Automated Repair of Binary and Assembly Programs for Cooperating Embedded Devices

Eric Schulte [1] Jonathan DiLorenzo [2]
Westley Weimer [2] Stephanie Forrest [1]

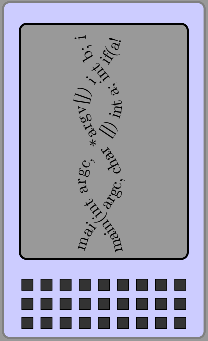
[1]Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-0001

[2]Department of Computer Science
University of Virginia
Charlottesville, VA 22904-4740

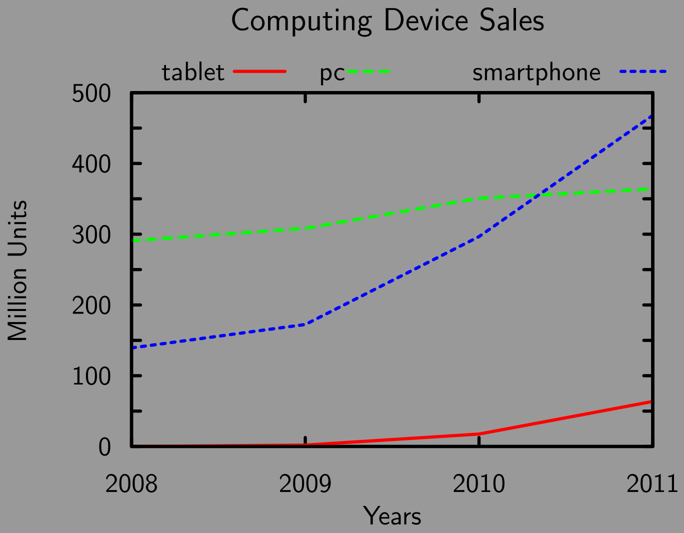
March 19, 2013

Outline

- Background
- Technical Approach
- Empirical Results
- Distributed Program Repair
- Discussion
- Conclusion



Embedded Devices



[Maier, 2011]

Embedded Devices

Resource Constraints

- Small disks
- Less memory
- Slow processors
- Slow, costly comm.

Genprog: Automatically Repairing Software Bugs

Use of algorithmic and heuristic methods to search for, generate, and evaluate program repairs.

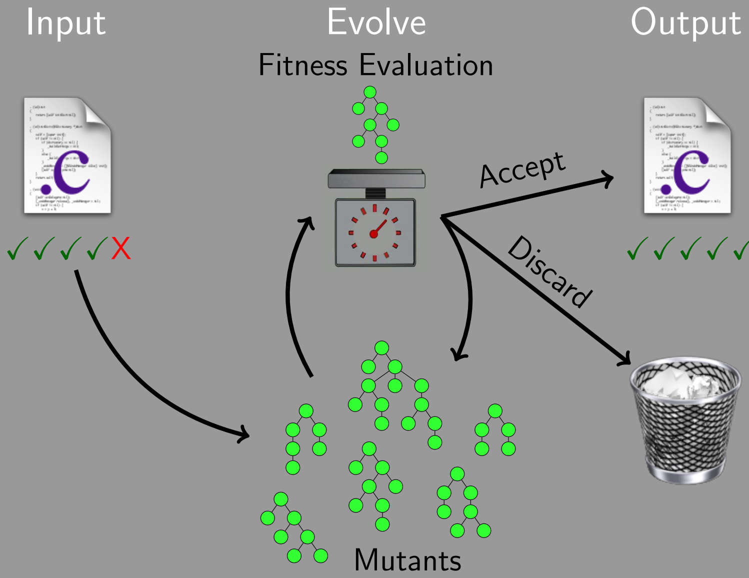
Strengths

- Repaired 55/105 bugs for $8 each [Le Goues et al., 2012a]
- Repairs multiple classes of bugs and security defects [Weimer et al., 2009]
- Wins human-competitive awards [Forrest et al., 2009; Le Goues et al., 2012b]

Limitations

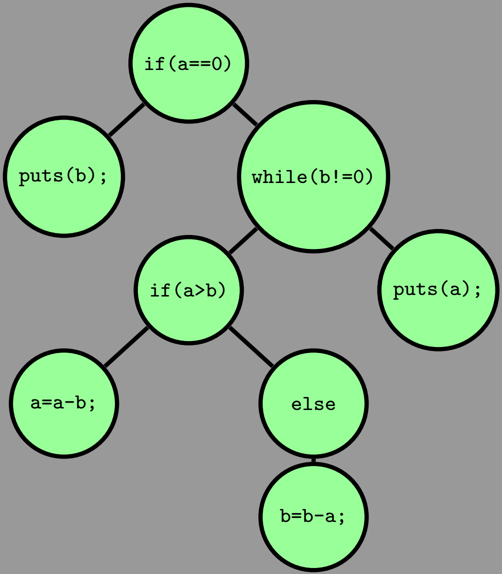
- Requires source code
- Requires build tool chain
- Requires program instrumentation
- Expensive fitness function (compilation, test execution)

Software Repair Algorithm: Baseline



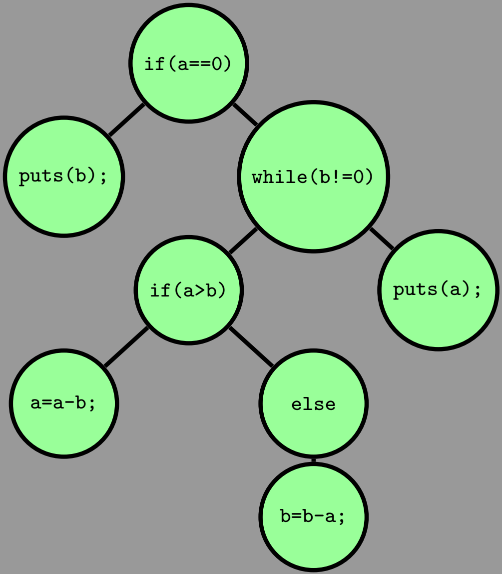
Software Repair Algorithm: Contributions

- How do we mutate?
- Where do we mutate?



Software Repair Algorithm: Contributions

- How do we mutate?
- Where do we mutate?

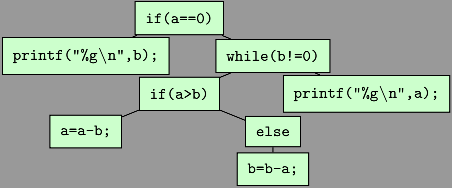


ASM and ELF Program Representations

Source

```c
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

AST

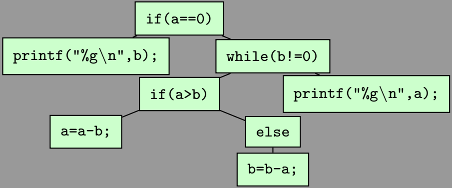


ASM and ELF Program Representations

Source

```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

AST



ASM

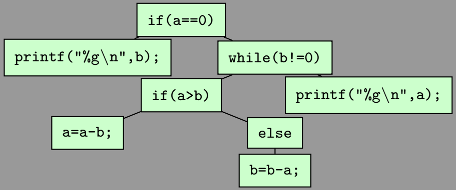
```
.file "gcd.c"
.globl main
.type main, @function
main:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $48, %rsp
```

ASM and ELF Program Representations

Source

```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else     { b=b-a; } } }
printf("%g\n", a);
```

AST



ASM

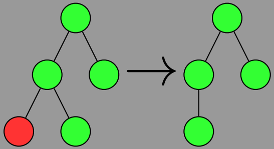
```
.file "gcd.c"
.globl main
.type main, @function
main:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $48, %rsp
```

ELF

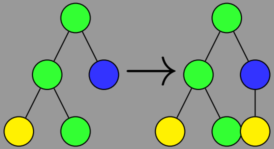
```
ELF\?
ELF header
program header table
section 1
...
.text section
[55] [48 89 e5] [48 83 ec 20]
[48 89 7d e8] [89 75 e4] [83
7d e4 01] [7e 60] ...
...
section n
section header table
```

ASM and ELF Program Mutations

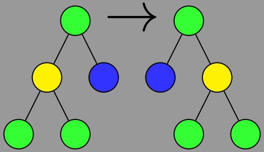
Delete



Insert



Swap



Delete:

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movq %rdx, -80(%rbp)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movl %eax, (%r15)
addq $4, %r15
```

```
movq 8(%rdx), %rdi
xorl %eax, %eax

addl $1, %r14d
call atoi
movq %rdx, -80(%rbp)
movl %eax, (%r15)
addq $4, %r15
```

Insert:

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movq -80(%rbp), %rdx
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movl %eax, (%r15)
addq $4, %r15
```

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movq -80(%rbp), %rdx
addl $1, %r14d
call atoi
movq %rdx, -80(%rbp)
movq -80(%rbp), %rdx
movl %eax, (%r15)
addq $4, %r15
```

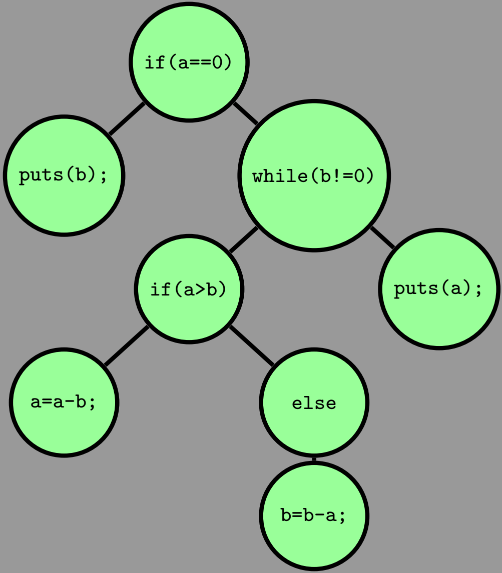
Swap:

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movq %rdx, -80(%rbp)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movl %eax, (%r15)
addq $4, %r15
```

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movq -80(%rbp), %rdx
addl $1, %r14d
call atoi
movq %rdx, -80(%rbp)
movl %eax, (%r15)
addq $4, %r15
```

(... additional ELF bookkeeping ...)

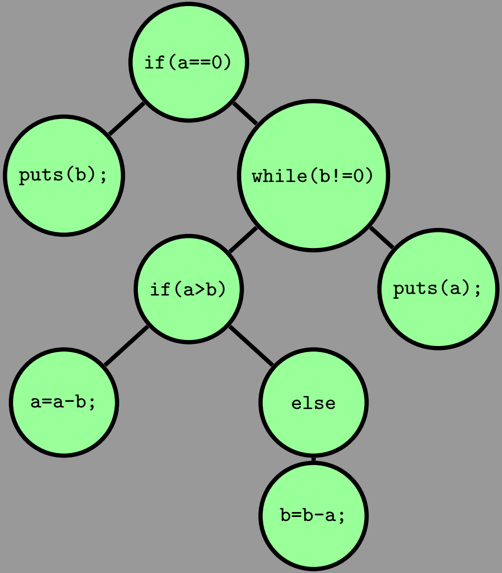
Genprog: Automatically Repairing Software Bugs



- How do we mutate?
- Where do we mutate?

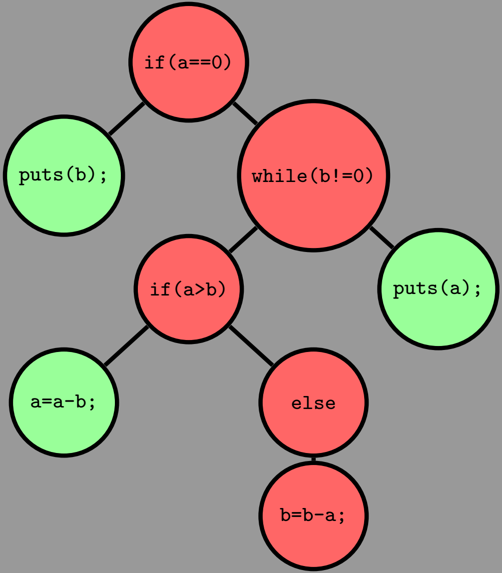
Genprog: Automatically Repairing Software Bugs

- How do we mutate?
- Where do we mutate?



Genprog: Automatically Repairing Software Bugs

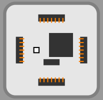
- How do we mutate?
- Where do we mutate?



Fault Localization

Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



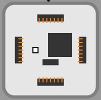
CPU

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```

Machine-code Instructions

Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



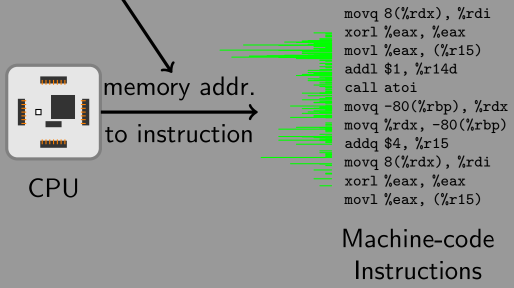
CPU

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```

Machine-code
Instructions

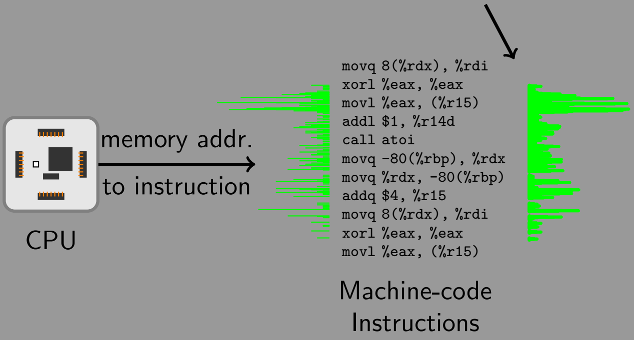
Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.

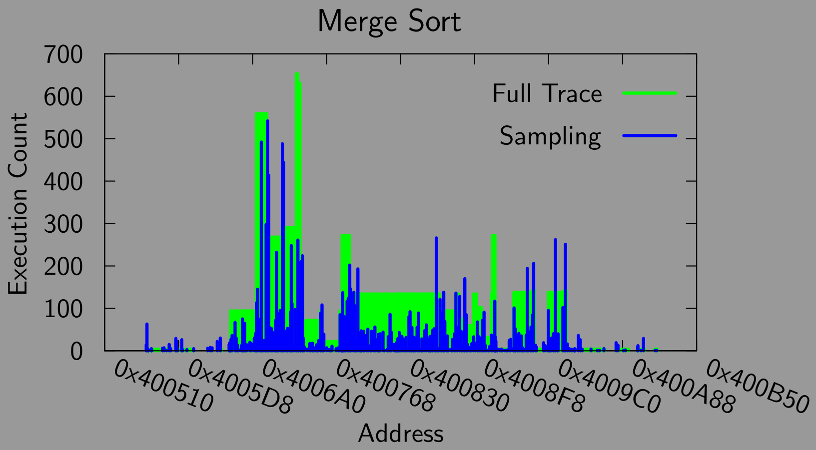


Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



Fault Localization Comparison



Genprog: Automatically Repairing Software Bugs

Use of algorithmic and heuristic methods to search for, generate, and evaluate program repairs.

Strengths

- Repaired 55/105 bugs for $8 each [Le Goues et al., 2012a]
- Repairs multiple classes of bugs and security defects [Weimer et al., 2009]
- Wins human-competitive awards [Forrest et al., 2009; Le Goues et al., 2012b]

Limitations

- Requires source code
- Requires build tool chain
- Requires program instrumentation
- Expensive fitness function (compilation, test execution)

Genprog: Automatically Repairing Software Bugs

Use of algorithmic and heuristic methods to search for, generate, and evaluate program repairs.

Strengths

- Repaired 55/105 bugs for $8 each [Le Goues et al., 2012a]
- Repairs multiple classes of bugs and security defects [Weimer et al., 2009]
- Wins human-competitive awards [Forrest et al., 2009; Le Goues et al., 2012b]

Limitations

- ~~Requires source code~~
- ~~Requires build tool chain~~
- ~~Requires program instrumentation~~
- Expensive fitness function (~~compilation~~, test execution)

Benchmark Programs

Program	Program Description	Bug
`atris`	graphical tetris game	local stack buffer exploi
`ccrypt`	encryption utility	segfault
`deroff`	document processing	segfault
`flex`	lexical analyzer generator	segfault
`indent`	source code processing	infinite loop
`look svr4`	dictionary lookup	infinite loop
`look ultrix`	dictionary lookup	infinite loop
`merge`	merge sort	duplicate inputs
`merge-cpp`	merge sort (in C++)	duplicate inputs
`s3`	sendmail utility	buffer overflow
`uniq`	duplicate text processing	segfault
`units`	metric conversion	segfault
`zune`	embedded media player	infinite loop

Empirical Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Total bugs repaired

Rep.	Num. Bugs
AST	13
ASM	12
ELF	11

Average success rate

100 runs per bug

Rep.	Success Rate
AST	78.17%
ASM	70.75%
ELF	65.83%

Empirical Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Expected fitness evaluations

Rep.	Evaluations
AST	583.98
ASM	188.38
ELF	207.15

Total runtime

Rep.	Sec.
AST	229.50
ASM	278.30
ELF	74.20

Empirical Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Example: Merge Sort

Repair by representation

- AST, 2 of 4900 Swaps
- ASM, 1 of 280 Deletes

merge.c

```
if(left[l-mid-1]<=right[0]){
  result=list; }
else{/* fix:swap branches */
  result=merge(left,l-mid,
                    right,mid); }
```

merge.s

```
cmpl %eax, %edx ; fix: del.
jg   .L12
movq -72(%rbp), %rax
```

Empirical Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Disk size

Rep.	Requirements
AST	Source code & build toolchain
ASM	Assembly code & linker
ELF	Compiled executable

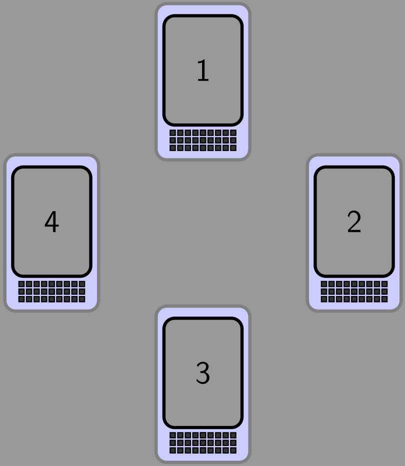
Empirical Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

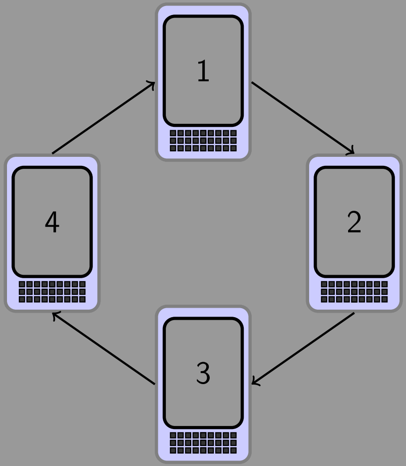
Working memory

Rep.	MB
AST	1402
ASM	756
ELF	200

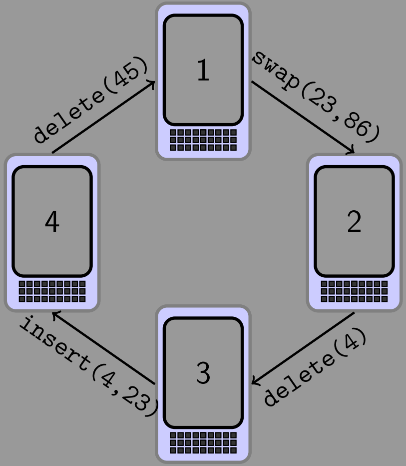
Distributed Genetic Repair Algorithm



Distributed Genetic Repair Algorithm



Distributed Genetic Repair Algorithm



Distributed Genetic Repair Evaluation

Relative performance of DGA

# Nodes	Expected Fitness Evaluations	Wall Clock Seconds	w/SMS
1	1	1	1
2	0.94	0.89	1.07
3	0.84	0.67	0.81
4	0.80	0.55	0.63

Distributed Genetic Repair Evaluation

Relative performance of DGA

# Nodes	Expected Fitness Evaluations	Wall Clock Seconds	w/SMS
1	1	1	1
2	0.94	0.89	1.07
3	0.84	0.67	0.81
4	0.80	0.55	0.63

Distributed Genetic Repair Evaluation

Relative performance of DGA

# Nodes	Expected Fitness Evaluations	Wall Clock Seconds	w/SMS
1	1	1	1
2	0.94	0.89	1.07
3	0.84	0.67	0.81
4	0.80	0.55	0.63

Discussion

- Search space
- System protection

ASM & ELF search space

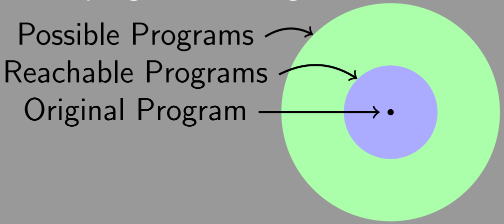
Program size

$\approx 3\times$ more assembly instructions than C statements

Search space size

$= |\text{alphabet}|^{\text{program size}}$

Possible program coverage



Discussion

- Search space
- System protection

System protection

- Arbitrary assembly is arbitrarily dangerous
- Light weight sandboxing (`ulimit` and `chroot`)

Conclusion

ASM and ELF representation

- Remove requirement for source code and build toolchain
- Language Agnostic; x86 or ARM assembly or ELF
- Change program repair search space
- Reduce resources; 95% smaller disk footprint, 86% less memory, 62% faster runtime

Distributed Genetic Program Repair

- Allows multiple devices to collaborate
- Fewer fitness evaluations and faster runtime

Conclusion

Take away

- Assembly mutations cause semantic changes!
- Software is not brittle!

Conclusion

Take away

- Assembly mutations cause semantic changes!
- Software is not brittle!

Thank You

Contact

Name	Eric Schulte
Email	`eschulte@cs.unm.edu`
Homepage	`http://cs.unm.edu/~eschulte`

Code

Program Repair Tool	`http://genprog.cs.virginia.edu`
`ptrace` Tracer	`http://github.com/eschulte/tracer`
ELF (*C*)	`http://github.com/eschulte/rw-elf`
ELF (*Common Lisp*)	`http://github.com/eschulte/elf`

Bibliography

S. Forrest, W. Weimer, T. Nguyen, and C. Le Goues. A genetic programming approach to automated software repair. In Genetic and Evolutionary Computing Conference, 2009.

C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In International Conference on Software Engineering, 2012a.

C. Le Goues, W. Weimer, and S. Forrest. Representations and operators for improving evolutionary software repair. In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, pages 959–966. ACM, 2012b.

D. Maier. Sales of smartphones and tablets to exceed pcs. Practical Ecommerce, October 2011. `http://www.practicalecommerce.com/articles/3069-Sales-of-Smartphones-and-Tablets-to-Exceed-PCs-`.

W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In International Conference on Software Engineering, pages 364–367, 2009.

Backup Slides

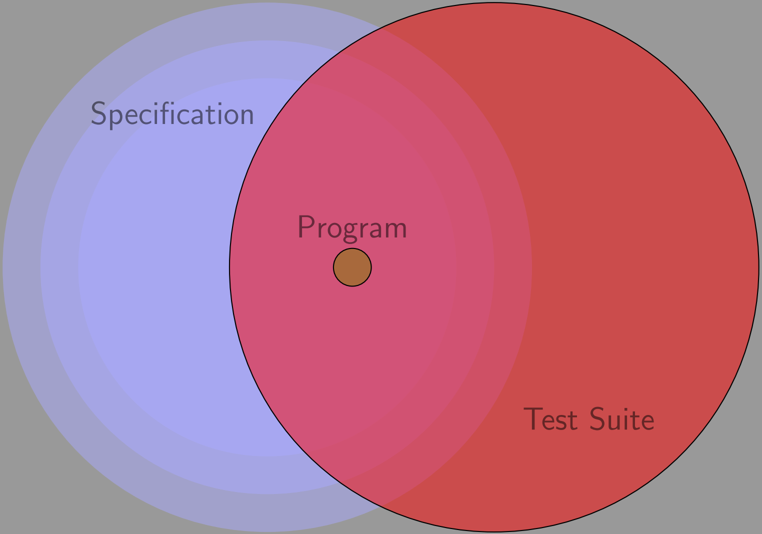
Backup: Wall Clock Times

Mean wall clock time in seconds to find a successful repair

	DGA		Naïve Parallel
# Nodes	Seconds	Rounds	Seconds
1			205.531
2	173.868	43.2	195.821
3	135.17	28.2	201.346
4	115.566	14.5	211.989

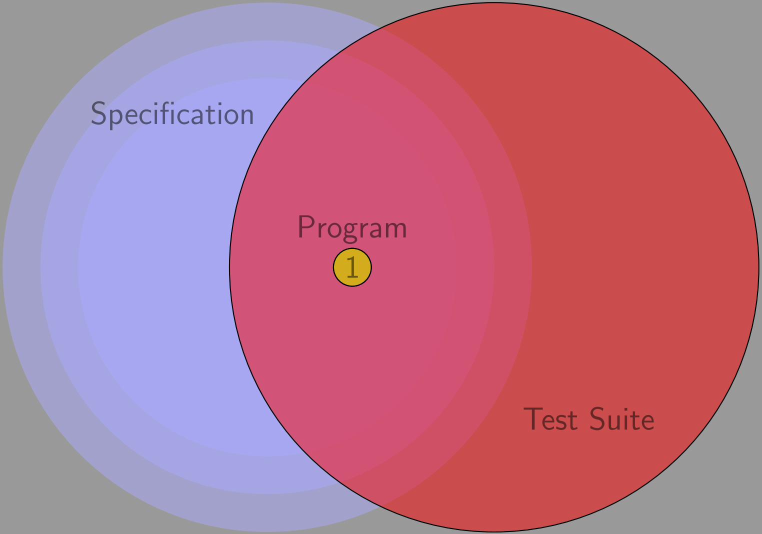
Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite



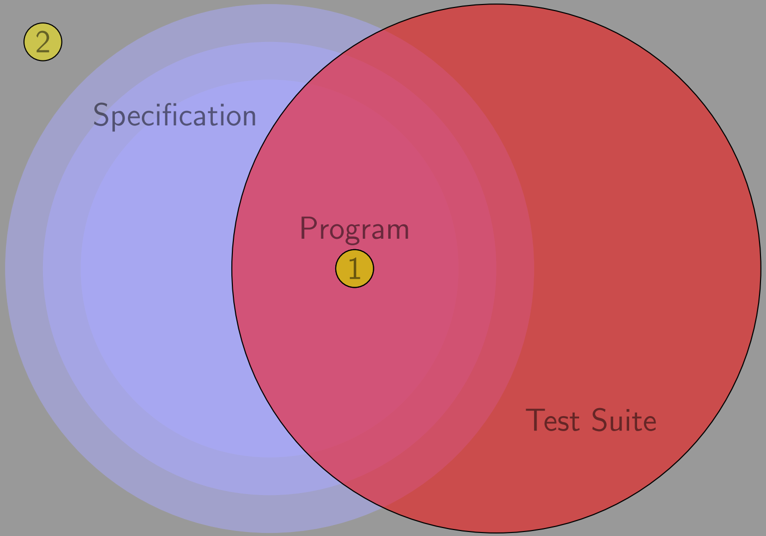
Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite



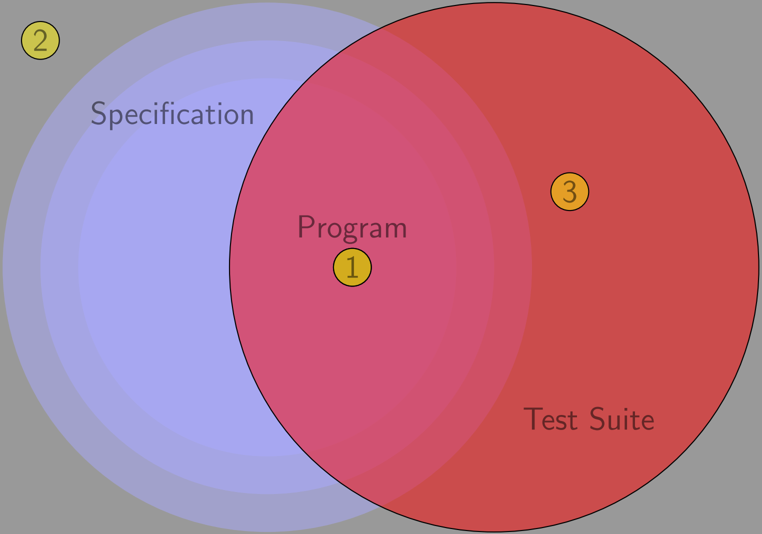
Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite



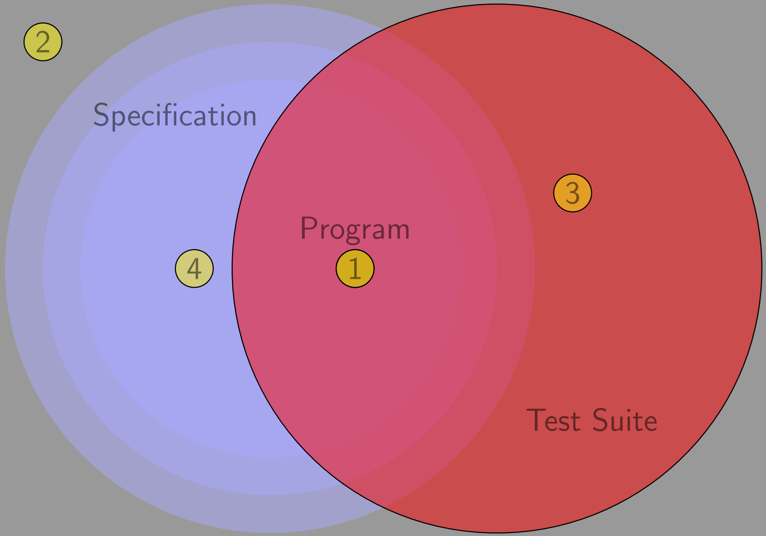
Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite



Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite



Backup: Software Mutational Robustness

Semantic Space, Specification & Test Suite