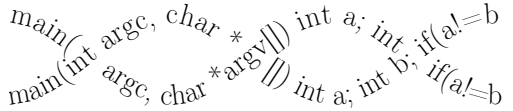


Dissertation Proposal: The Evolution of Robust Software

Eric Schulte

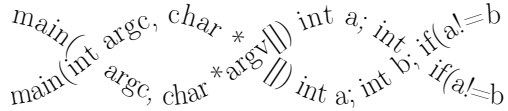
University of New Mexico

August 15, 2012

Outline

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Natural Selection of Software



Insight

Over the past 50 years developers and engineers have **selected**, **copied**, **modified**, and **pasted** software tools, environments and code in a process resembling **natural selection**.

Research Questions

- What effects have these processes had on software?
- Can these insights lead to new tools and techniques?

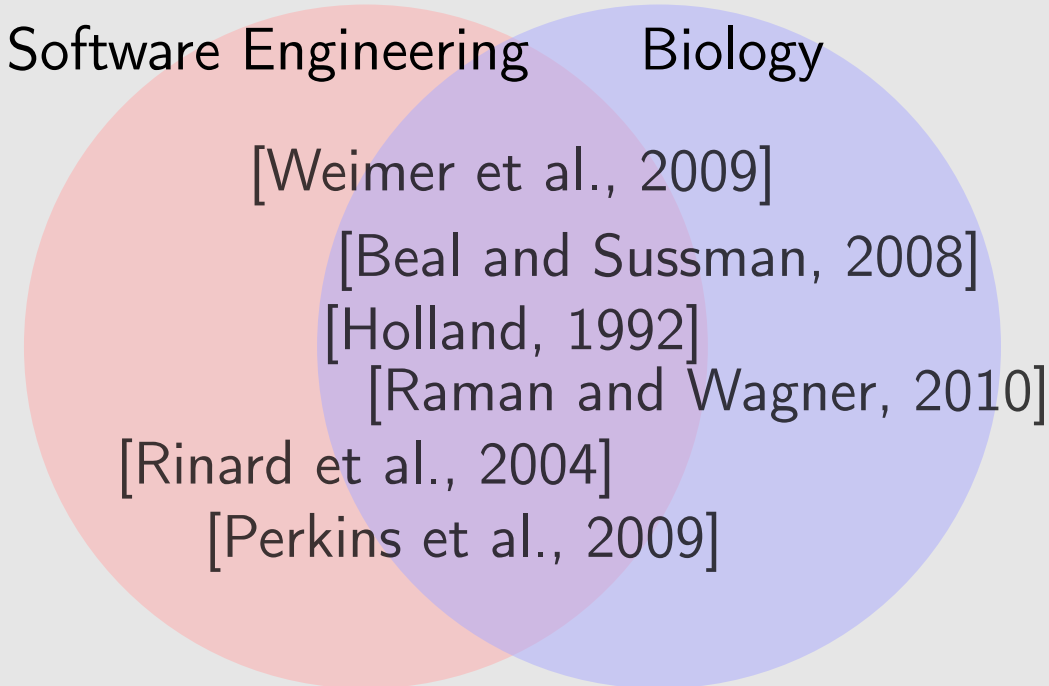
Motivation

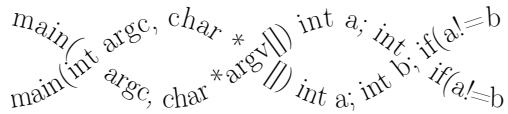
main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Need

- 1.3 Million employed software developers in the US in 2008.
- Expect 21% increase by 2018 [Bureau of Labor Statistics, 2011].

Opportunity





Biological Robustness

Genotype and Phenotype

Realms

genotype genetic information which specifies an organism

phenotype resulting physical organism in the world

Robustness

genotypic (*mutational*) robustness to internal variance in genetic information

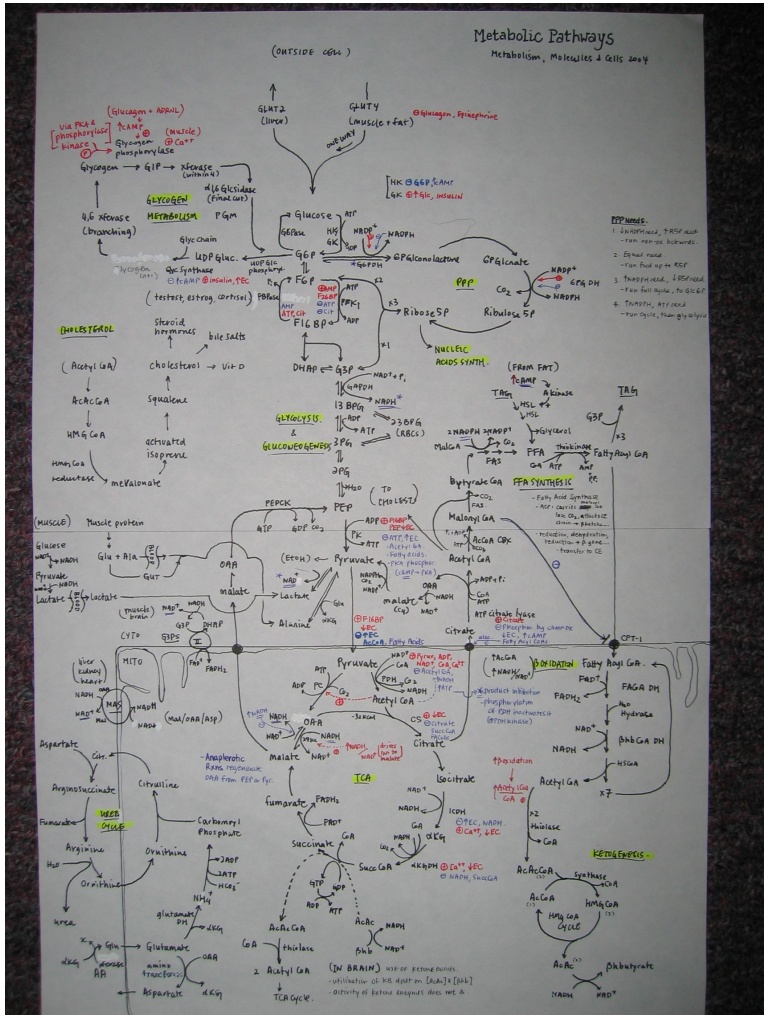
phenotypic (*environmental*) robustness to external variance in the world

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Biological Robustness

Mechanisms of robustness

- Important amino acids over represented [Knight et al., 1999]
- Buffer changes (e.g., metabolic pathways) [Wagner, 2005]
- Degenerate vital functions [Edelman and Gally, 2001]



(from http://www.raymondcheong.com/)

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Biological Robustness

Robustness and Evolution

- Evolution finds large *neutral spaces*
- Evolution increases mutational robustness [Van Nimwegen et al., 1999]
- Accrued genetic information leads to evolutionary innovation [Ciliberti et al., 2007]

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Evolutionary Computation

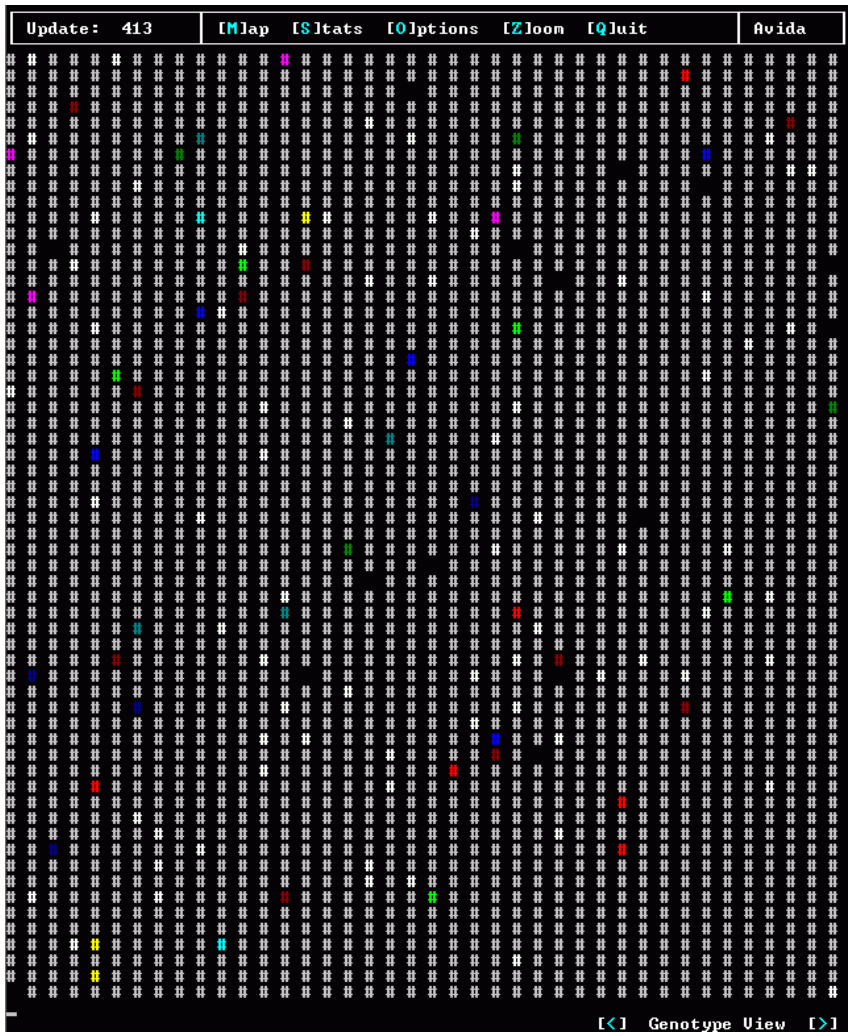
Digital Evolution

Computational simulation of adaptive systems [Holland, 1962].

Avida

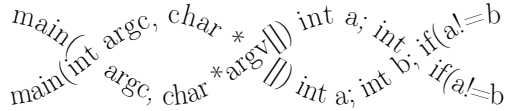
[Ofria and Wilke, 2004]

- computational models which *mimic* evolved biological systems
- allow time frames and controls impossible in-situ

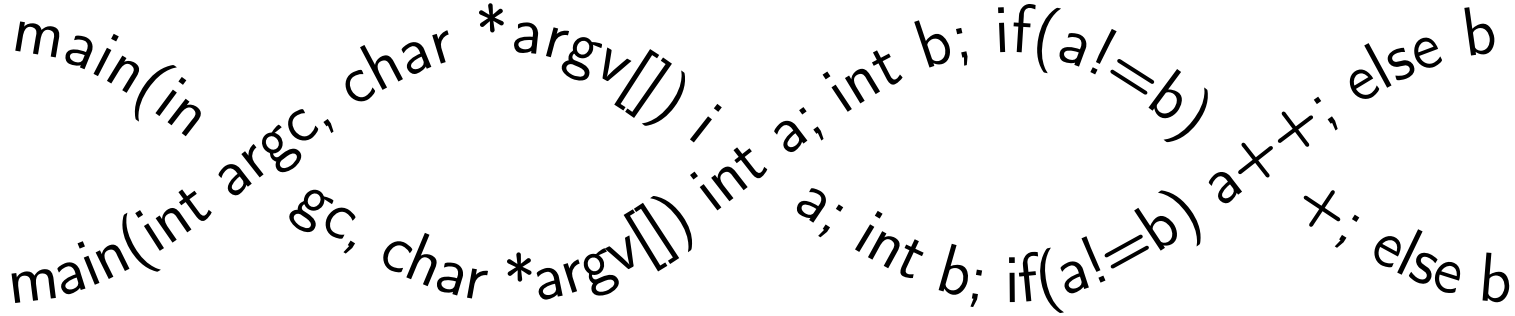


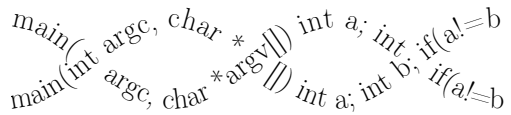
Evolutionary Computation

Genetic Programming



- Natural selection as a heuristic used to program computers
- Simplified programming languages [Koza, 1992]
- Used to repair real-world extant software written by humans



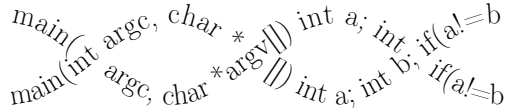


Software Engineering

Acceptably Correct Computation

- Ignore memory errors
  [Rinard et al., 2004]
- Hallucinate acceptable inputs
  [Beal and Sussman, 2008]
- Dynamically enforce learned invariants
  [Perkins et al., 2009]
- Drop loop executions at runtime
  [Misailovic et al., 2011]

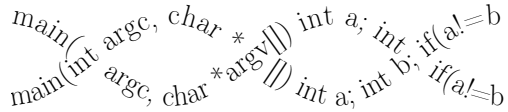
Software Mutational Robustness



Software Mutational Robustness:

The fraction of mutants of software that are functional.

Software Mutational Robustness



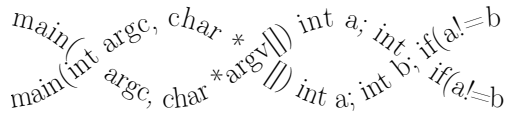
Software Mutational Robustness:

The fraction of mutants of software that are functional.

Depends upon:

- The software itself

Software Mutational Robustness



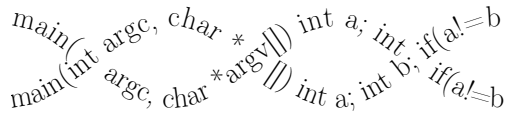
Software Mutational Robustness:

The fraction of mutants of software that are functional.

Depends upon:

- The software itself
- The mutation operators

Software Mutational Robustness



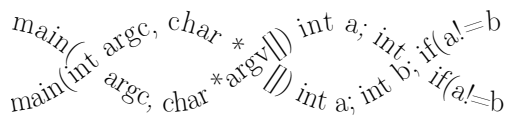
Software Mutational Robustness:

The fraction of mutants of software that are functional.

Depends upon:

- The software itself
- The mutation operators
- The test of functionality

Software Mutational Robustness



Software Mutational Robustness:

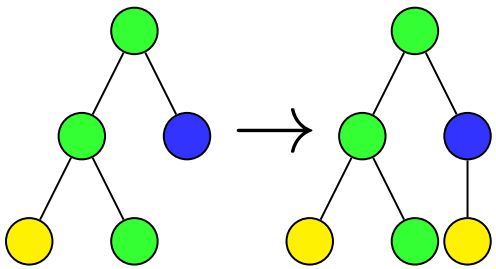
The fraction of mutants of software that are functional.

Independent of:

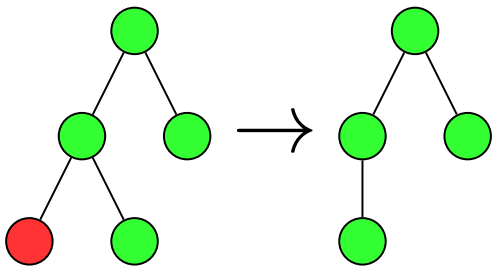
- The software itself
- The mutation operators
- The test of functionality

Software Mutational Robustness

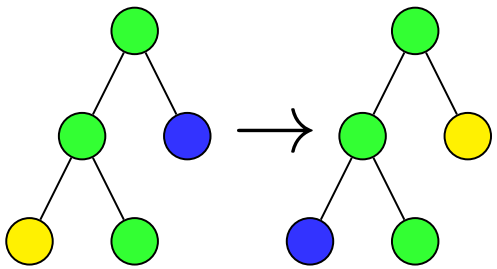
Mutation Operators



(a) Insert AST



(b) Delete AST



(c) Swap AST

```
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
```

(d) Insert ASM

```
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
```

(e) Delete ASM

```
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
```

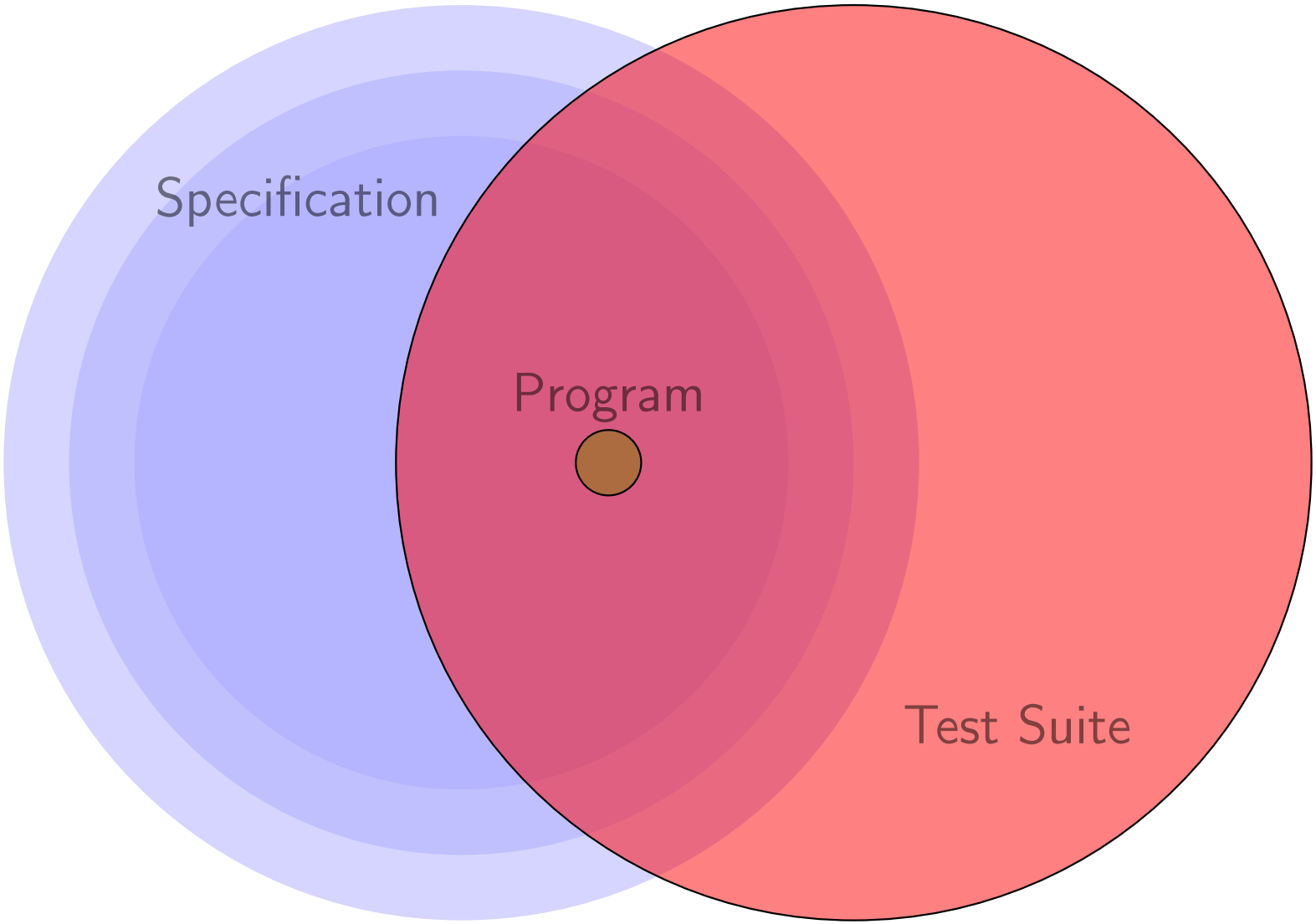
(f) Swap ASM

(*ELF level mutation operators not shown*)

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

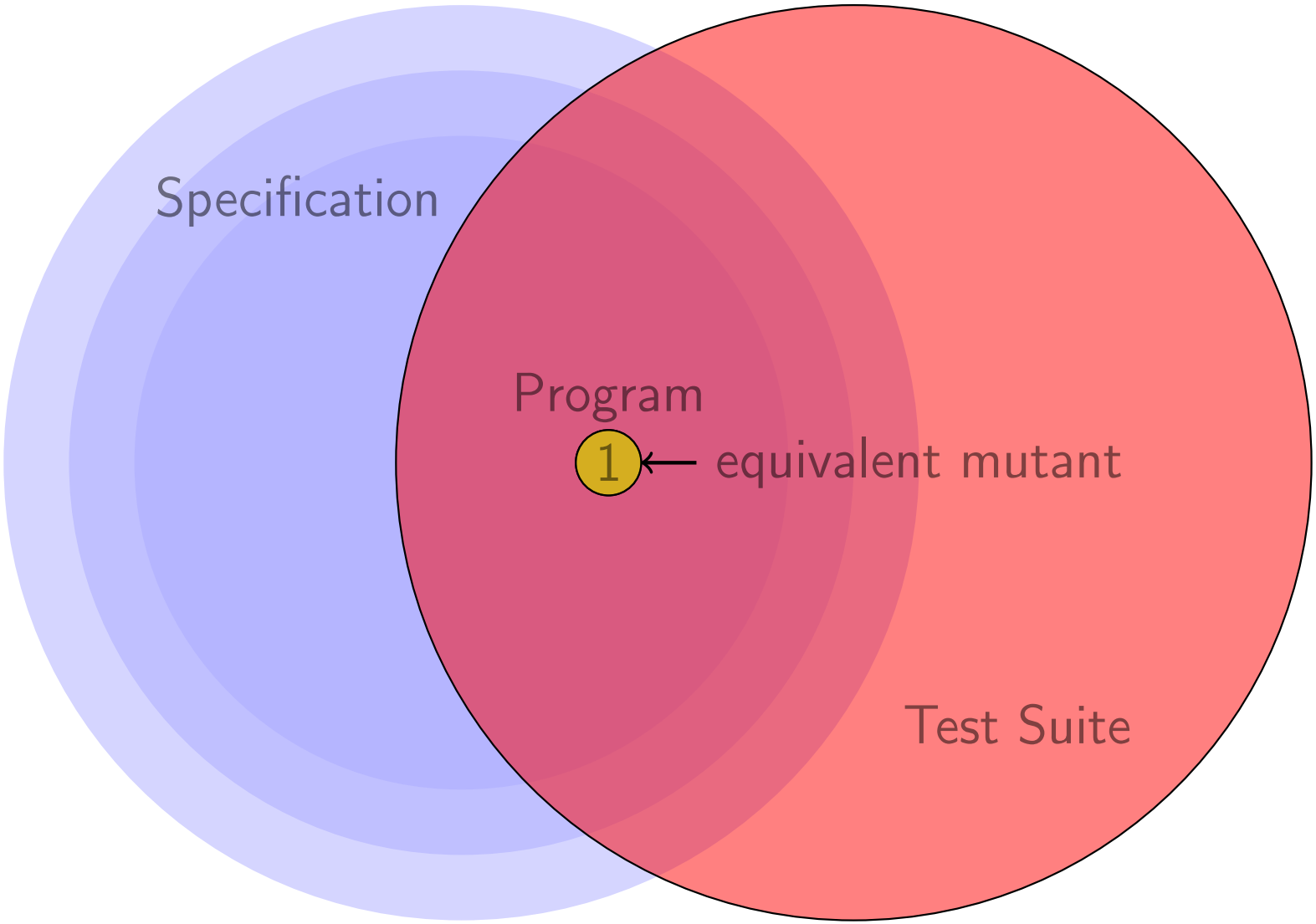
Software Mutational Robustness

Semantic Space & Mutation Testing



Software Mutational Robustness

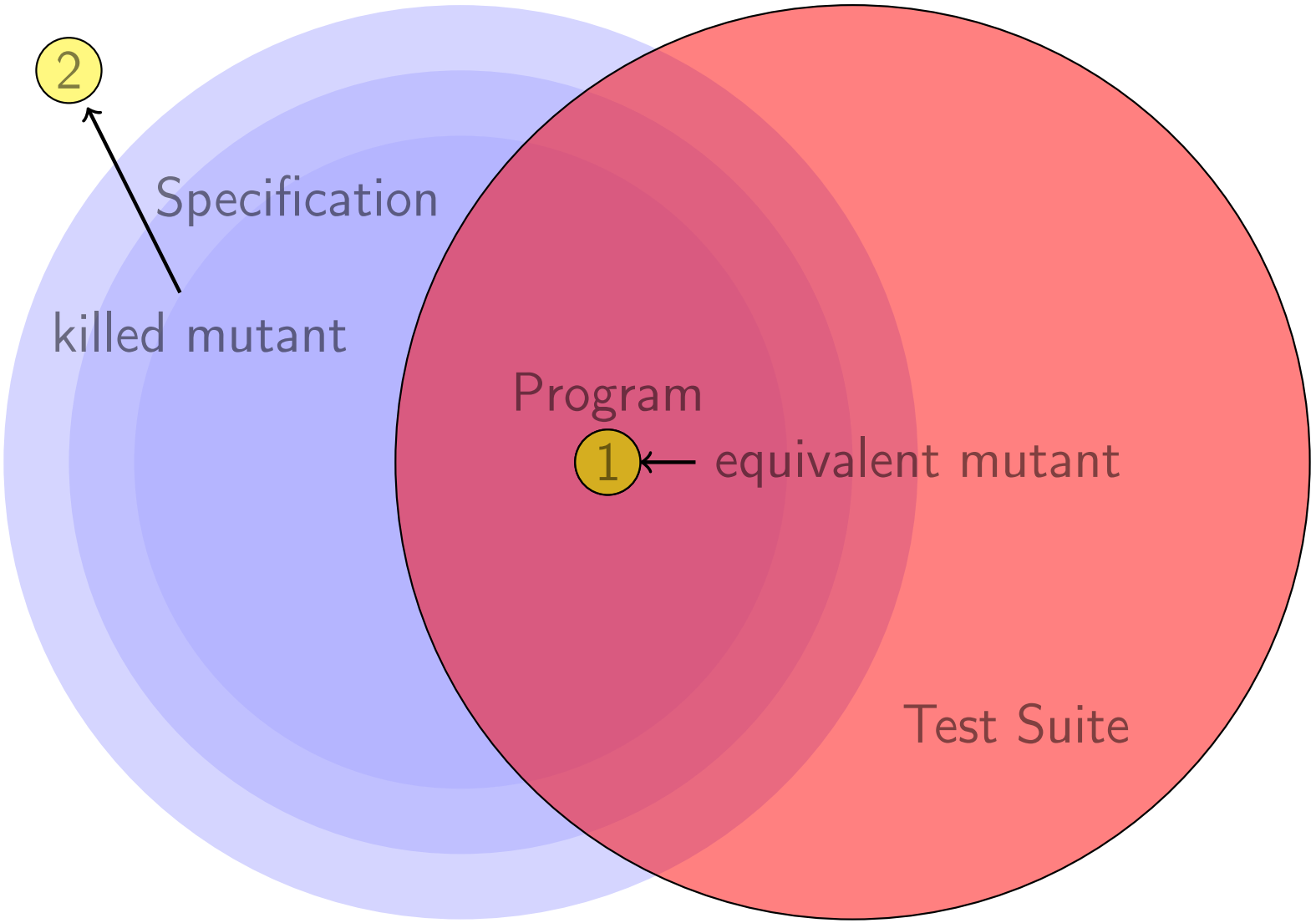
Semantic Space & Mutation Testing



main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Mutational Robustness

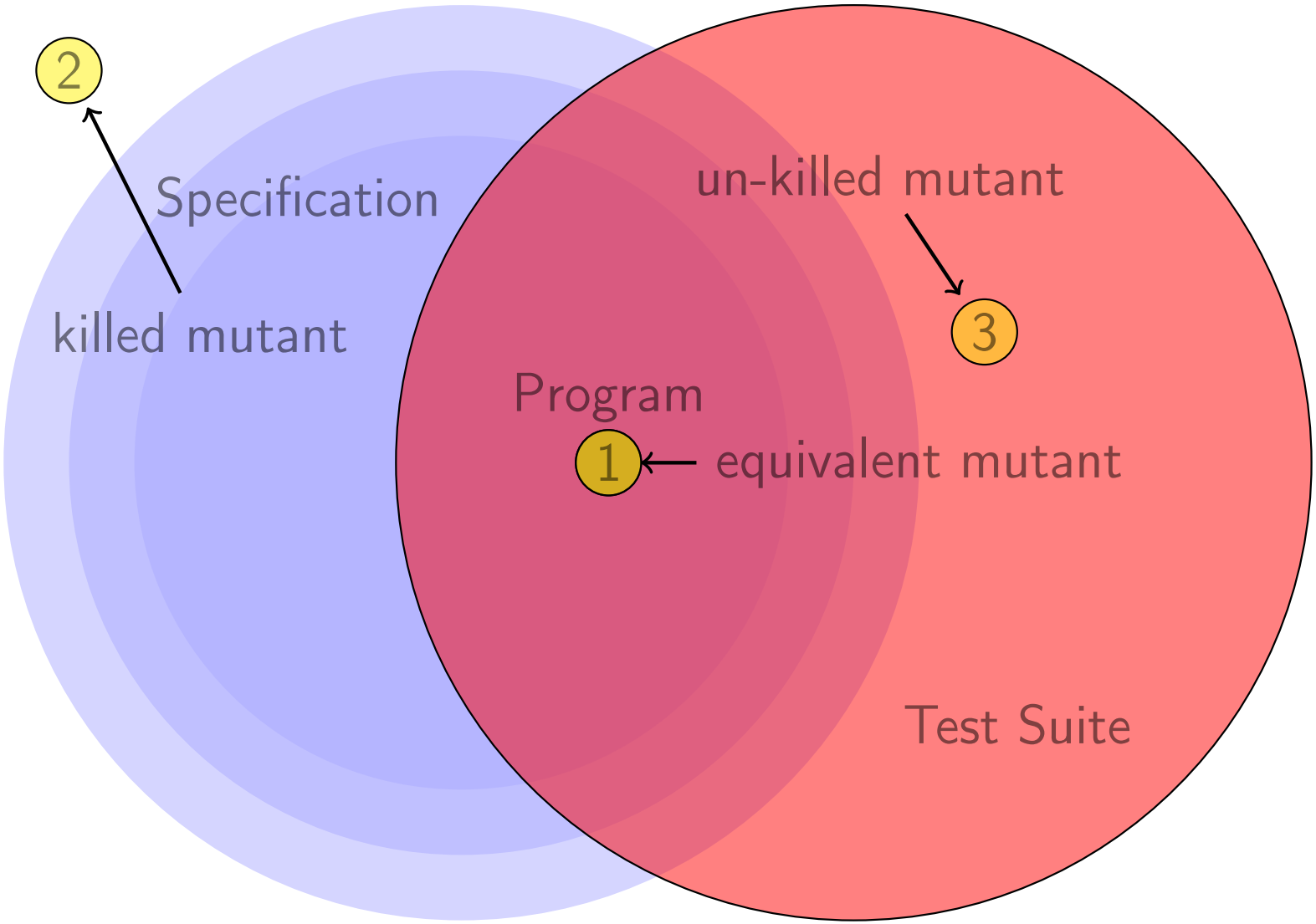
Semantic Space & Mutation Testing



main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Mutational Robustness

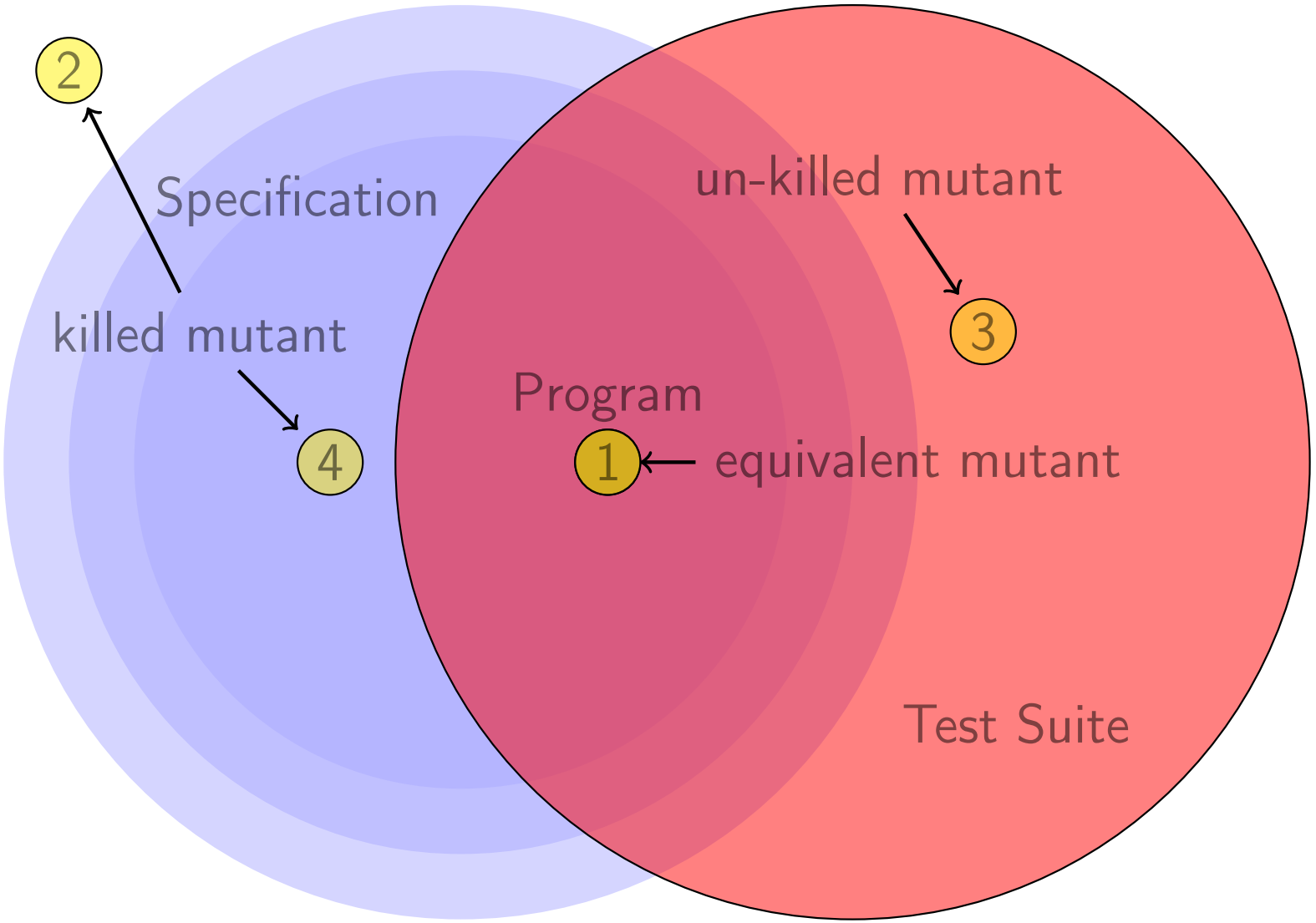
Semantic Space & Mutation Testing



main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Mutational Robustness

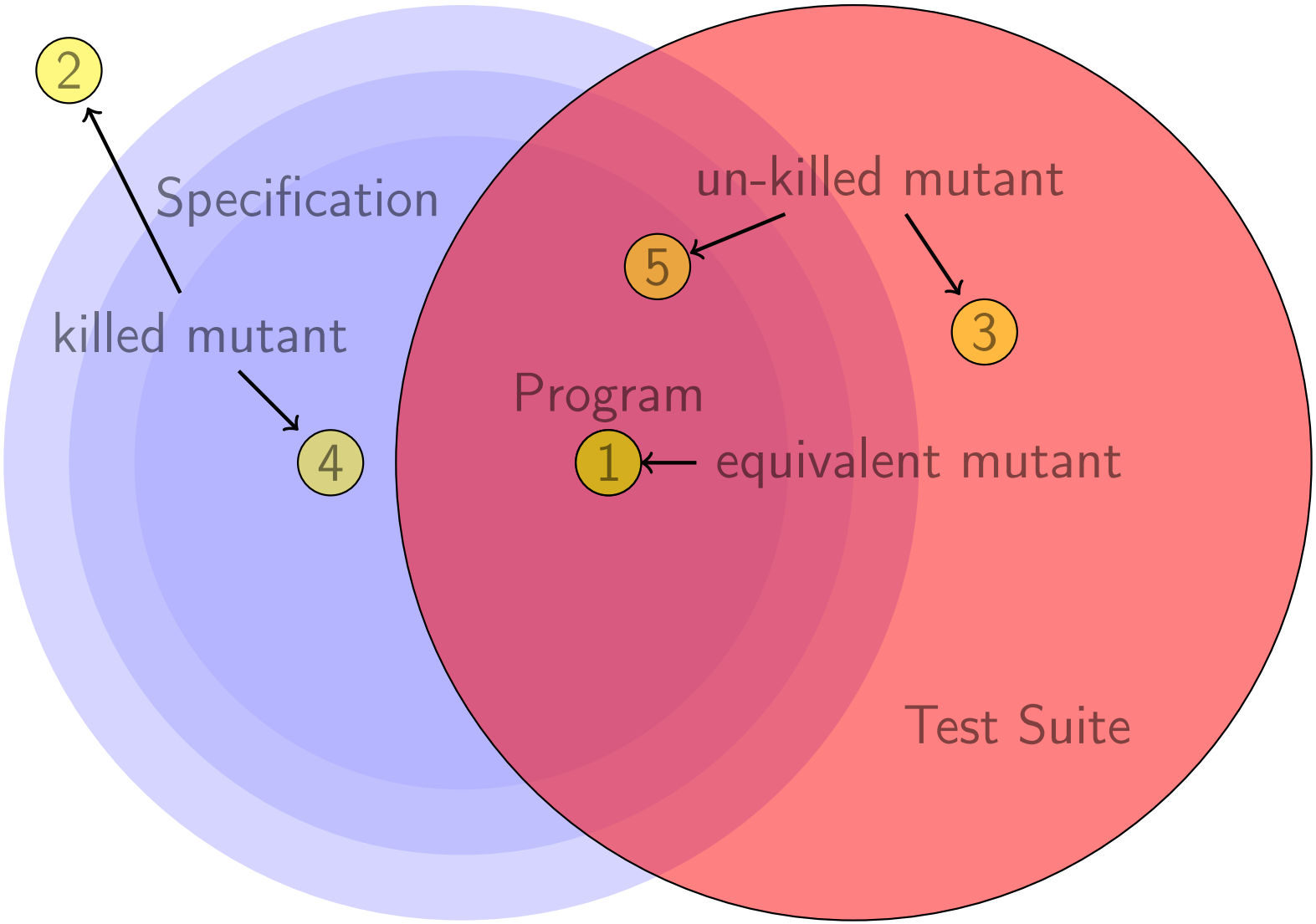
Semantic Space & Mutation Testing



main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Mutational Robustness

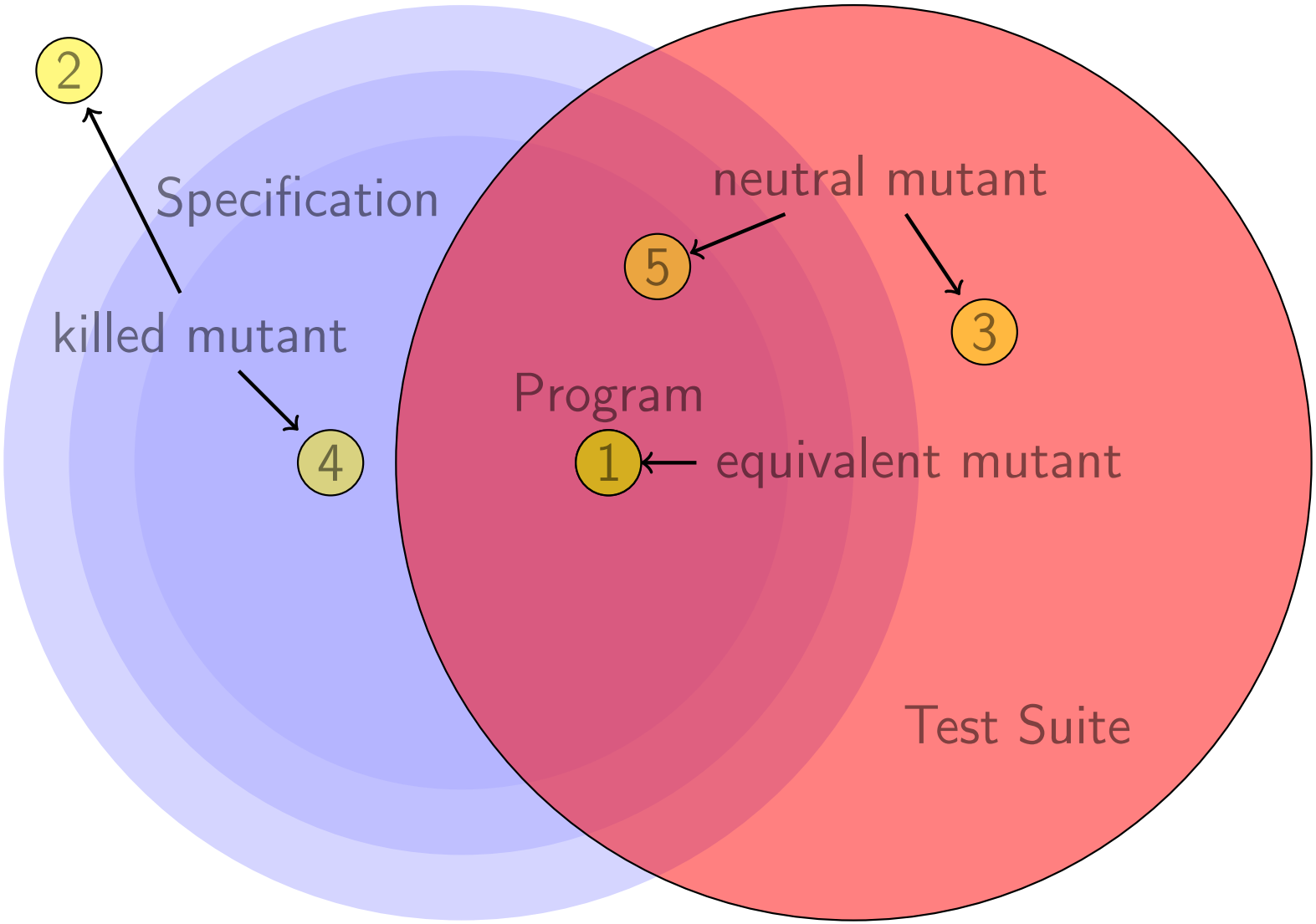
Semantic Space & Mutation Testing



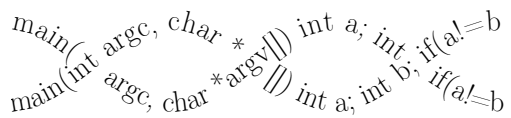
main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Mutational Robustness

Semantic Space & Mutation Testing



Software Mutational Robustness



Benchmarks

Sorting Programs

- bubble
- insertion
- merge
- quick

Siemens Benchmark

- grep
- printtokens
- schedule
- sed
- space
- tcas

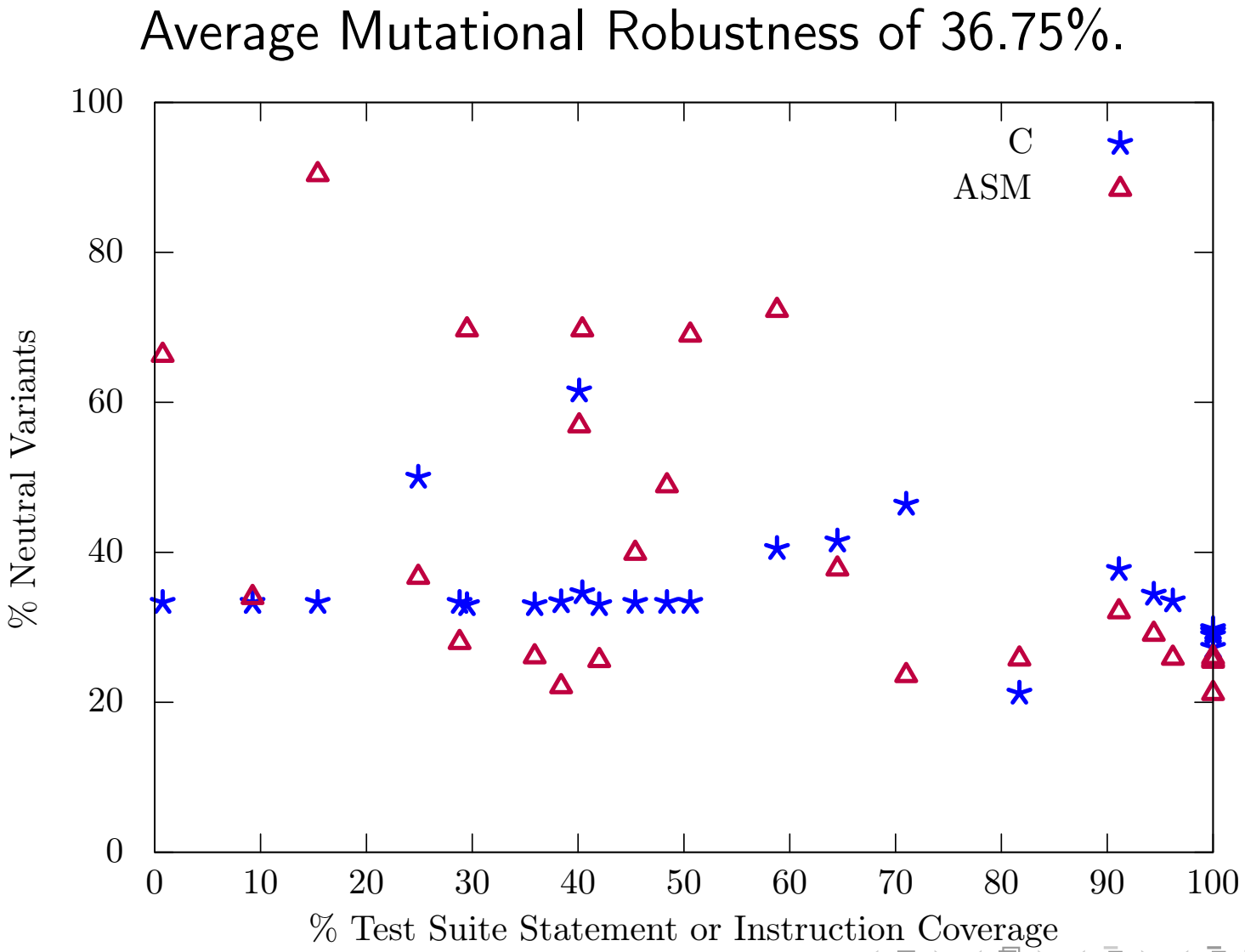
Real Programs

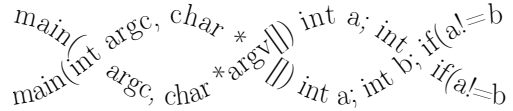
- bzip2 1.0.2 (& alt. suite)
- ccrypt 1.2 (& alt. suite)
- imagemagick 6.5.2
- jansson 1.3
- leukocyte
- lighttpd 1.4.15
- nullhttpd 0.5.0
- oggenc 1.0.1 (& alt. suite)
- potion 40b5f03
- redis 1.3.4
- tiff 3.8.2
- vyquon 335426d

Software Mutational Robustness

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Preliminary Results





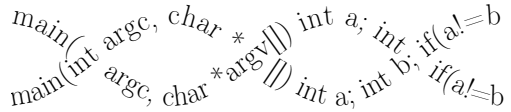
Causes of Mutational Robustness

Level: Source VS. Compiled

Hypothesis: Compilation and linking increase software robustness.

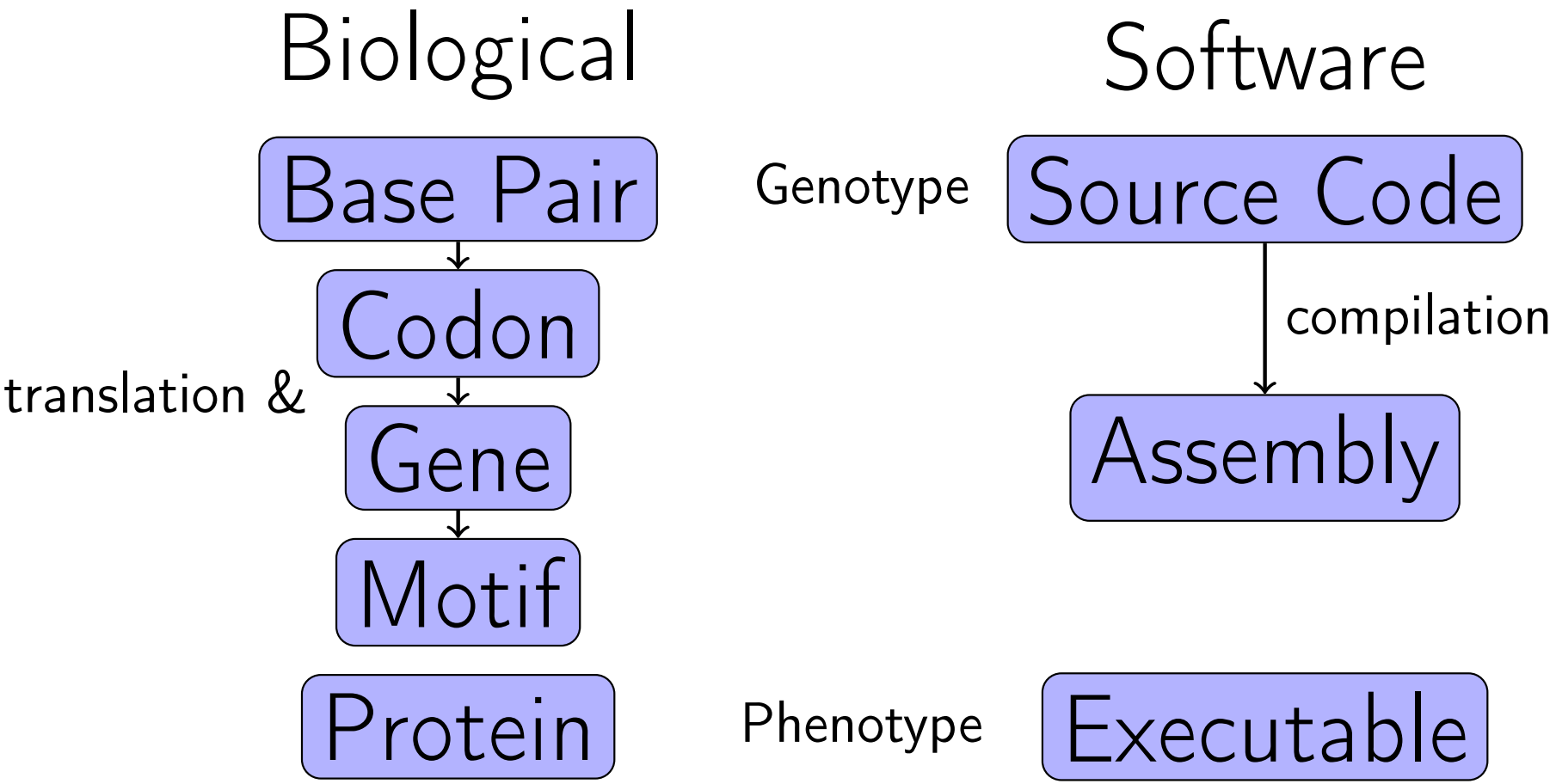
Biological		Software
Base Pair	Genotype	Source Code
Codon		
Gene		Assembly
Motif		
Protein	Phenotype	Executable

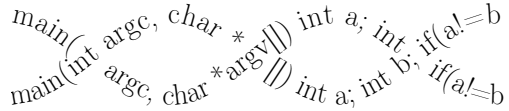
Causes of Mutational Robustness



Level: Source VS. Compiled

Hypothesis: Compilation and linking increase software robustness.

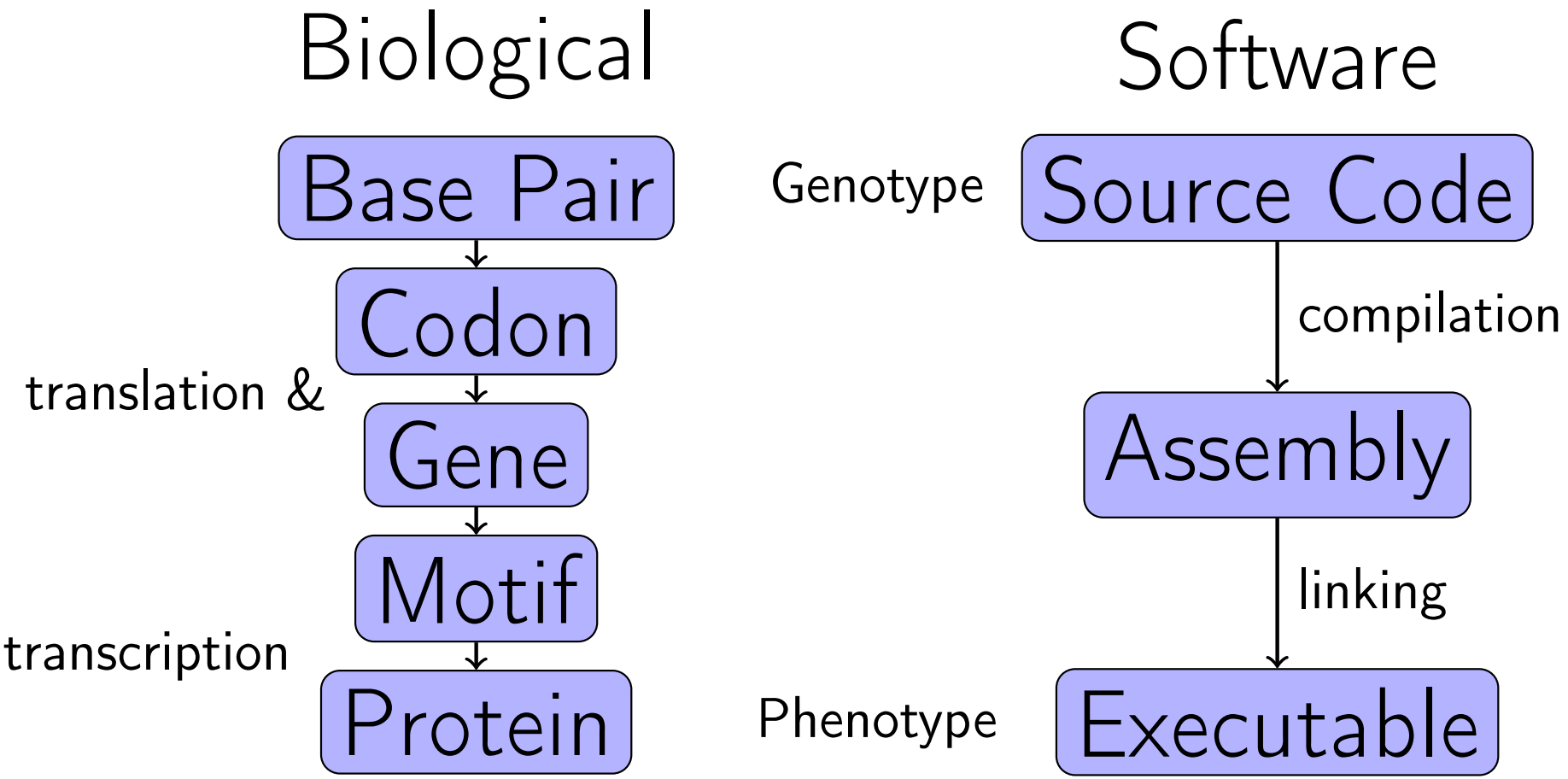


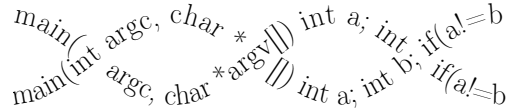


Causes of Mutational Robustness

Level: Source VS. Compiled

Hypothesis: Compilation and linking increase software robustness.





Causes of Mutational Robustness

Provenance: Evolved VS. Engineered

Hypothesis: Evolution increases mutational robustness and evolvability.

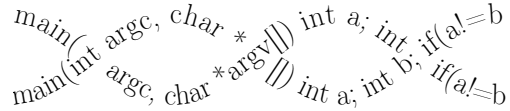
Compare three classes of software

1. Those programmed entirely by human engineers.
2. Those programmed initially by human engineers and then incrementally evolved.
3. Those programmed entirely through an evolutionary process.

Test for

- mutational robustness
- evolvability

Correlates of Mutational Robustness



Evolvability

Are mutational robustness and evolvability correlated in software?

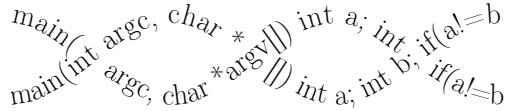
Requirements

- metric of evolvability
- metric of mutational robustness

Examples

- what level of mutational robustness is desirable in software
- when beneficial (e.g., maintaining critical behavior)
- when detrimental (e.g., rapidly changing specification)

Correlates of Mutational Robustness



Environmental Robustness

Are mutational and environmental robustness correlated in software?

Definitions

mutational robustness robustness to *genotypic* variation

environmental robustness robustness to *environmental* variation

Requirements

- metric of mutational robustness
- metric of environmental robustness (fuzz testing)

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Software Diversity

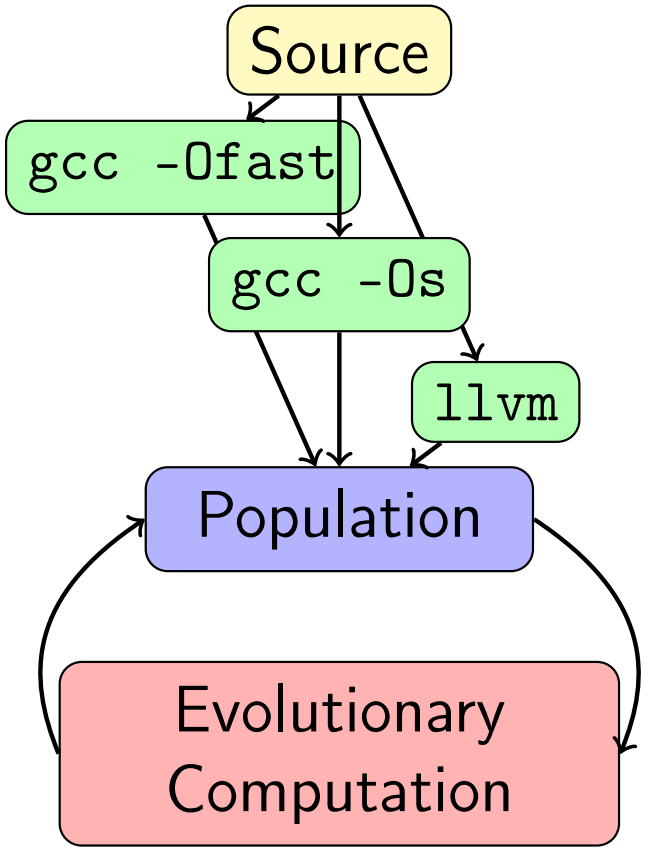
Compilers and Linkers

Software Diversity:

- increases security and reliability [Williams et al., 2009]
- proactively fixes bugs (Zak Fry in Schulte et al. [2012])

Multiple Compilation:

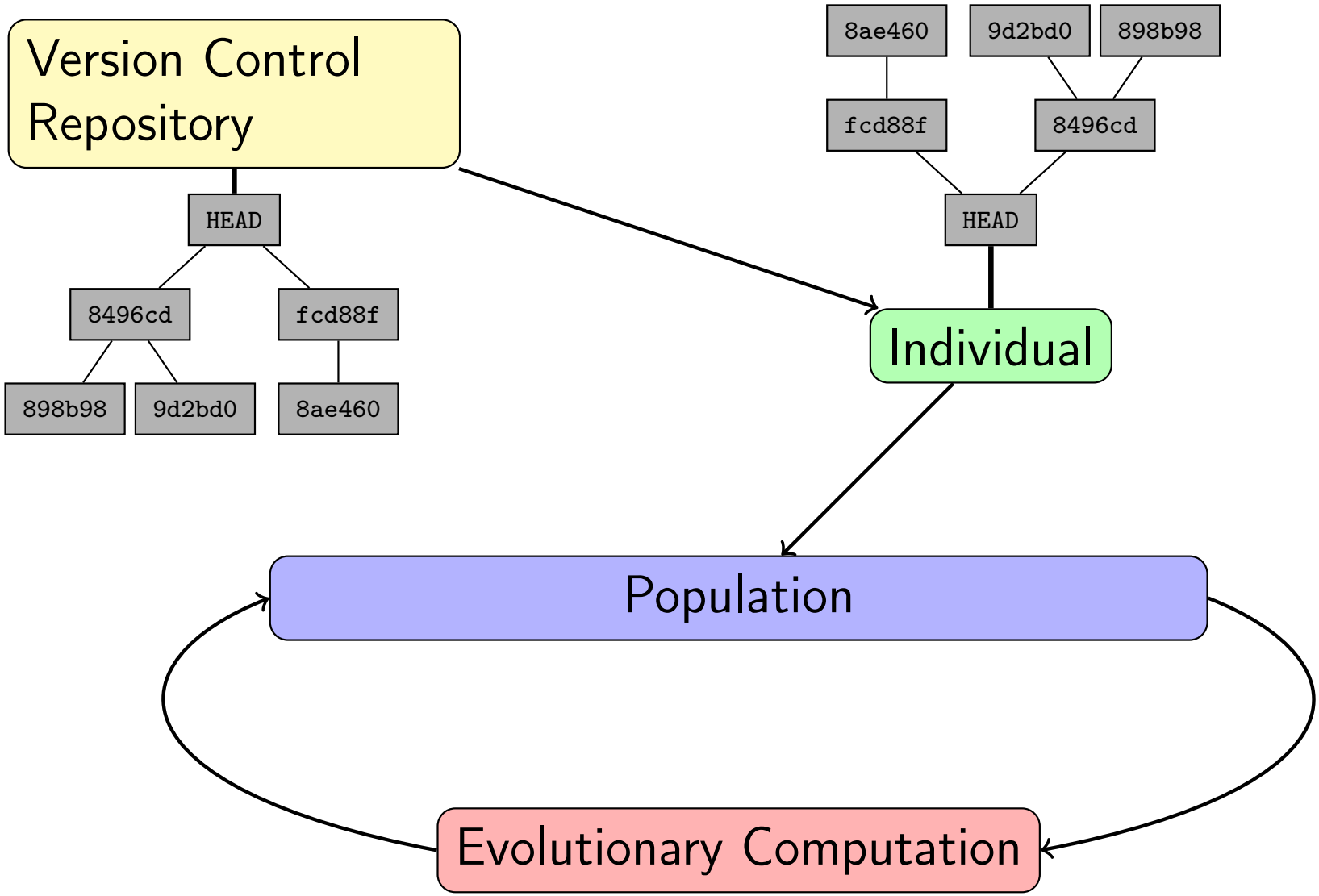
- *Jump-start* population diversity
- Increase library of genetic material
- Mix and match optimization between compilers



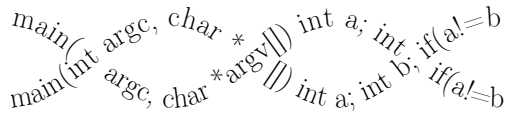
main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

New Program Representation

Program Atavism using Version Control



Software Optimization



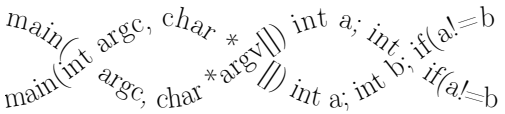
Components

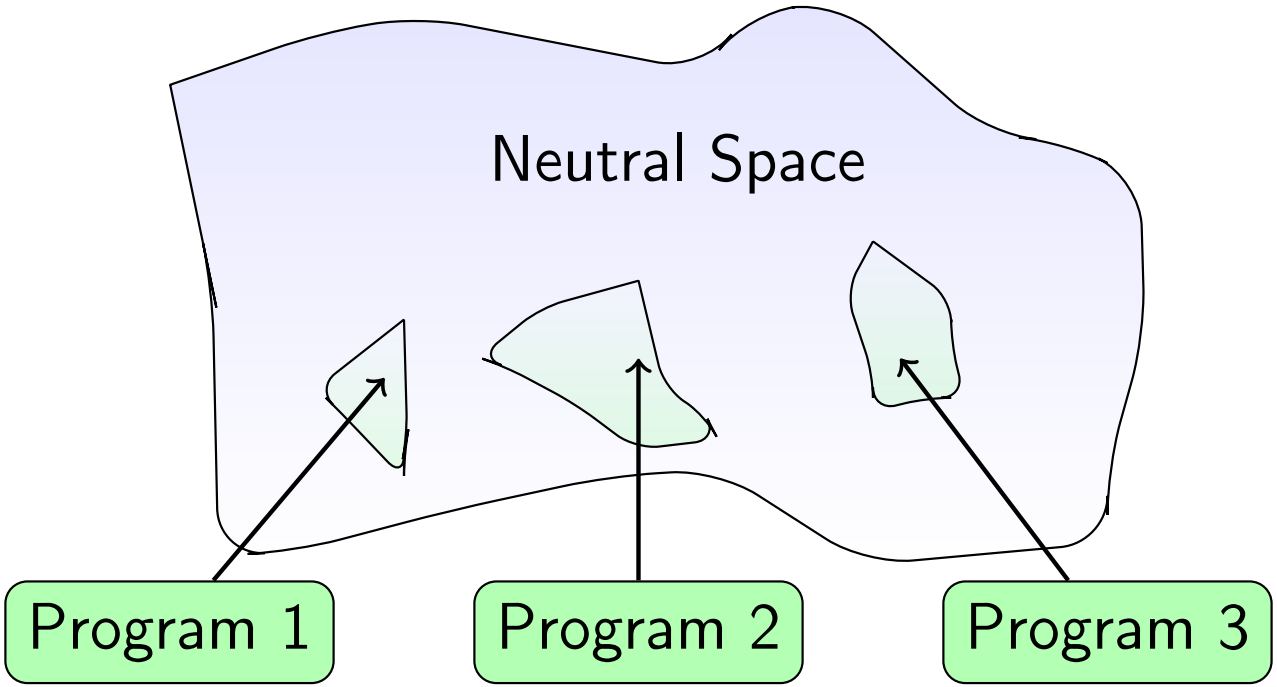
- extant program with existing test suite
- system emulator
- multi-objective fitness function combining
  - existing test suite
  - performance metrics

Benefits

- flexibility through unsound transformations
- explicit priorities through multi-objective fitness functions
- optimization of hard-a-priori features
  - hardware specific and multi-threaded issues
  - performance trade-offs (e.g., memory/cpu), etc. . .

Software Husbandry

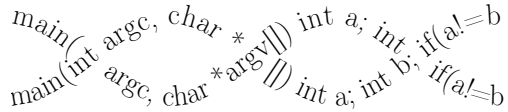




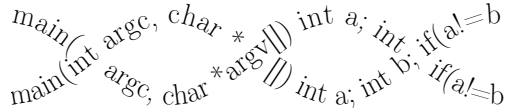
Benefits

- transfer optimization between projects
- transfer functionality between projects
- source of implementation diversity

Summary of Preliminary Work



Summary of Preliminary Work



- Genprog

Summary of Preliminary Work

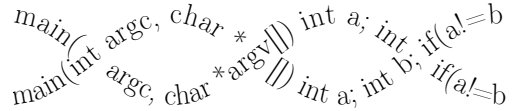
main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

- Genprog
  - ASM Rep.

`bubble.s`

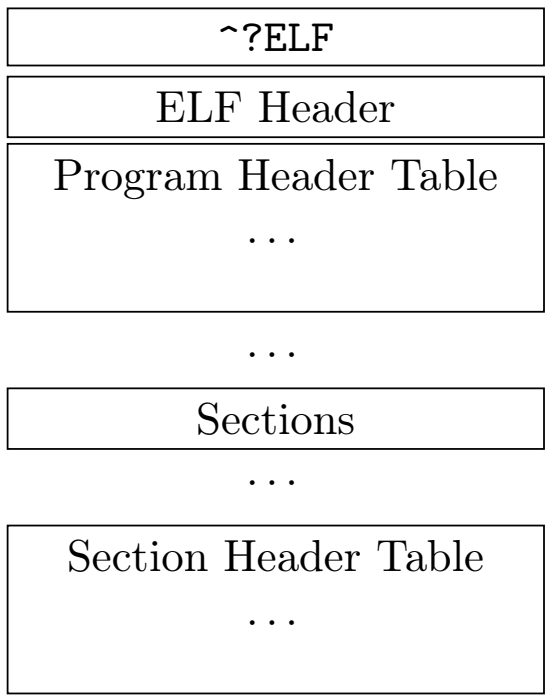
```
        .section .bss
        .lcomm buf, 500
        .lcomm lst, 500
        .section .text
        .globl _start
_start:
        popl %eax
        popl %eax
        movl $0, %esi
        movl $10, %edx
        jmp  read_arg
store_arg:
        movl %ecx, lst(,%esi,4)
        incl %esi
read_arg:
        popl %eax
        cmpl $0, %eax
        je   sort
        movl $0, %edi
        movl $0, %ecx
read_ch:
        movb 0(%eax,%edi), %bl
        cmpl $0, %ebx
        je   store_arg
        imul %edx, %ecx
        subl $48, %ebx
        addl %ebx, %ecx
        incl %edi
        jmp  read_ch
sort:
        movl $0, %ecx
        movl $0, %edi
        movl lst(,%edi,4), %eax
main_loop:
        movl %eax, %ebx
        movl %edi, %edx
        incl %edi
        cmpl %edi, %esi
        je   cond_restart
        movl lst(,%edi,4), %eax
        cmpl %ebx, %eax
        jge  main_loop
        incl %ecx
        movl %ebx, lst(,%edi,4)
        movl %eax, lst(,%edx,4)
        jmp  sort
cond_restart:
        cmpl $0, %ecx
        jne  sort
        movl $0, %edi
        movl $0, %edi
print_loop:
        pushl %edi
        movl lst(,%edi,4), %eax
        pushl $0
        movl $10, %ecx
cnv_int_down:
        movl $0, %edx
        idiv %ecx
        addl $48, %edx
        pushl %edx
        cmpl $0, %eax
        jne  cnv_int_down
        movl $0, %edi
cnv_int_up:
        popl %eax
        cmpl $0, %eax
        je   do_print
        movb %al, buf(,%edi,1)
        incl %edi
        jmp  cnv_int_up
do_print:
        movl $4, %eax
        movl $buf, %ecx
        movl $1, %ebx
        movl %edi, %edx
        int  $0x80
        movl $32, buf
        movl $buf, %ecx
        movl $4, %eax
        movl $1, %ebx
        movl $1, %edx
        int  $0x80
        popl %edi
        incl %edi
        cmpl %edi, %esi
        jg   print_loop
        movl $10, buf
        movl $buf, %ecx
        movl $4, %eax
        movl $1, %ebx
        int  $0x80
        movl $1, %eax
        movl $0, %ebx
        int  $0x80
```

Summary of Preliminary Work



- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`

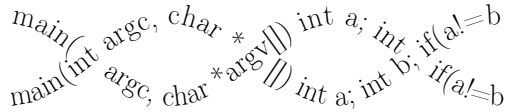
Structure of an ELF Binary



[a] http://gitweb.adaptive.cs.unm.edu/elf.git

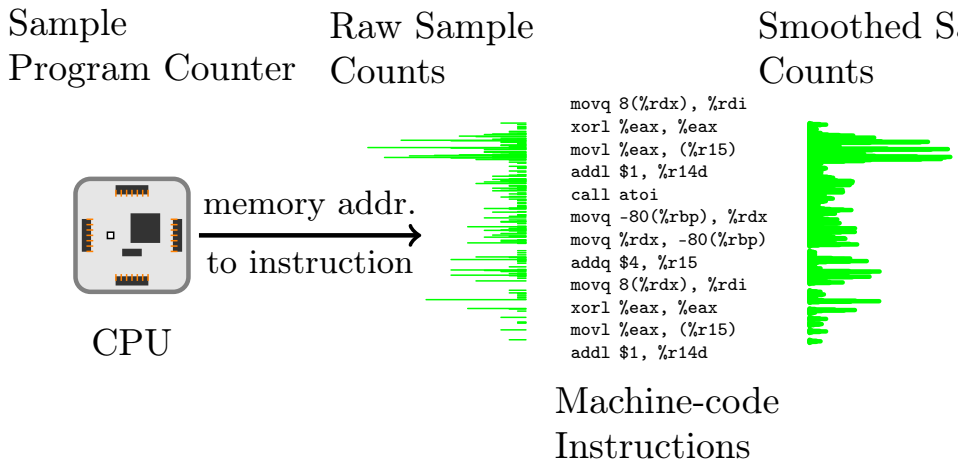
[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work



- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`
  - Fault Localization

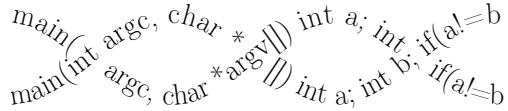
Fault Localization Overview



[a] http://gitweb.adaptive.cs.unm.edu/elf.git

[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work

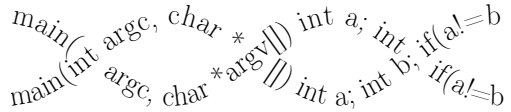


- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`
  - Fault Localization
- Mutational Robustness

[a] http://gitweb.adaptive.cs.unm.edu/elf.git

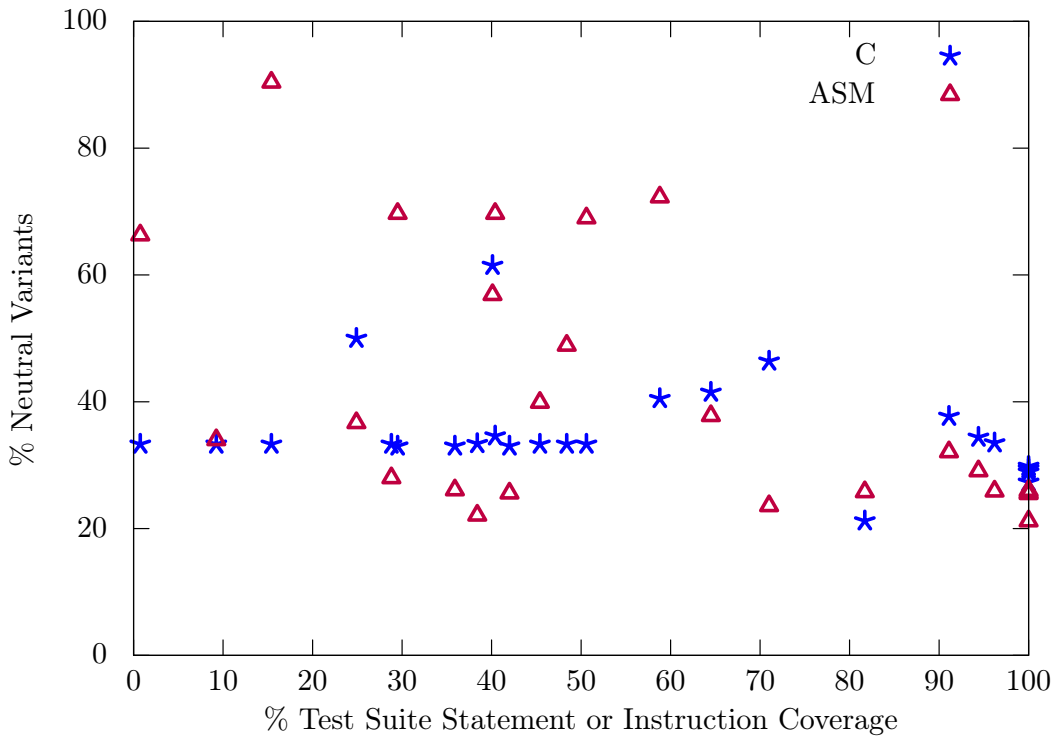
[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work



- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`
  - Fault Localization
- Mutational Robustness
  - Experimentation

Software Mutational Robustness



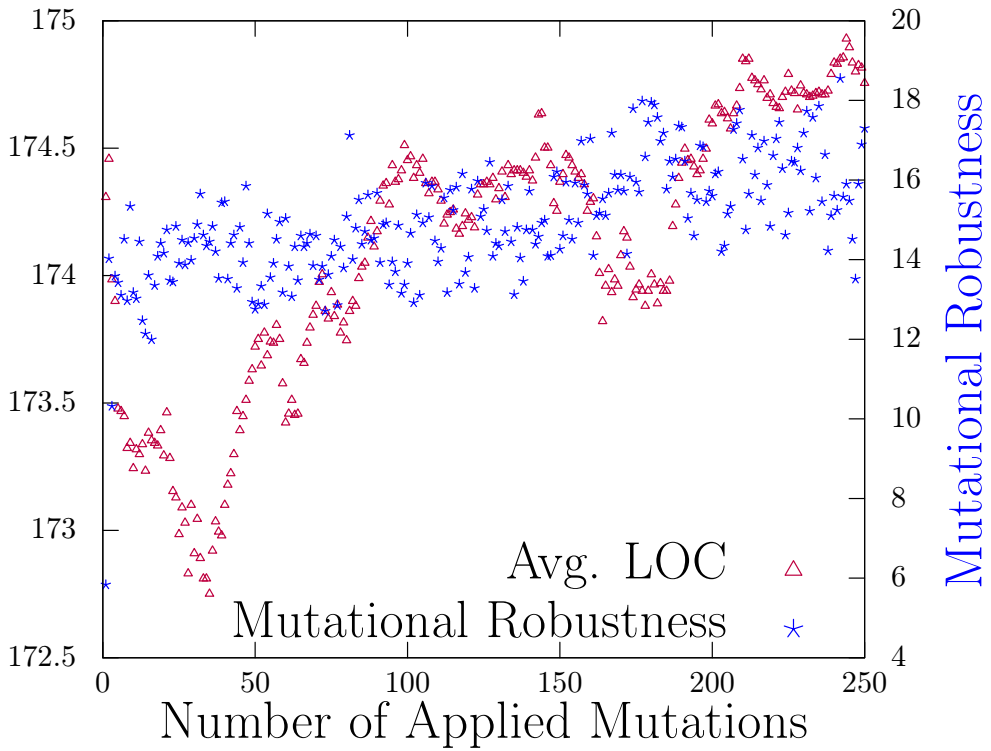
[a] http://gitweb.adaptive.cs.unm.edu/elf.git

[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work

- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b] – `C`
  - Fault Localization
- Mutational Robustness
  - Experimentation
  - Random walks

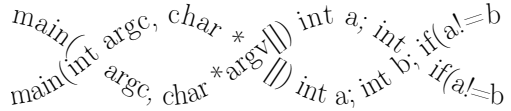
Neutral Variants of Insertion Sort



[a] http://gitweb.adaptive.cs.unm.edu/elf.git

[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work



- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`
  - Fault Localization
- Mutational Robustness
  - Experimentation
  - Random walks
  - Multiple languages

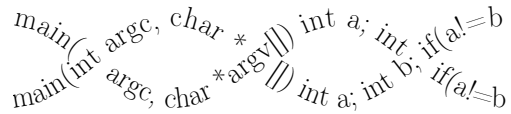
Robustness Across Multiple Languages

	C	C++	Haskell	OCaml	Avg.
bubble	25.7	28.2	27.6	16.7	24.6
insertion	26.0	42.0	35.6	23.7	31.8
merge	21.2	46.0	24.9	22.7	28.7
quick	25.5	42.0	26.3	11.4	26.3
Avg.	24.6	39.5	28.6	18.6	
Std.Dev.	2.3	7.8	4.8	5.7	

[a] http://gitweb.adaptive.cs.unm.edu/elf.git

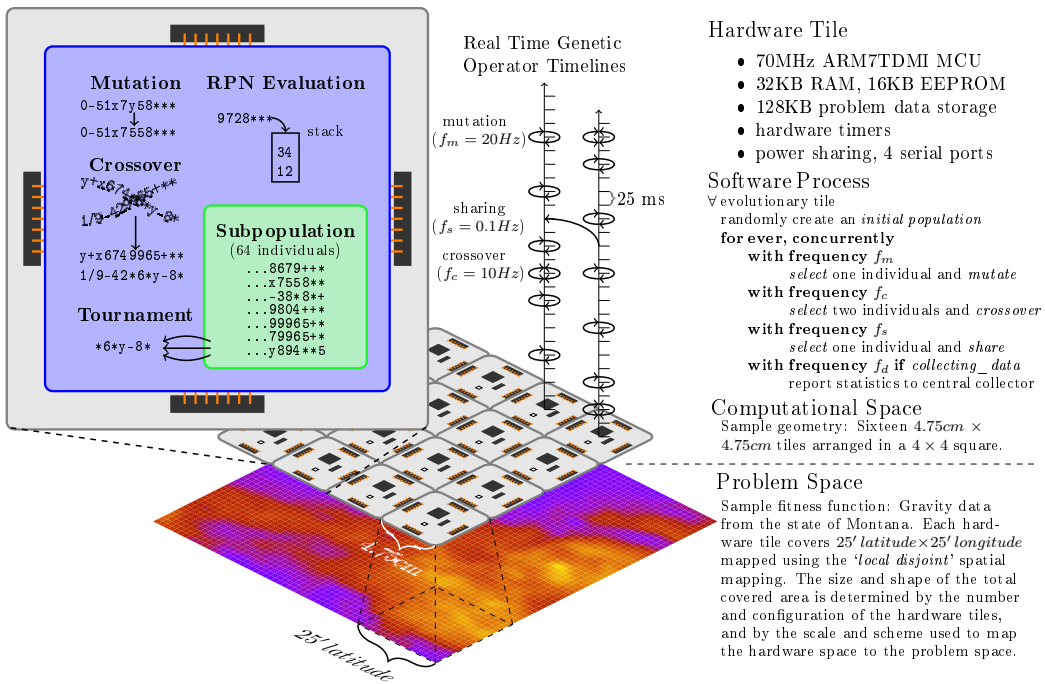
[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Summary of Preliminary Work



- Genprog
  - ASM Rep.
  - ELF Rep.
    - elf[a] – `Lisp`
    - rw-elf[b]– `C`
  - Fault Localization
- Mutational Robustness
  - Experimentation
  - Random walks
  - Multiple languages
- Robust Hardware

Physical Evolutionary Computation

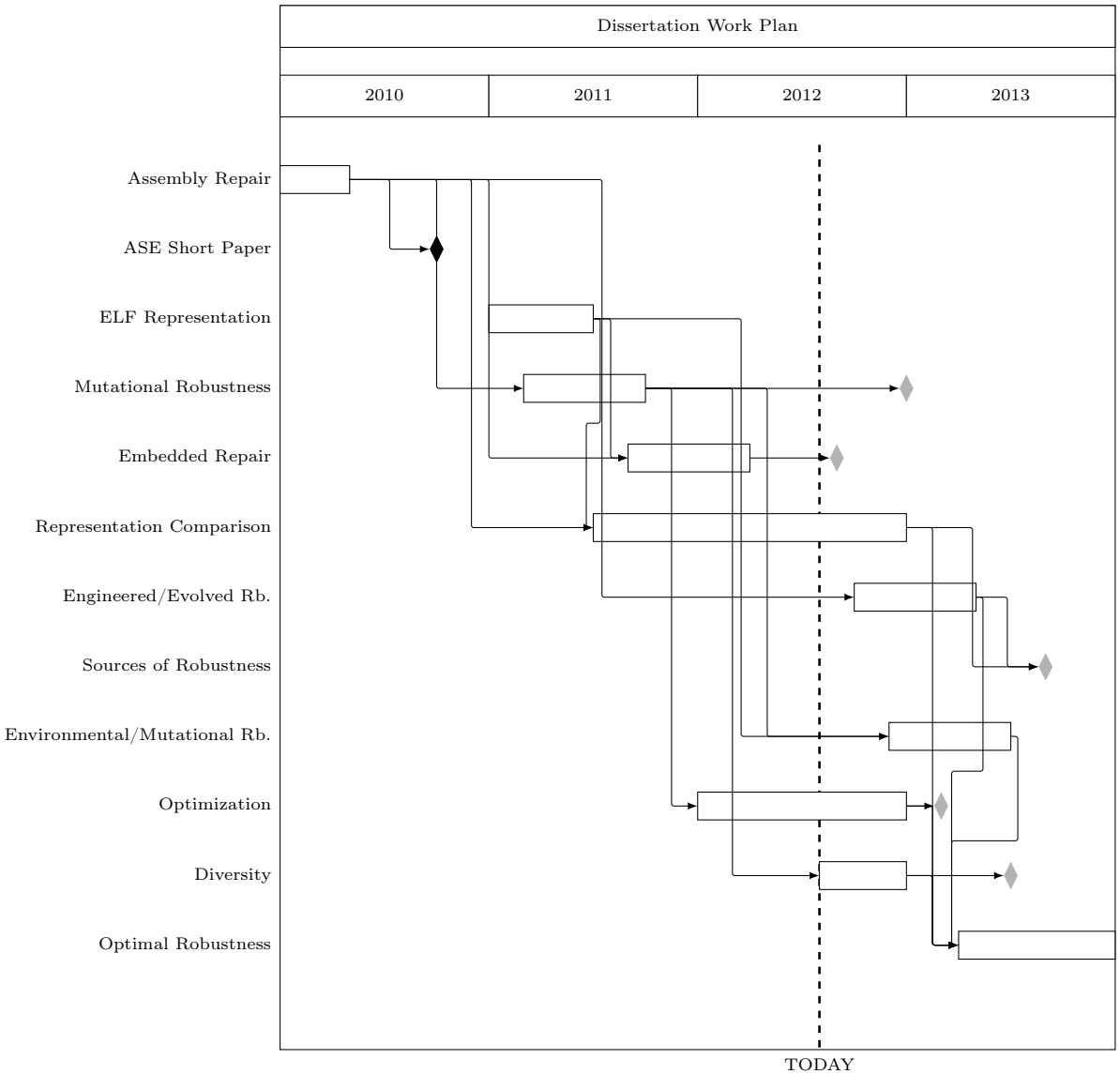


[a] http://gitweb.adaptive.cs.unm.edu/elf.git

[b] http://gitweb.adaptive.cs.unm.edu/rw-elf.git

Work-plan and Timeline

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b



Conclusion

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

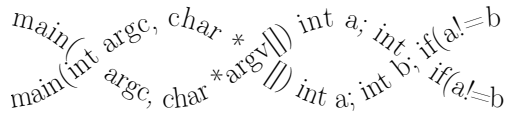
- Software is a product of natural selection
- Software is not fragile
- Software robustness can be used to build new tools
- Software may provide general insight into evolution and robustness.

Thank You

main(int argc, char *argv[]) int a; int b; if(a!=b
main(int argc, char *argv[]) int a; int b; if(a!=b

Eric Schulte

office	FEC309
email	eschulte@cs.unm.edu
homepage	http://cs.unm.edu/~eschulte



Bibliography

J. Beal and G. J. Sussman. Engineered robustness by controlled hallucination. November 2008.

U. D. o. L. Bureau of Labor Statistics. Computer software engineers and computer programmers. On the internet, November 2011. http://www.bls.gov/oco/ocos303.htm.

S. Ciliberti, O. Martin, and A. Wagner. Innovation and robustness in complex regulatory gene networks. *Proceedings of the National Academy of Sciences*, 104(34):13591, 2007.

G. M. Edelman and J. Gally. Degeneracy and complexity in biological systems. *Proceedings of the National Academy of Sciences*, 98(24):13763, 2001.

J. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The MIT press, 1992.

R. Knight, S. Freeland, and L. Landweber. Selection, history and chemistry: the three faces of the genetic code. *Trends in biochemical sciences*, 24(6):241–247, 1999.

J. Koza. Genetic programming: On the programming of computers by means of natural selection, 1992. *See http://miriad. lip6. fr/microbes Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology*, 229, 1992.

S. Misailovic, D. Roy, and M. Rinard. Probabilistically accurate program transformations. *Static Analysis*, pages 316–333, 2011.

C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229, 2004.

J. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, et al. Automatically patching errors in deployed software. 2009.

K. Raman and A. Wagner. The evolvability of programmable hardware. *Journal of The Royal Society Interface*, jun 2010.

M. Rinard, C. Cadar, D. Dumitran, D. Roy, T. Leu, and W. Beebee Jr. Enhancing server availability and security through failure-oblivious computing. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, pages 21–21. USENIX Association, 2004.

E. Schulte, Z. P. Fry, E. Fast, S. Forrest, and W. Weimer. Software mutational robustness: Bridging the gap between mutation testing and evolutionary biology. http://arxiv.org/abs/1204.4224, April 2012.

E. Van Nimwegen, J. Crutchfield, and M. Huynen. Neutral evolution of mutational robustness. *Proceedings of the National Academy of Sciences*, 96(17):9716, 1999.

A. Wagner. Robustness and evolvability in living systems. 2005.

W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering*, pages 364–374. IEEE Computer Society, 2009.