Applying Genetic Programming to Bytecode and Assembly

Eric Collom

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

29 April '14,
UMM Senior Seminar

The big picture

-
-
-
-
-

Outline

1. Background
2. Why Evolve Instruction-level Code
3. FINCH:Evolving Programs
4. Using Instruction-level code to automate bug repair
5. Conclusions

Outline

1. Background
   - EC
   - Java Bytecode and x86 Assembly
2. Why Evolve Instruction-level Code
3. FINCH:Evolving Programs
4. Using Instruction-level code to automate bug repair
5. Conclusions

Evolutionary Computation

Java Bytcode and x86 Assembly

Outline

1 Background

2 Why Evolve Instruction-level Code

3 FINCH:Evolving Programs

4 Using Instruction-level code to automate bug repair

5 Conclusions

Source Code Constraints

Flexibility

Outline

1 Background

2 Why Evolve Instruction-level Code

3 FINCH:Evolving Programs
- How it Works
- Results

4 Using Instruction-level code to automate bug repair

5 Conclusions

Selecting Offspring

- There is still a chance to produce non-compilable code
- Solution: Add restrictions to code selection.
- Stack and Frame Depth
- Variable Types
- Control Flow

Crossover

Non-Halting Offspring

Outline

1 Background

2 Why Evolve Instruction-level Code

3 FINCH:Evolving Programs

4 Using Instruction-level code to automate bug repair
- How it Works
- Results

5 Conclusions

Selecting Offspring

Genetic Operators

Non-Halting Offspring

Outline

1 Background

2 Why Evolve Instruction-level Code

3 FINCH:Evolving Programs

4 Using Instruction-level code to automate bug repair

5 Conclusions

Conclusions

References