Automated Program Repair through the Evolution of Assembly Code

Eric Schulte

University of New Mexico

08 August 2010

Introduction

We present a method of *automated program repair* through the evolution of *assembly code* compiled from *extant software*.

- previous work demonstrated the repair of C programs through evolution of C statements
- we extend previous work by operating at the level of assembly commands

Benefits

Benefits of the assembly code level of representation include

General
: applicable to any language which compiles to Java byte code or x86 assembly code

Expressive
: the small scale of assembly instructions is capable of expressing repairs not possible at the C statement level

Coverage
: the small alphabet of assembly commands and large number of commands in assembly programs provides access to a larger subset of the space of possible programs

Robust
: the functionality of assembly programs is more robust to mutation

Comparative Performance

Repair at the Assembly level is roughly 17% slower than at the C level.

Technical Approach

Stages

preprocessing

fault localization

mutation

evolution

Technical Approach

Stages

preprocessing

fault localization

mutation

evolution

C Statement Tree

Buggy C Program

Technical Approach

Stages

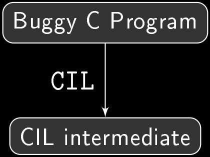
preprocessing

fault localization

mutation

evolution

C Statement Tree



Technical Approach

Stages

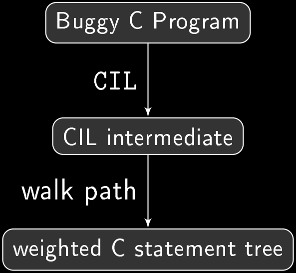
preprocessing

fault localization

mutation

evolution

C Statement Tree



Technical Approach

Stages

- preprocessing
- fault localization
- mutation
- evolution

C Statement Tree

Buggy C Program

↓ CIL

CIL intermediate

↓ walk path

weighted C statement tree

↓ tree mut.

population

Technical Approach

Stages

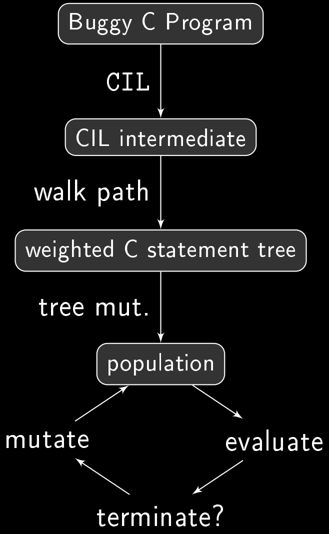
preprocessing

fault localization

mutation

evolution

C Statement Tree



Technical Approach

Stages

preprocessing

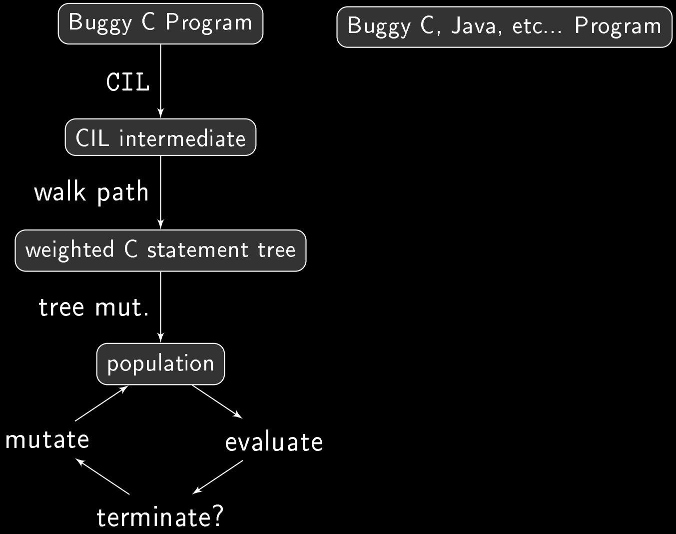
fault localization

mutation

evolution

C Statement Tree

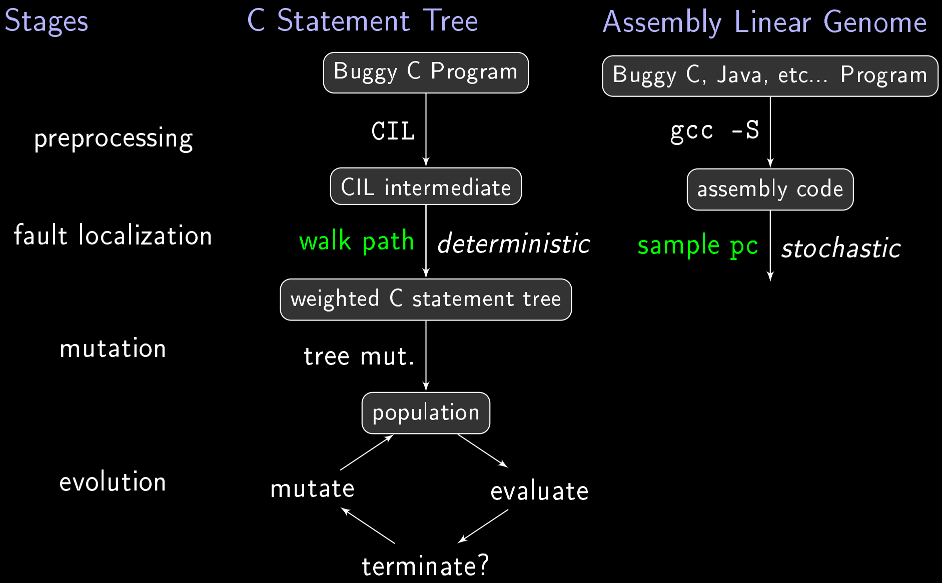
Assembly Linear Genome



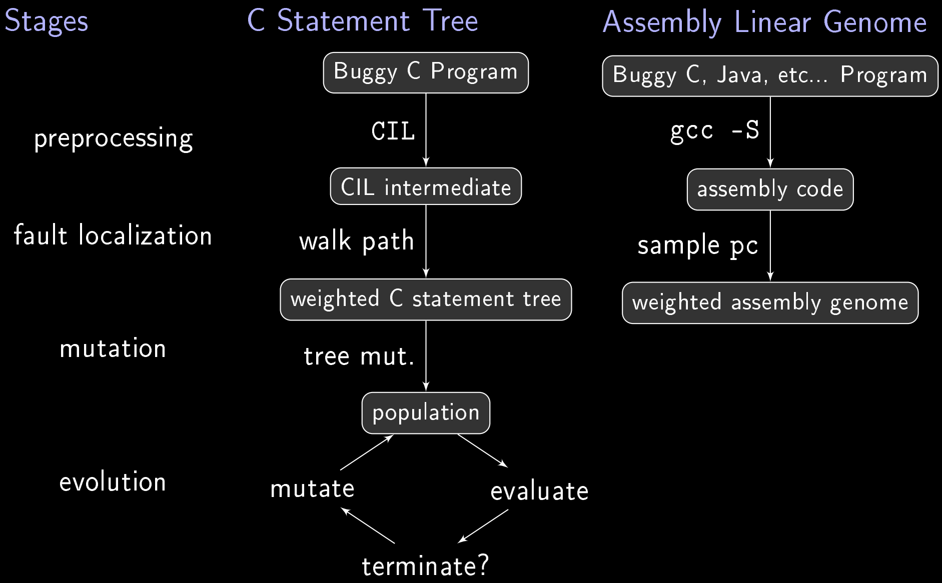
Technical Approach

Stages	C Statement Tree	Assembly Linear Genome
	Buggy C Program	Buggy C, Java, etc... Program
preprocessing	↓ `CIL`	↓ `gcc -S`
	CIL intermediate	assembly code
fault localization	↓ walk path	
	weighted C statement tree	
mutation	↓ tree mut.	
	population	
evolution	population → evaluate → terminate? → mutate → population	

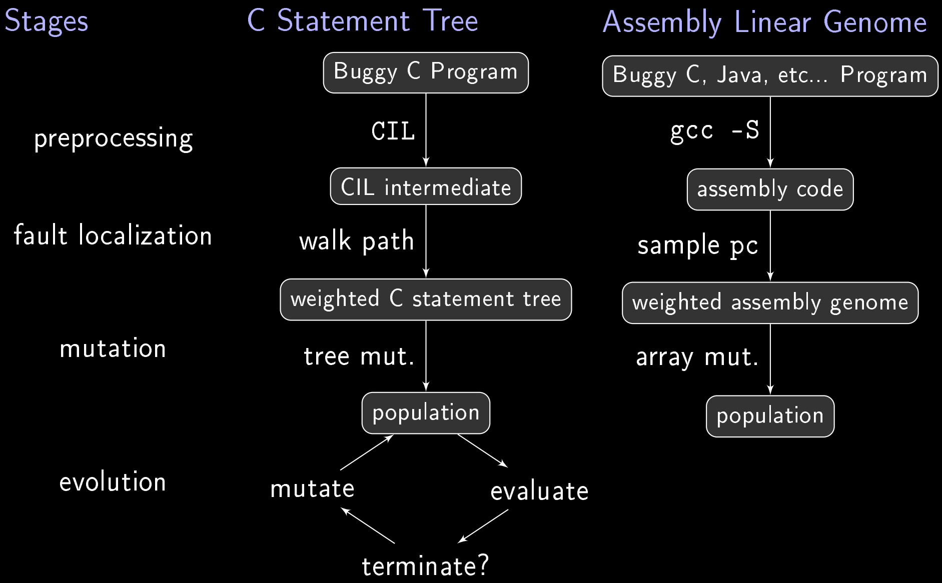
Technical Approach



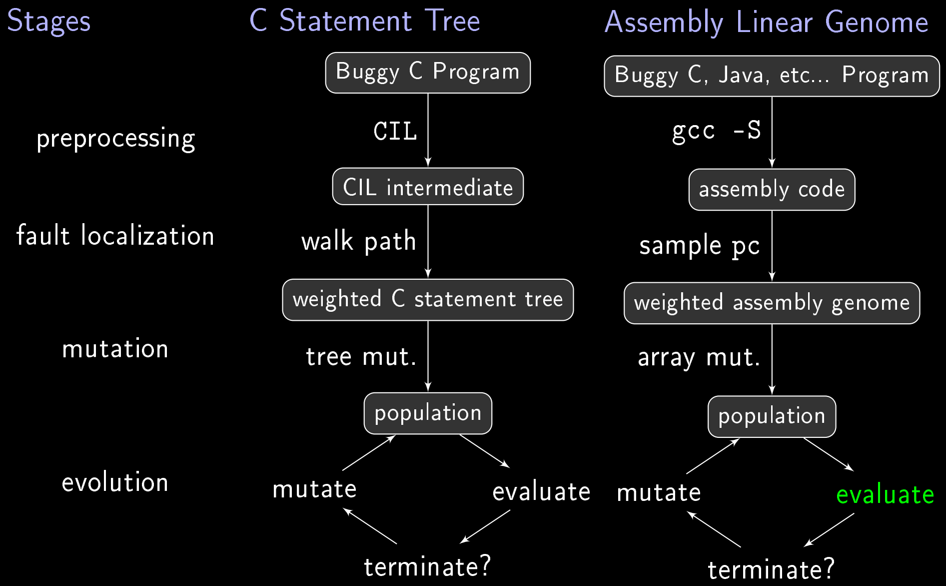
Technical Approach



Technical Approach



Technical Approach



Representation and Genetic Operators

Linear Genome

Individuals are *linear genomes* of weighted assembly commands.

```
("main:" "pushl %ebp" "movl %esp" ... )
```

Genetic Operators

insert selects an instruction, selects a location, copies the instruction and inserts in the location

delete selects an instruction and deletes it

swap selects two instructions and swaps them

crossover selects a crossover point for each of two individuals and exchanges all instructions after that point between the individuals

Fault Localization

Technique

CPU: program counter

oprofile ↓ raw addresses

```
Dump of assembler code for function main:
0x08048414  main+0: push  %ebp
0x08048415  main+1: mov   %esp,%ebpmain:
0x08048417  main+3: and   $0xfffffff0,%esp
```

gdb ↓ offset in method

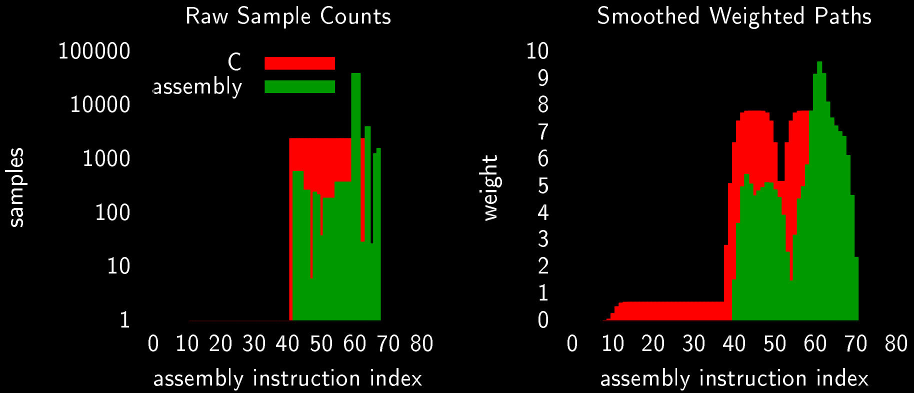
```
main: pushl %ebp
movl %esp, %ebp
andl $-16, %esp
```

mem-mapping.clj ↓ index in genome

```
("main:" "pushl %ebp" "movl %esp" ... )
```

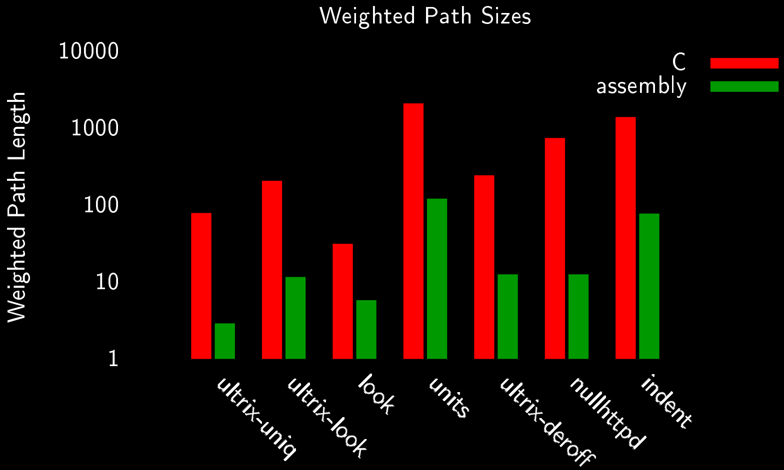
Fault Localization

Results



Fault Localization

Comparative Path Size



Generality to Multiple Languages

Input: Integer a
Input: Integer b
Output: $gcd(a, b)$ or $\perp$

if $a \equiv 0$ **then**
  print a
end if
while $b \neq 0$ **do**
  if $a > b$ **then**
    $a \leftarrow a - b$
  else
    $b \leftarrow b - a$
  end if
end while
print a

Figure: A Buggy version of Euclid's Algorithm

	C	Haskell	Java
Program Length	79	885	33
Total Solutions	2	15	13
Unique Solutions	2	10	1

Table: GCD Repair Results by Language

Mutational Robustness & Neutral Spaces

Concepts

Mutational Robustness robustness of *phenotype* to changes in *genotype* in the case of computer programs

- **phenotype** behavior of the program
- **genotype** representation of the program

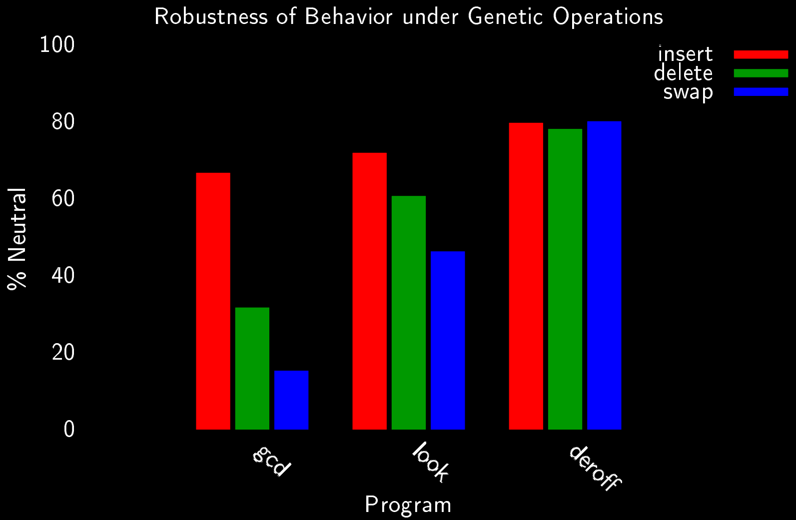
Neutral Space contiguous region of *genotype space* with constant *phenotype*

Mutational Robustness & Neutral Spaces

Relevance

- Mutational Robustness directly influences evolvability
- A large neutral space allows for significant diversity in
  - representation
  - behavior
  - efficiency
  - size
  - etc. . .
- A *neutral neighborhood* defines which functionalities are reachable in a single genotypic step

Robustness under Genetic Operators



Future Work

Investigate Mutational Robustness

- differences across *languages*, *algorithms* and *representations*
- properties of the *neutral spaces* of assembly programs

Improve GP Technique

- apply ongoing work on the C statement level
- homologous crossover, steady state mutation, etc. . .

Non-Repair Evolution

- machine specific optimization
- disruption of mono-culture (N-variant)

Discussion

- Questions
- Comments
- Suggestions