproduction is not enough
  - Can DSE and SBST help?
- Use of different input generation techniques
  - Grammar-based symbolic execution
  - Backward symbolic analysis?
  - Other SBST approaches?
  - SBST targeted at different kinds of programs?
  - Combination of techniques