UFMG

# Multi-Language Benchmark Generation via L-Systems

Compilers Laboratory

Vinicius Francisco da Silva
silva.vinicius@dcc.ufmg.br
Fernando Magno Quintão Pereira
fernando@dcc.ufmg.br

# Agenda

# Agenda

DCC

- Context

# Agenda

DCC

- Context
  - L-System

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs
- System Overview

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs
- System Overview
  - Syntactical Building Blocks

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs
- System Overview
  - Syntactical Building Blocks
  - Execution Flow

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs
- System Overview
  - Syntactical Building Blocks
  - Execution Flow
  - Memory Management

# Agenda

DCC

- Context
  - L-System
  - Self Similar Structures
  - Self Similar Programs
- System Overview
  - Syntactical Building Blocks
  - Execution Flow
  - Memory Management
  - Multi Language Support

# Agenda

DCC

- Evaluation
  - gcc vs clang

# Agenda

DCC

- Evaluation
  - gcc vs clang
  - Comparison between Different Programming Languages

# Agenda

- Evaluation
  - gcc vs clang
  - Comparison between Different Programming Languages
  - The Evolution of gcc

# Multi-Language Benchmark Generation via L-Systems

# Multi-Language Benchmark Generation via L-Systems

# What is L-System?

DCC

# L-System

DCC

- Mathematical and recursive formalism.

# L-System

DCC

- Mathematical and recursive formalism.
- Created to describe similar structures.

# L-System

DCC

- Mathematical and recursive formalism.
- Created to describe similar structures.
- Defined by a grammar called L-Grammar

# L-System

DCC

- Mathematical and recursive formalism.
- Created to describe similar structures.
- Defined by a grammar called L-Grammar

$$G = (\Sigma, \omega, P)$$

# L-System

- Mathematical and recursive formalism.
- Created to describe similar structures.
- Defined by a grammar called L-Grammar

$$G = (\Sigma$$

Finite set of symbols

L-Grammar

# L-System

- Mathematical and recursive formalism.
- Created to describe similar structures.
- Defined by a grammar called L-Grammar

$$G = (\quad \omega$$

Initial string formed by symbols from Σ

L-Grammar

# L-System

- Mathematical and recursive formalism.
- Created to describe similar structures.
- Defined by a grammar called L-Grammar

$$G = (\qquad P)$$

Define how symbols are rewritten at each iteration.

L-Grammar

# L-System

DCC



Aristid Lindenmayer
1925 - 1989

# L-System

DCC



Aristid Lindenmayer
1925 - 1989

# Self similar structures

[Axiom]

A

# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B
B = A + B + A

# Self similar structures

DCC

[Axiom]

A

[Productions]

A = B - A - B
B = A + B + A

[Semantics]

[-] 60°
[+] 300°
[A] Line
[B] Line

# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B
B = A + B + A

[Semantics]

[-] 60º
[+] 300º
[A] Line
[B] Line

Strings:

A

# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B
B = A + B + A

[Semantics]

[-] 60º
[+] 300º
[A] Line
[B] Line

Strings:

A

B−A−B



# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B

B = A + B + A

[Semantics]

[-] 60º

[+] 300º

[A] Line

[B] Line

Strings:

A

B−A−B

A+B+A−B−A−B−A+B+A





# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B

B = A + B + A

[Semantics]

[-] 60°

[+] 300°

[A] Line

[B] Line

Strings:

A

B−A−B

A+B+A−B−A−B−A+B+A

B−A−B+A+B+A+B−A−B−
A+B+A−B−A−B−A+B+A−
B−A−B+A+B+A+B−A−B



# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B

B = A + B + A

[Semantics]

[-] 60º

[+] 300º

[A] Line

[B] Line

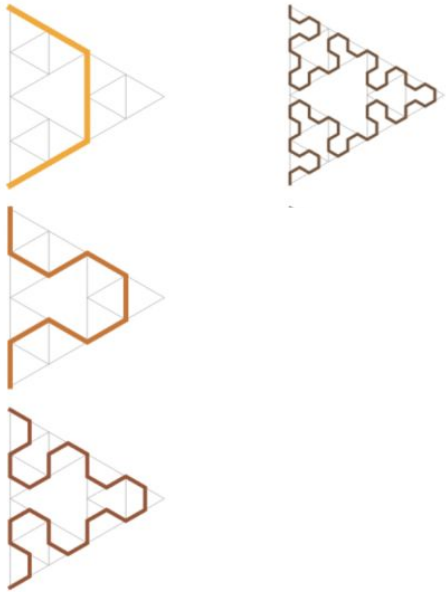
Strings:

A

B−A−B

A+B+A−B−A−B−A+B+A

B−A−B+A+B+A+B−A−B−
A+B+A−B−A−B−A+B+A−
B−A−B+A+B+A+B−A−B



# Self similar structures

[Axiom]

A

[Productions]

A = B - A - B

B = A + B + A

[Semantics]

[-] 60º

[+] 300º

[A] Line

[B] Line

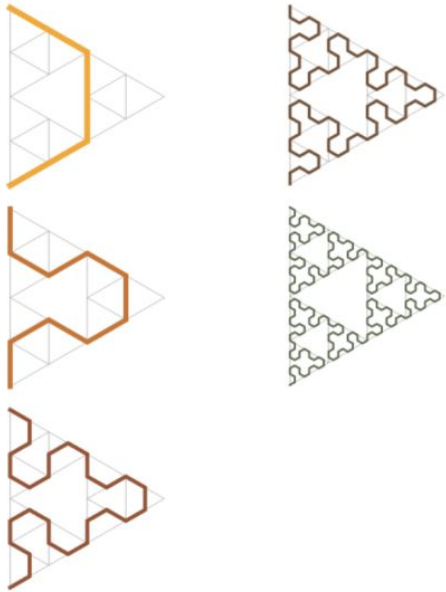
Strings:

A

B−A−B

A+B+A−B−A−B−A+B+A

B−A−B+A+B+A+B−A−B−
A+B+A−B−A−B−A+B+A−
B−A−B+A+B+A+B−A−B



# Self similar structures

## Sierpinski Triangle

[Axiom]

A

[Productions]

A = B - A - B

B = A + B + A

[Semantics]

[-] 60º

[+] 300º

[A] Line

[B] Line

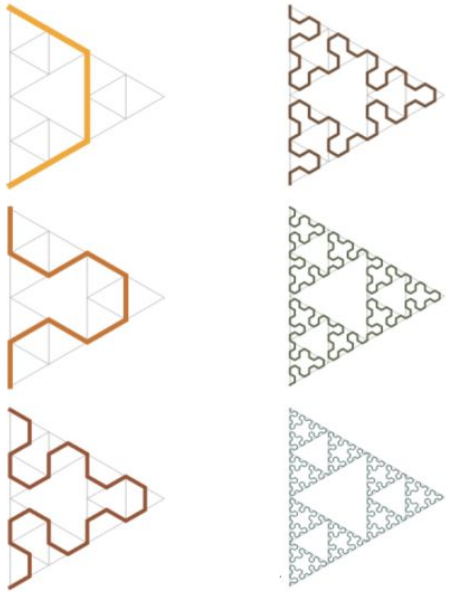
Strings:

A

B−A−B

A+B+A−B−A−B−A+B+A

B−A−B+A+B+A+B−A−B−
A+B+A−B−A−B−A+B+A−
B−A−B+A+B+A+B−A−B



# Multi-Language Benchmark Generation via L-Systems

# Self similar programs

DCC

# Self similar programs

DCC

How can we generate programs using L-System?

# Self similar programs

How can we generate programs using L-System?

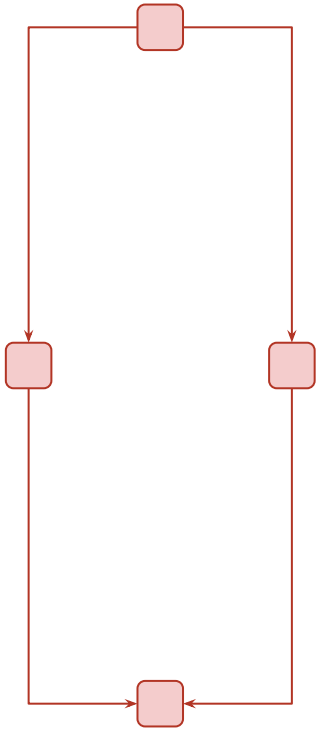


Programs have self-similar structures.

# Self similar programs

How can we generate programs using L-System?

Programs have self-similar structures.

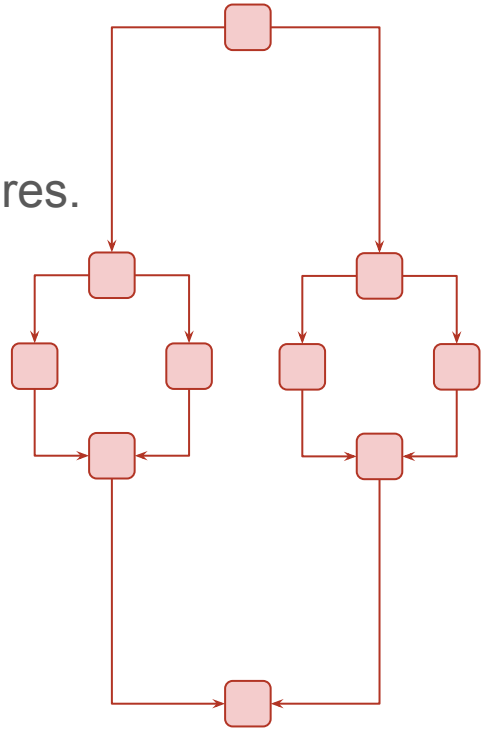


```
g(x) = if .... then .... else ....
```

# Self similar programs

How can we generate programs using L-System?

Programs have self-similar structures.



```
g(x) = if .... then g(x) else g(x)
```

# System Overview

DCC

How can we implement this generator?

# System Overview

DCC

BenchGen

# System Overview

BenchGen

L-grammar

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

$$\Sigma = \phi$$

Control Flow commands

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

$$\Sigma = \phi$$

Control Flow commands

$$\phi = \{\text{“}IF\text{”}, \text{“}LOOP\text{”}, \text{“}CALL\text{”}\}$$

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

$$\Sigma = \phi \bigcup \alpha$$

Data structure manipulation

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

$$\Sigma = \phi \bigcup \alpha$$

Data structure manipulation

$$\alpha = \{\text{“}new\text{”}, \text{“}insert\text{”}, \text{“}remove\text{”}, \text{”}contains\text{”}\}$$

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

$$\Sigma = \phi \bigcup \alpha \bigcup \{\text{“ = ”}, \text{“,”}, \text{“(”}, \text{“)”}\}$$

Additional tokens

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

Initial string formed by symbols from Σ

BenchGen

L-grammar

# System Overview

DCC

$$G = (\Sigma, \omega, P)$$

BenchGen

L-grammar

Define how symbols are replaced at each iteration

# Example

BenchGen



L-grammar

[Axiom]
A

# Example

DCC

BenchGen

L-grammar

[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A

# Example

BenchGen

L-grammar

[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A

*Iteration 1*:
insert IF(B, LOOP(A)) remove

# Example

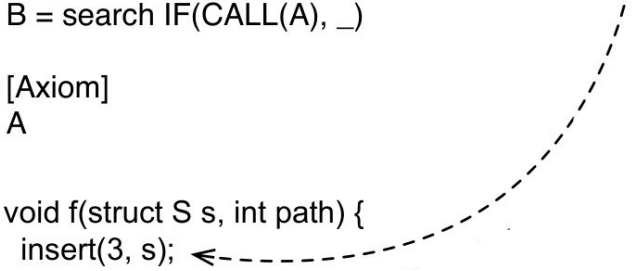
BenchGen

L-grammar

[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A

*Iteration 1*:
insert IF(B, LOOP(A)) remove



```
void f(struct S s, int path) {
  insert(3, s);
}
```

# Example

BenchGen

L-grammar

```
[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A

void f(struct S s, int path) {
  insert(3, s);
  if (path & 1) {
        B(s);
  } else {


  }

}
```

*Iteration 1*:
insert IF(B, LOOP(A)) remove

# Example

BenchGen

L-grammar

```
[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A

void f(struct S s, int path) {
  insert(3, s);
  if (path & 1) {
        B(s);
  } else {
        for (int i0 = 0; i0 < 5; i0++) {
              A(s);
        }
  }

}
```

*Iteration 1*:
insert IF(B, LOOP(A)) remove

# Example

BenchGen

L-grammar

```
[Productions]
A = insert IF(B, LOOP(A)) remove
B = search IF(CALL(A), _)

[Axiom]
A
```

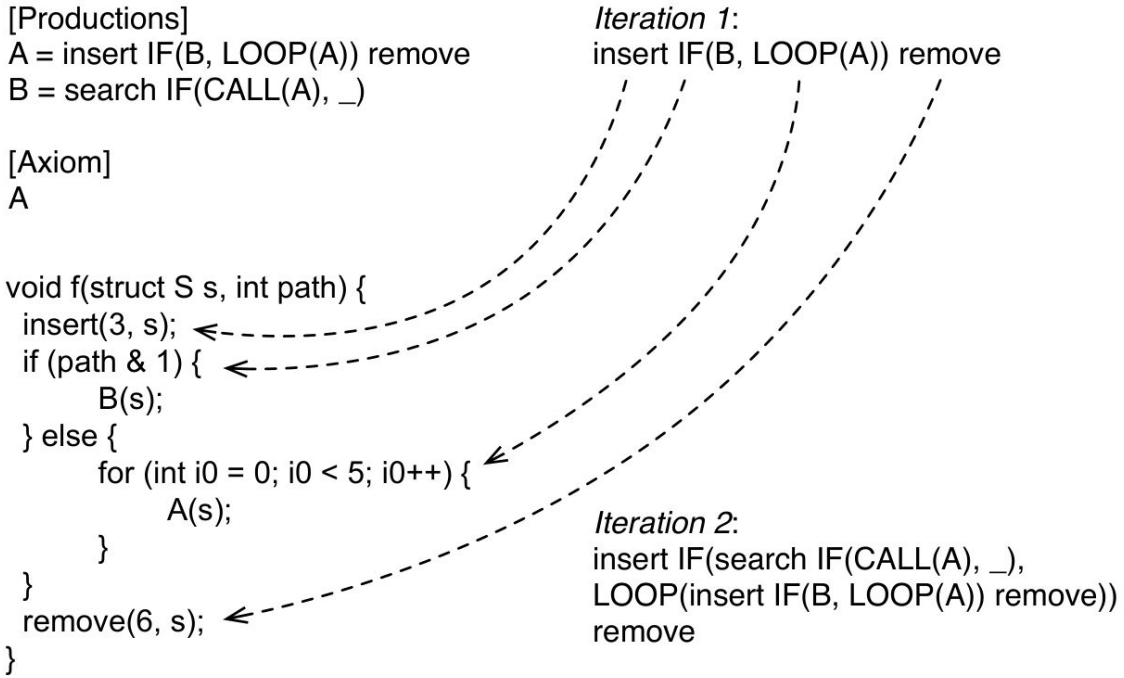
*Iteration 1*:
insert IF(B, LOOP(A)) remove

```
void f(struct S s, int path) {
  insert(3, s);
  if (path & 1) {
        B(s);
  } else {
        for (int i0 = 0; i0 < 5; i0++) {
              A(s);
        }
  }
  remove(6, s);
}
```

# Example

BenchGen

L-grammar



# Execution flow

DCC

# Execution flow

DCC

- BenchGen provides executable programs

# Execution flow

DCC

- BenchGen provides executable programs
  - The control flow is governed by dynamic variable PATH

# Execution flow

- BenchGen provides executable programs
  - The control flow is governed by dynamic variable *PATH*

# Execution flow

DCC

- BenchGen provides executable programs
  - The control flow is governed by dynamic variable *PATH*
    - PATH must be **dynamic**: a static value would let the compiler eliminate branches via constant folding.

# Execution flow

DCC

- BenchGen provides executable programs
  - The control flow is governed by dynamic variable *PATH*
    - PATH must be **dynamic**: a static value would let the compiler eliminate branches via constant folding.

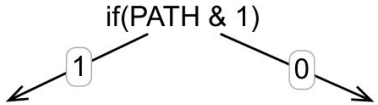
# Execution flow

Code pattern:

```
if (PATH & 1) {

} else {

}
```

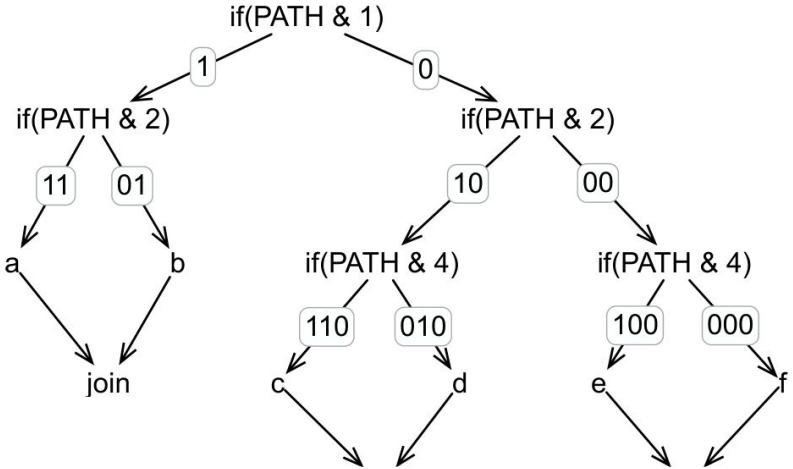



DCC

# Execution flow

Code pattern:

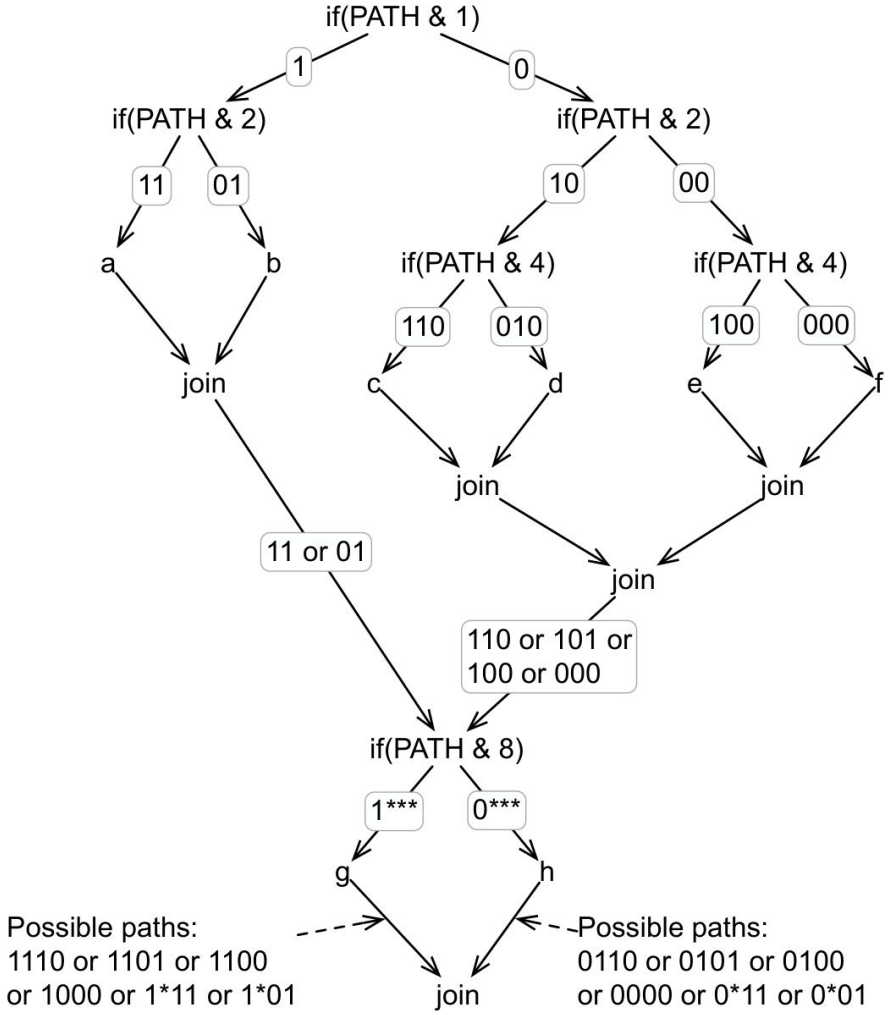
```
if (PATH & 1) {
    if (PATH & 2) {
        a;
    } else {
        b;
    }
} else {
    if (PATH & 2) {
        if (PATH & 4) {
            c;
        } else {
            d;
        }
    } else {
        if (PATH & 4) {
            e;
        } else {
            f;
        }
    }
}
```




DCC

# Execution flow

```
Code pattern:
if (PATH & 1) {
    if (PATH & 2) {
        a;
    } else {
        b;
    }
} else {
    if (PATH & 2) {
        if (PATH & 4) {
            c;
        } else {
            d;
        }
    } else {
        if (PATH & 4) {
            e;
        } else {
            f;
        }
    }
}
if (PATH & 8) {
    g;
} else {
    h;
}
```



# Memory Management

DCC

# Memory Management

DCC

- Benchmarks generated by BenchGen are free of memory leaks

# Memory Management

DCC

- Benchmarks generated by BenchGen are **free** of memory leaks

# Memory Management

DCC

- Benchmarks generated by BenchGen are **free** of memory leaks
  - BenchGen implements a reference counting garbage collector

# Memory Management

DCC

- Benchmarks generated by BenchGen are **free** of memory leaks
  - BenchGen implements a reference counting garbage collector
    - Each data structure has *refCounter* = *1*

# Memory Management

DCC

- Benchmarks generated by BenchGen are **free** of memory leaks
  - BenchGen implements a reference counting garbage collector
    - Each data structure has *refCounter* = *1*
    - *refCounter* += *1* when the reference of data structure is passed

# Memory Management

DCC

- Benchmarks generated by BenchGen are **free** of memory leaks
  - BenchGen implements a reference counting garbage collector
    - Each data structure has *refCounter* = *1*
    - *refCounter* += *1* when the reference of data structure is passed
    - *refCounter* -= *1* when a reference to the object no longer exists

# Memory Management - Example

DCC

```
typedef struct {
  unsigned int* data;
  size_t size;
  size_t refC;
  int id;
} array_t;
```

Reference counter

```
array_t* arr1 = arr0;
arr0->refC++;
```

New reference detected

```
}
```

# Memory Management - Example

DCC

```
typedef struct {
  unsigned int* data;
  size_t size;
  size_t refC;
  int id;
} array_t;
```

Reference counter

```
array_t* arr1 = arr0;
arr0->refC++;
```

New reference detected

```
{ …
  arr0->refC—;
```

Reference no longer exists

```
}
```

# Memory Management - Example

DCC

```
typedef struct {
  unsigned int* data;
  size_t size;
  size_t refC;
  int id;
} array_t;
```

Reference counter

```
array_t* arr1 = arr0;
arr0->refC++;
```

New reference detected

```
{ …
  arr0->refC—;
  if (!arr0->refC) {
    free(arr0->data);
    free(arr0);
  }
}
```

Reference no longer exists

There are no more references

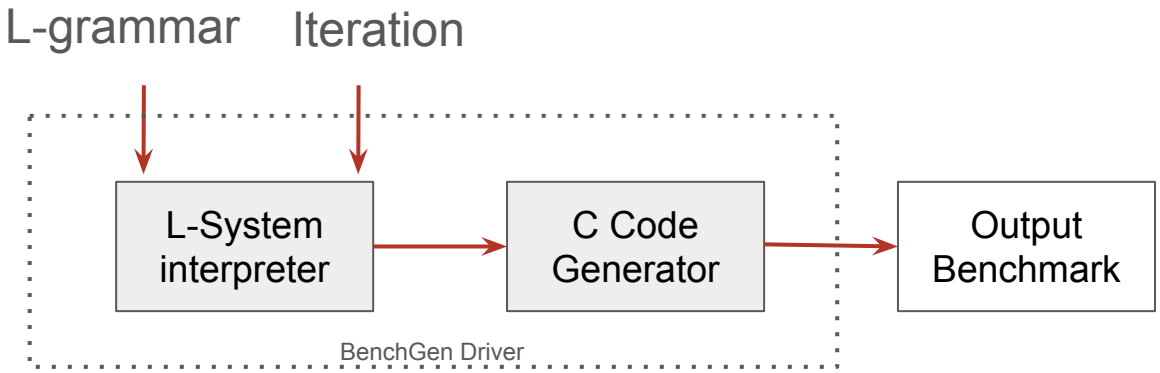
# Multi-Language Benchmark

## Generation via L-Systems

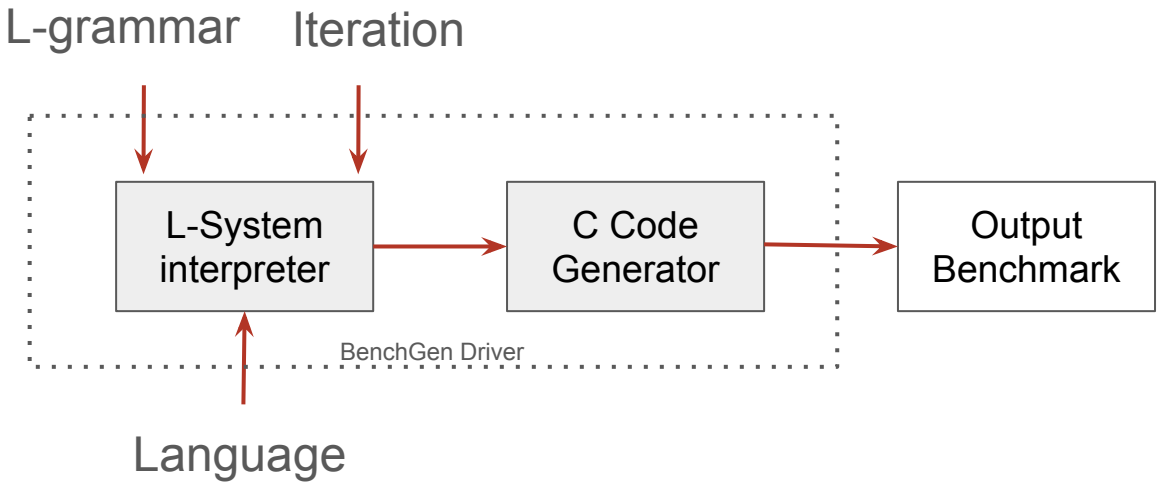
# Multi-Language Support

DCC

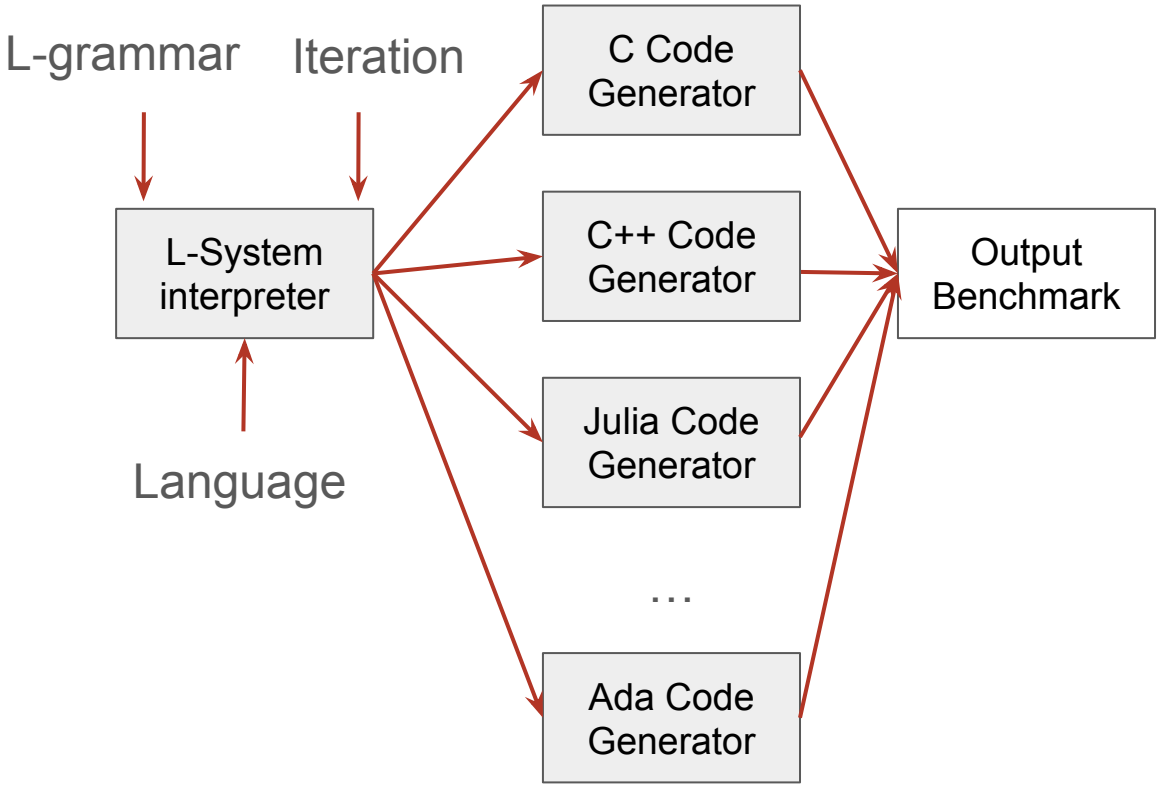
# Multi-Language Support



# Multi-Language Support



# Multi-Language Support



# Multi-Language Support

BenchGen supports eleven programming languages

DCC

# Multi-Language Support

DCC

BenchGen supports eleven programming languages

1. C
2. C++
3. Go
4. Julia
5. Odin
6. V
7. Zig
8. D
9. Nim
10. Ada
11. Pipefish

# Multi-Language Support

DCC

BenchGen supports one intermediate representation

1. MLIR

# Evaluation

DCC

# Evaluation

DCC

Goal:

# Evaluation

DCC

Goal: Demonstrate BenchGen's usability across different scenarios!

# Evaluation - CS1 gcc vs clang

DCC

# Evaluation - CS1 gcc vs clang

DCC

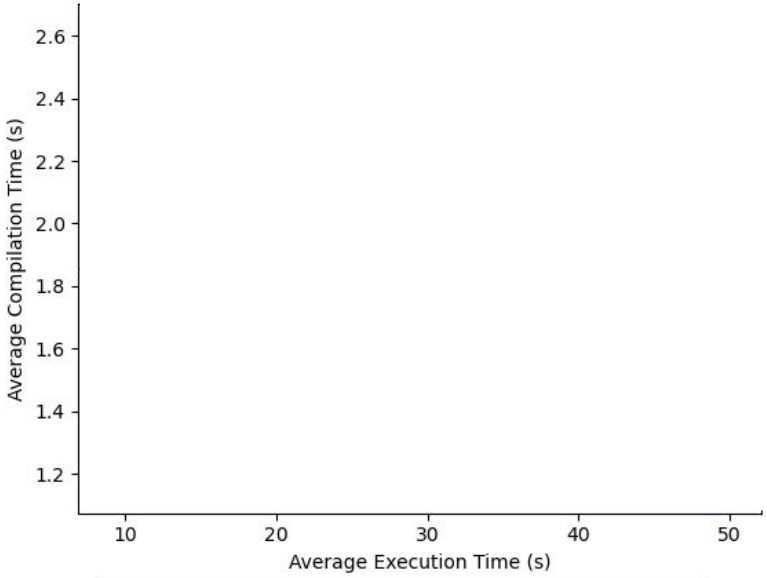
- gcc 14.2

# Evaluation - CS1 gcc vs clang

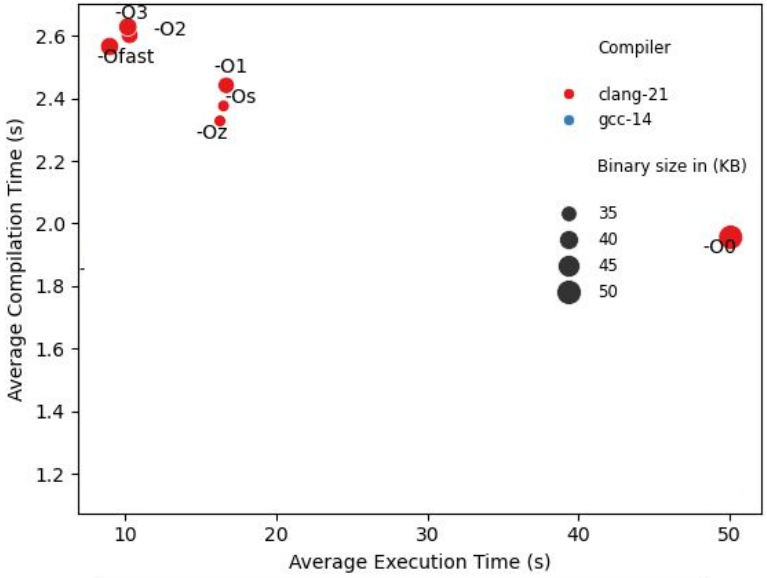
DCC

- gcc 14.2
- clang 21.0

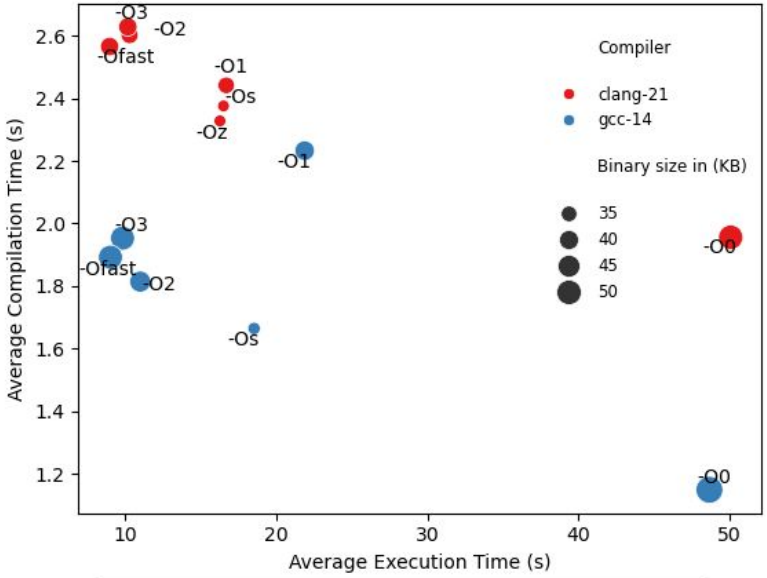
# Evaluation - CS1 gcc vs clang



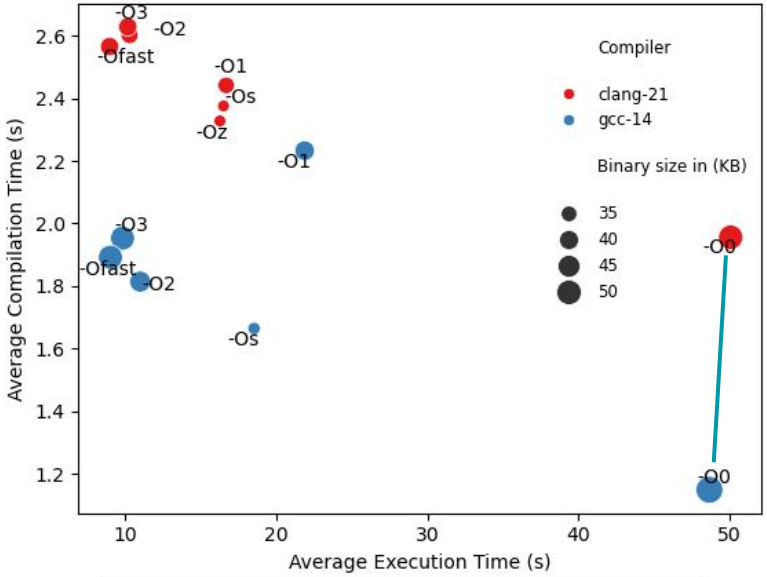
# Evaluation - CS1 gcc vs clang



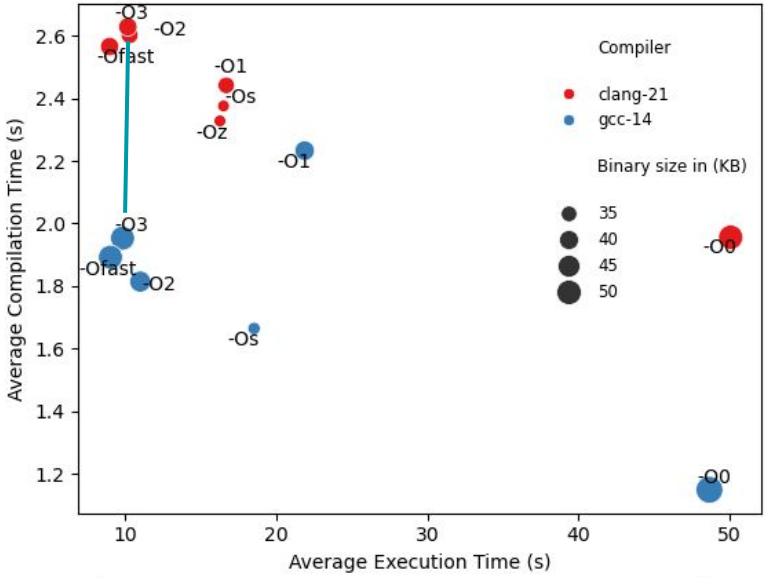
# Evaluation - CS1 gcc vs clang



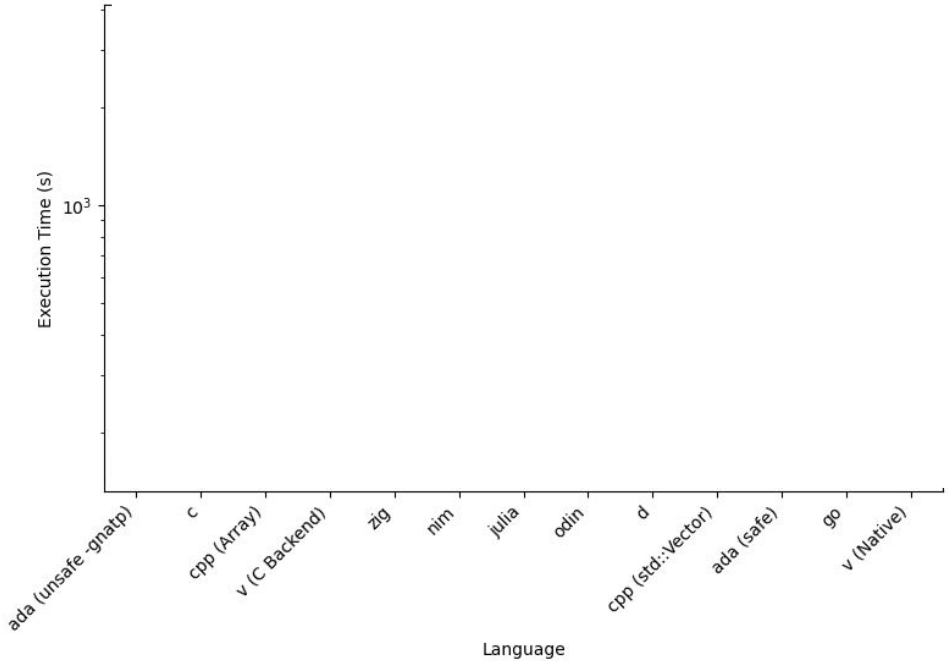
# Evaluation - CS1 gcc vs clang



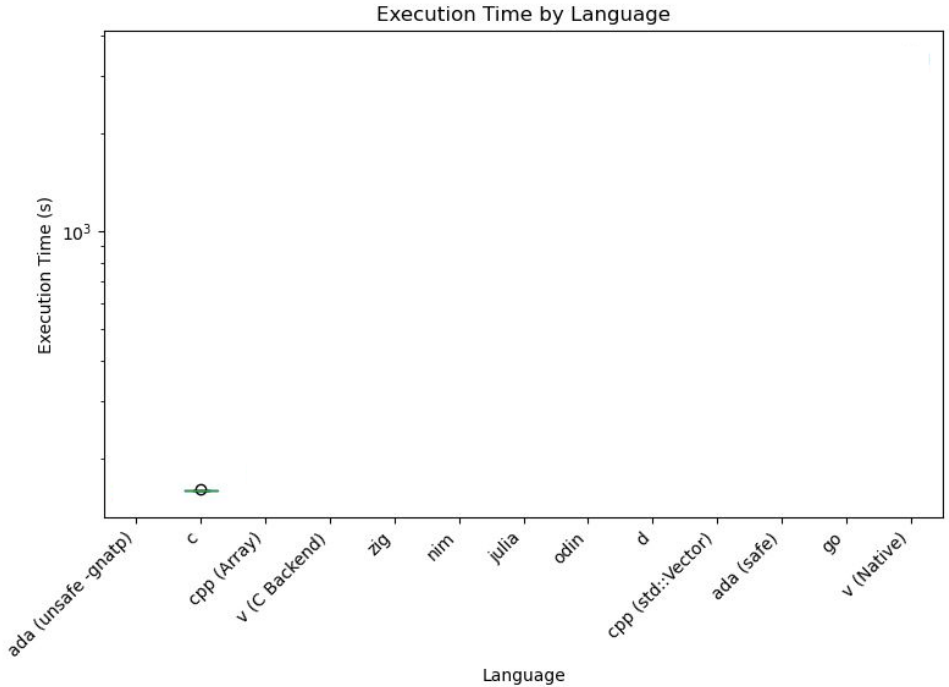
# Evaluation - CS1 gcc vs clang



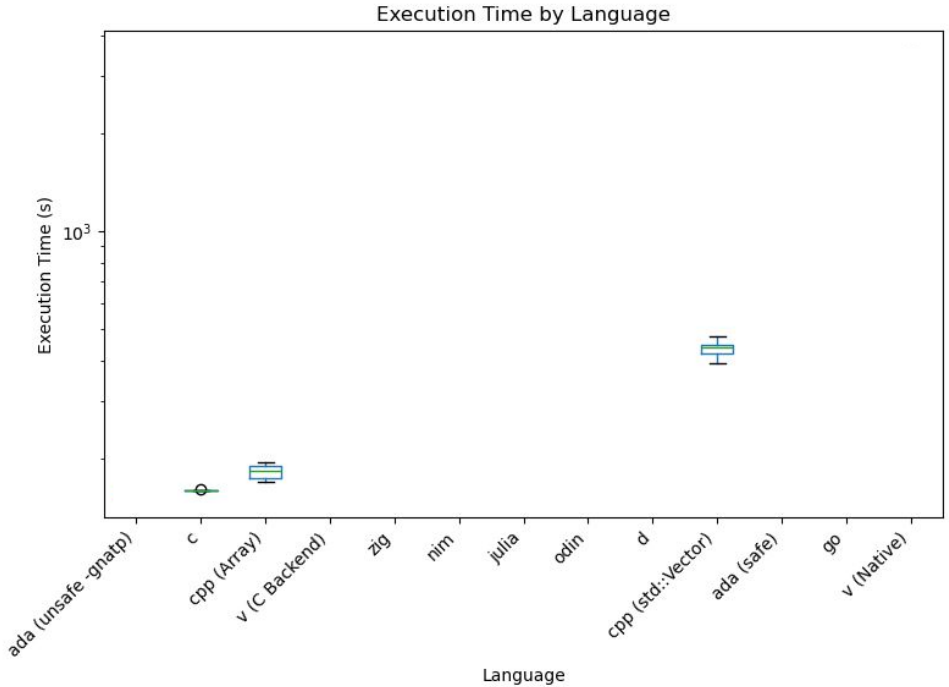
# Evaluation - CS2 Programming Languages Comparison



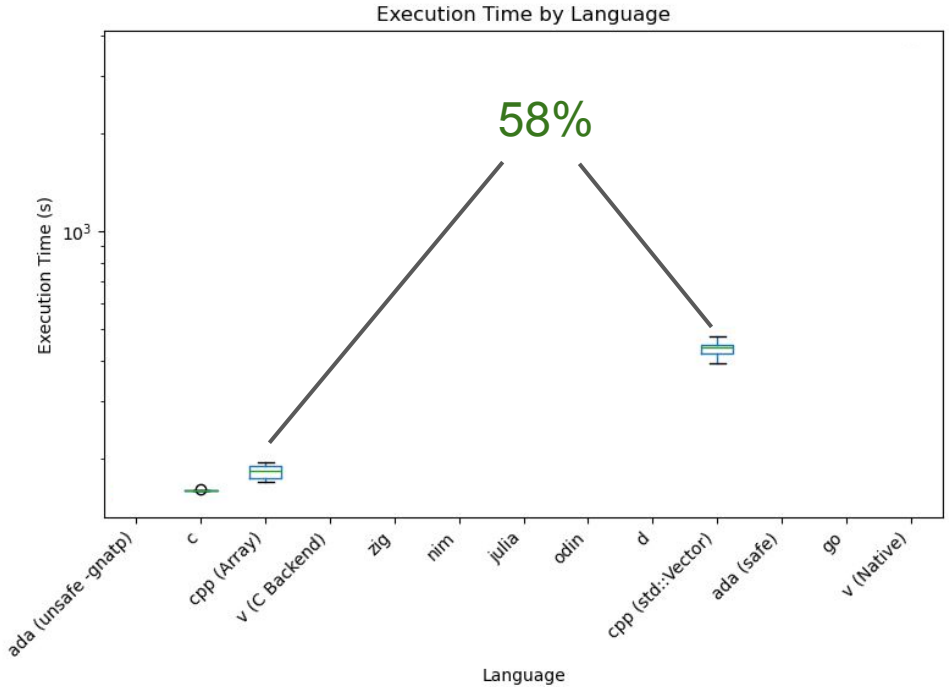
# Evaluation - CS2 Programming Languages Comparison



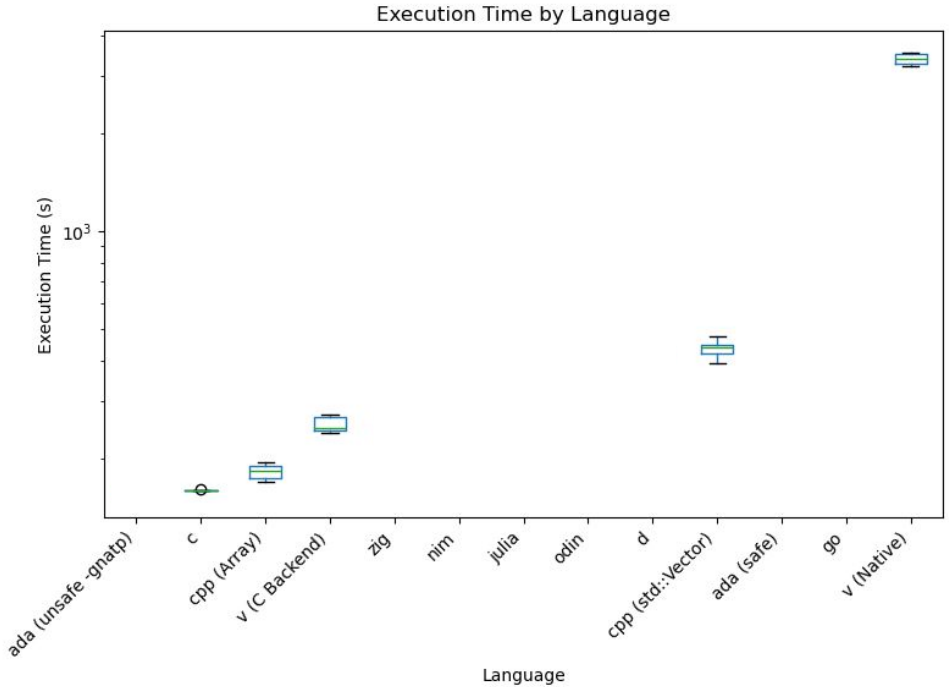
# Evaluation - CS2 Programming Languages Comparison



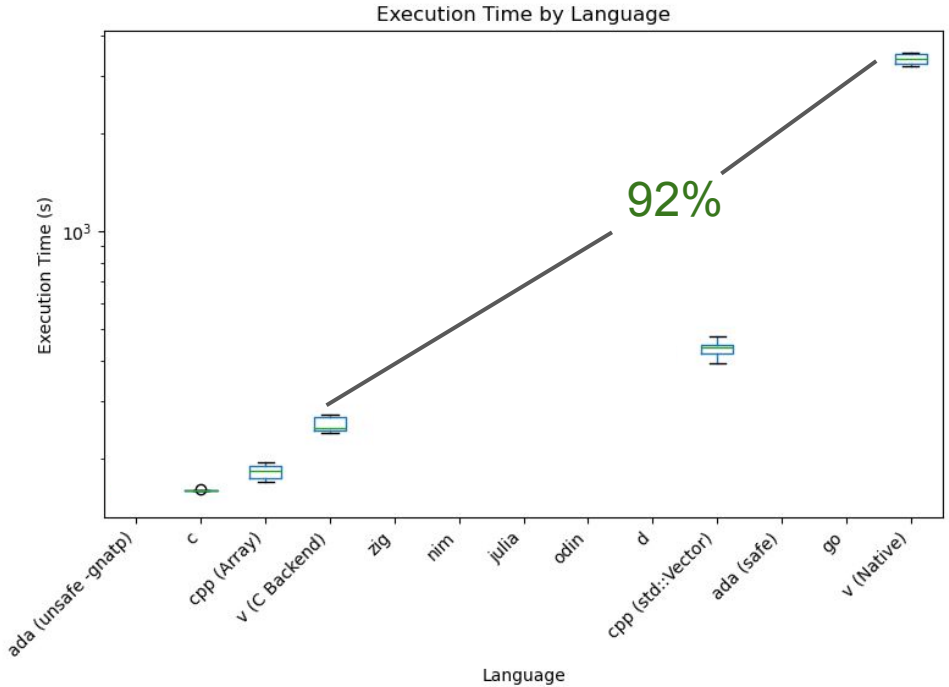
# Evaluation - CS2 Programming Languages Comparison



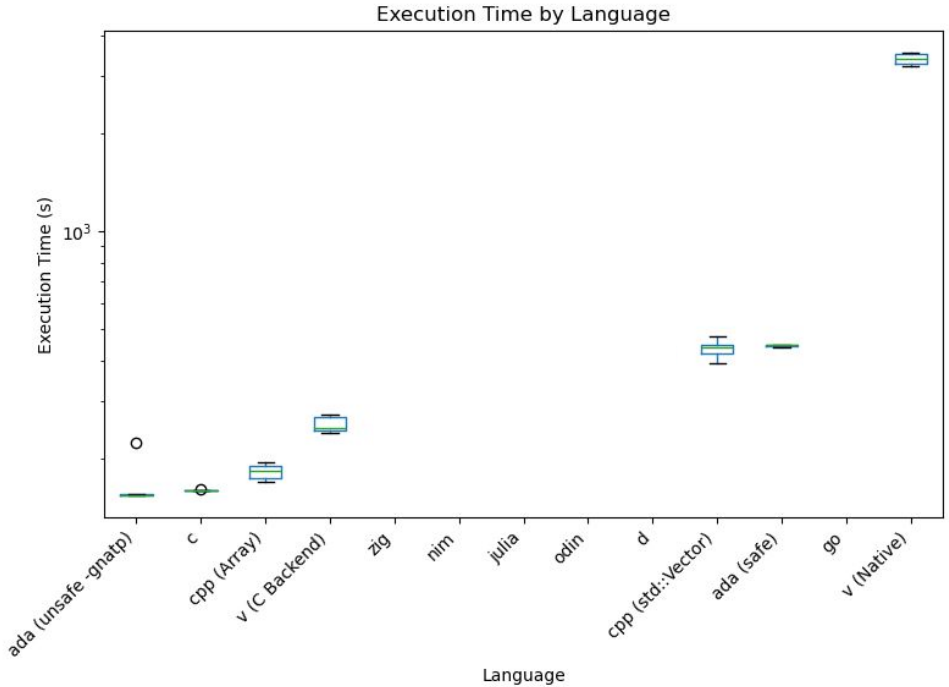
# Evaluation - CS2 Programming Languages Comparison



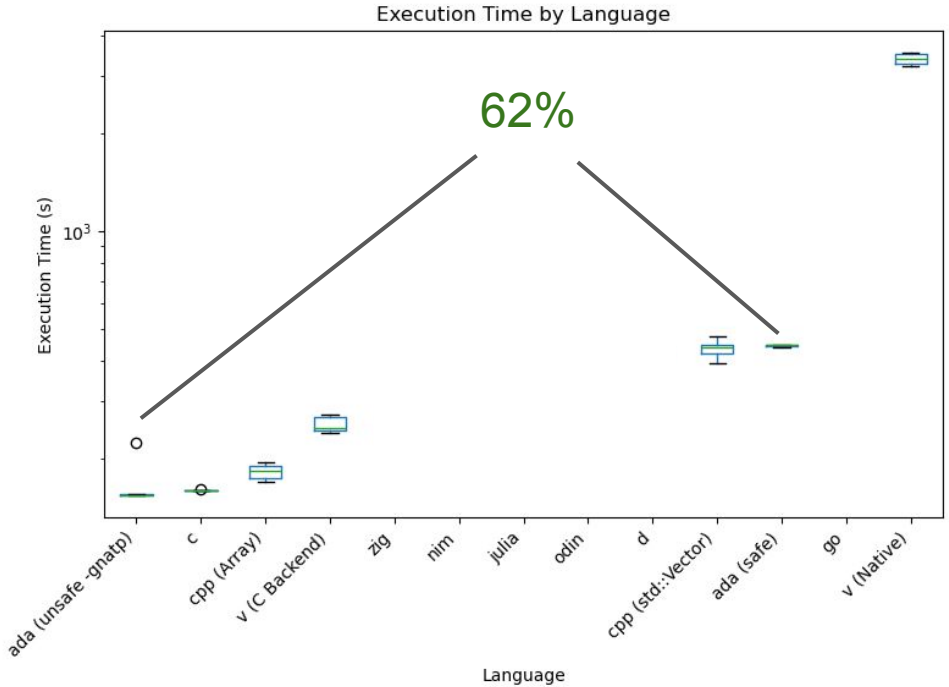
# Evaluation - CS2 Programming Languages Comparison



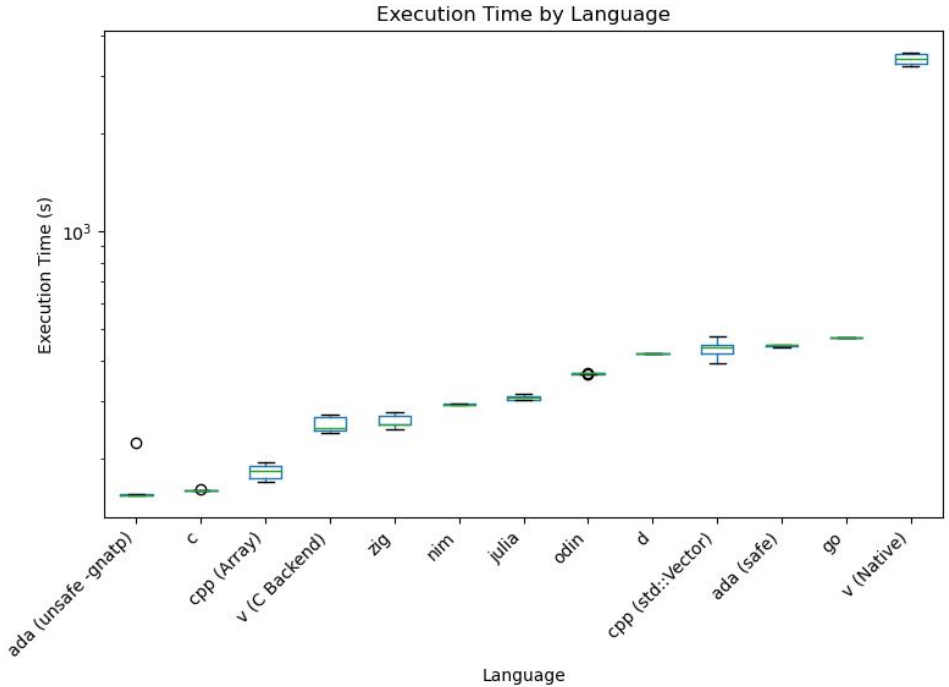
# Evaluation - CS2 Programming Languages Comparison



# Evaluation - CS2 Programming Languages Comparison



# Evaluation - CS2 Programming Languages Comparison

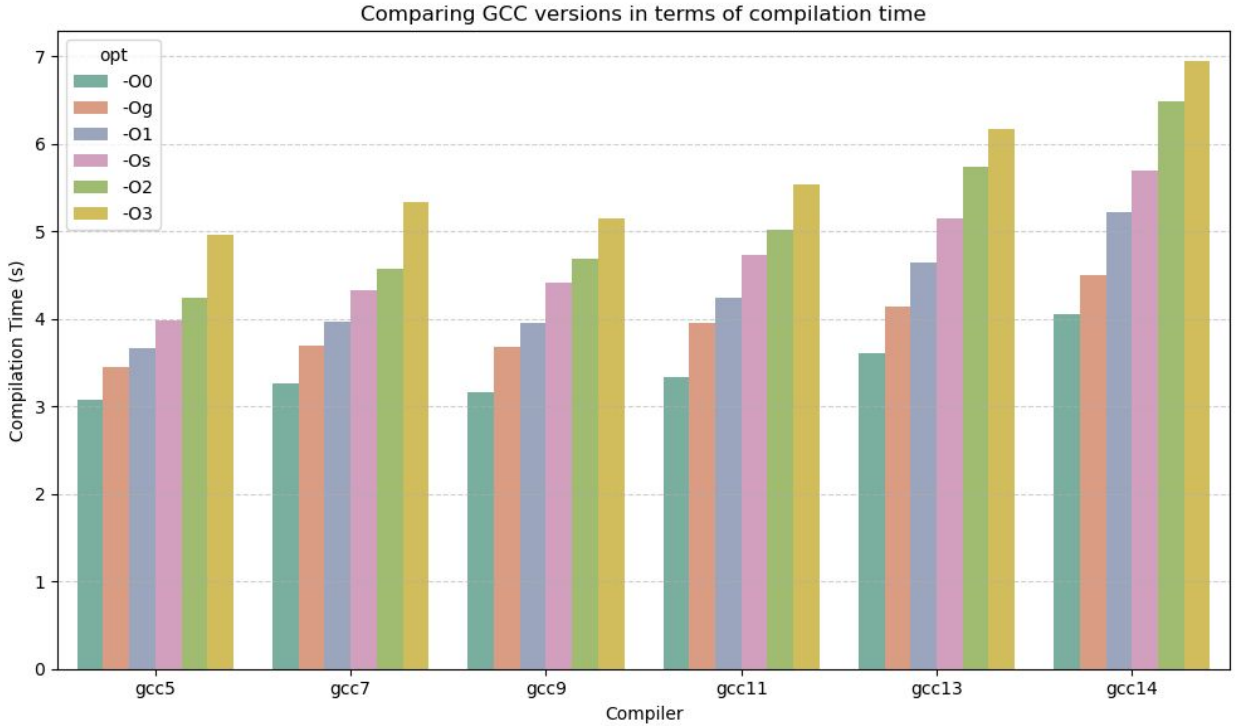


# Evaluation - CS4: The Evolution of gcc

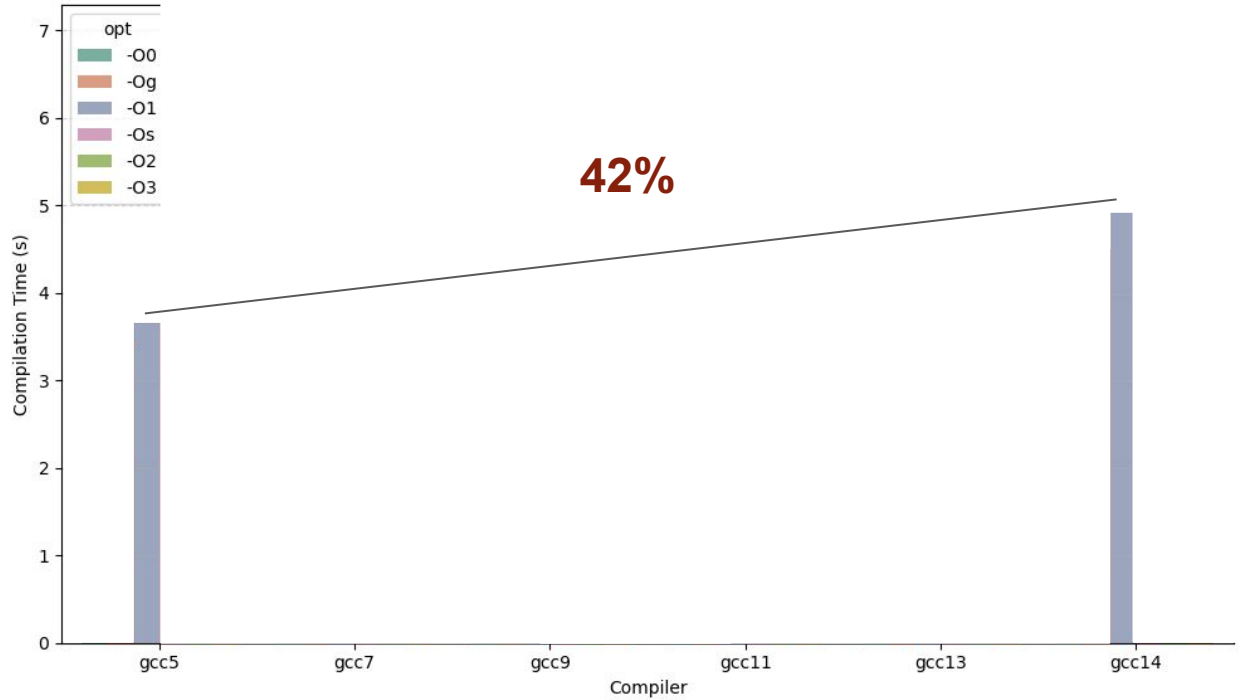
DCC



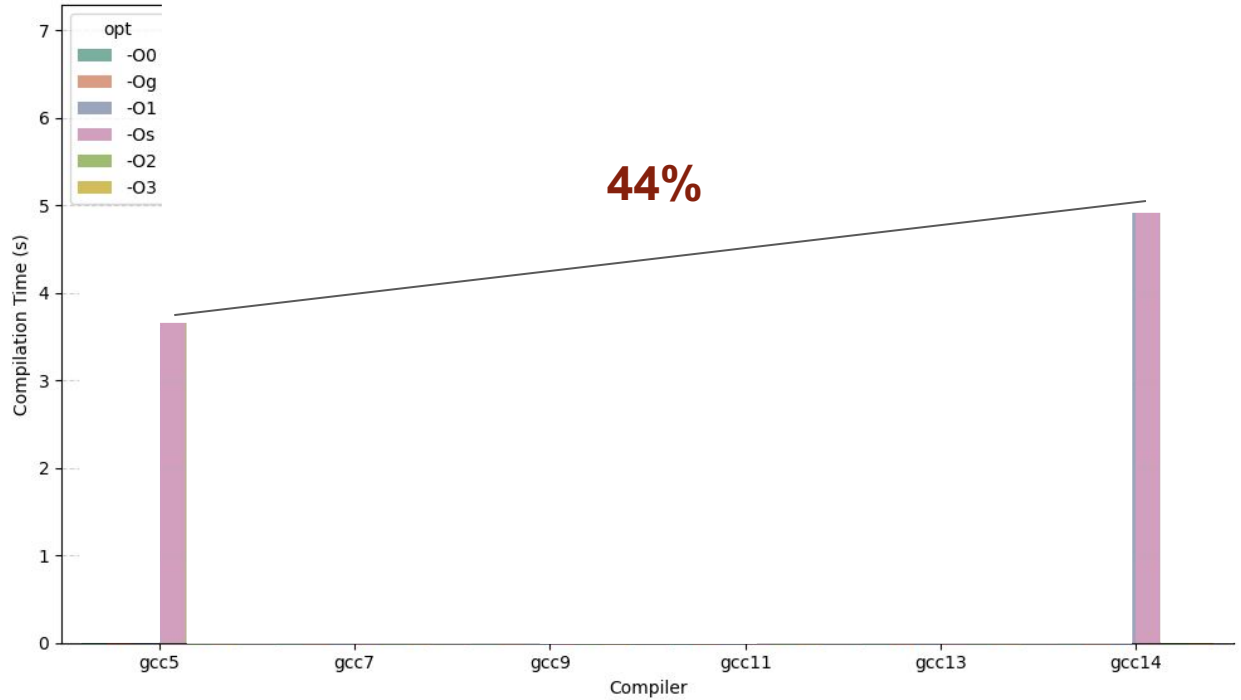
# Evaluation - CS4: The Evolution of gcc



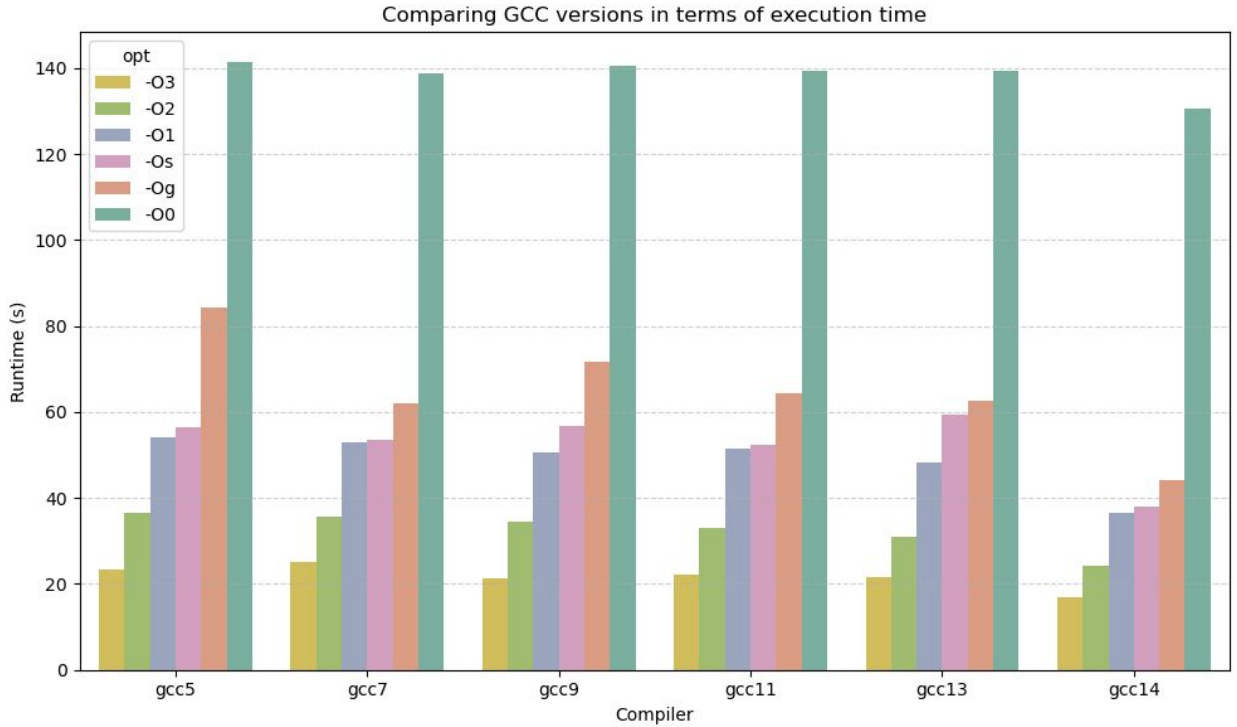
# Evaluation - CS4: The Evolution of gcc



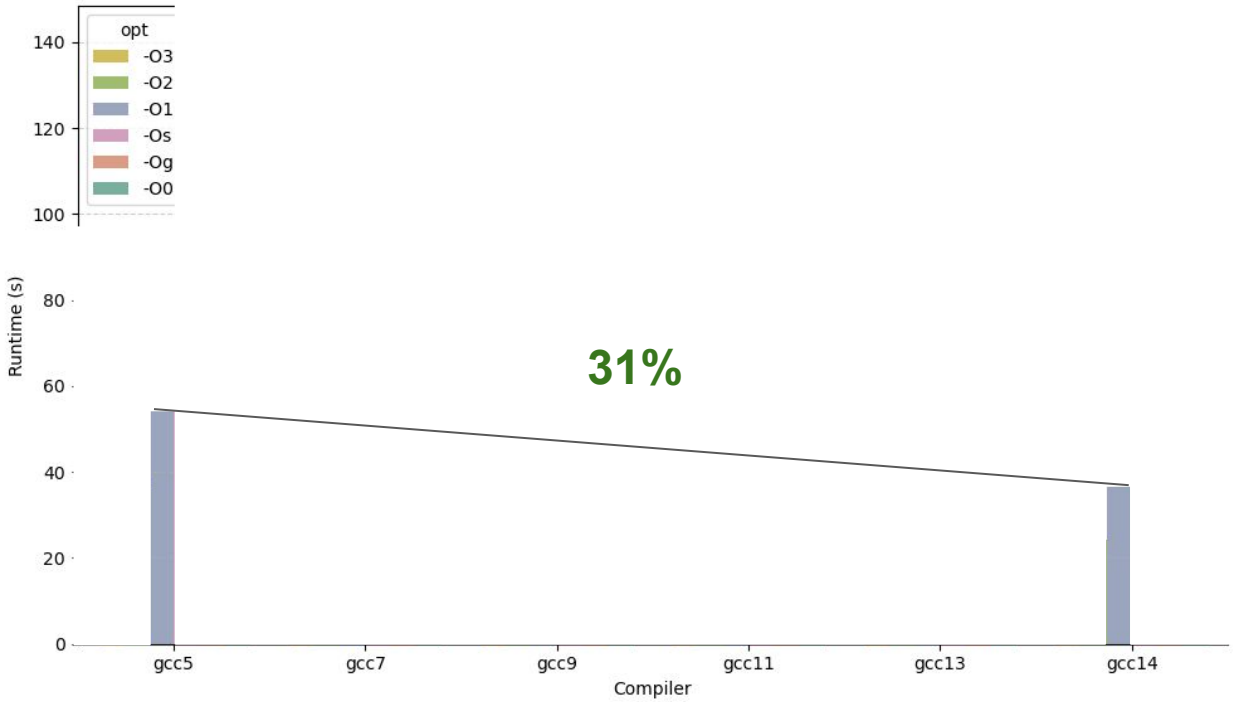
# Evaluation - CS4: The Evolution of gcc



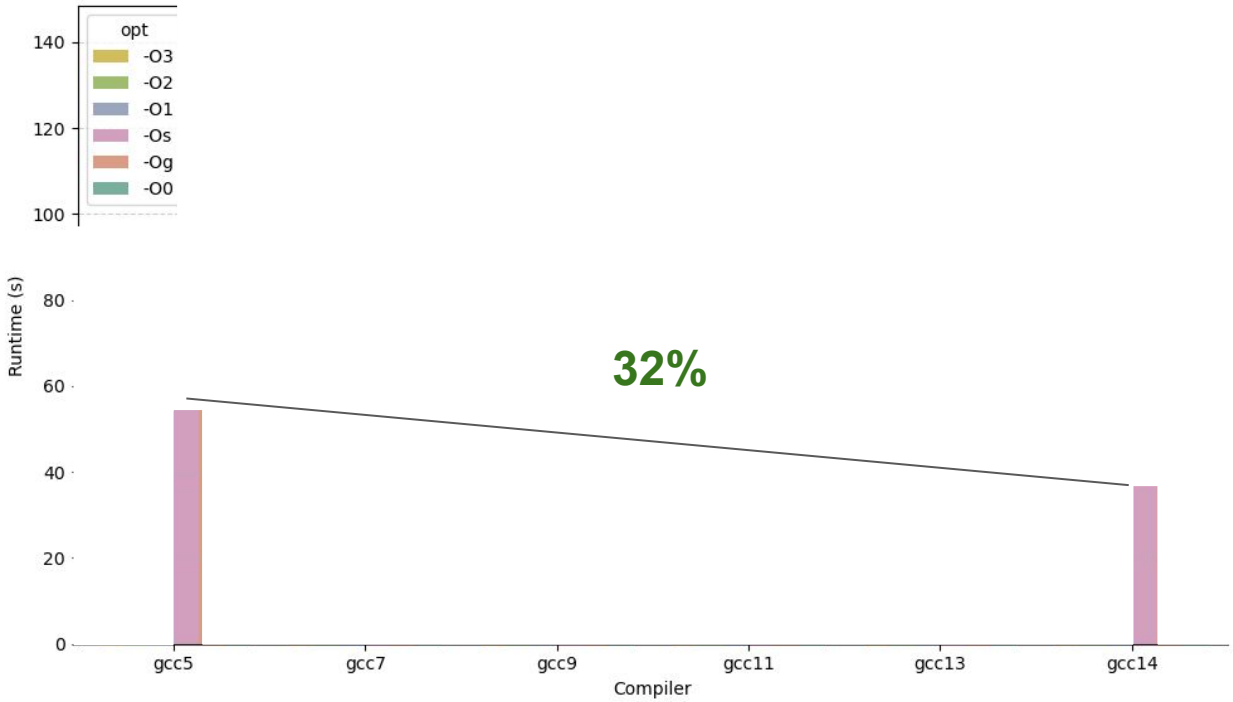
# Evaluation - CS4: The Evolution of gcc



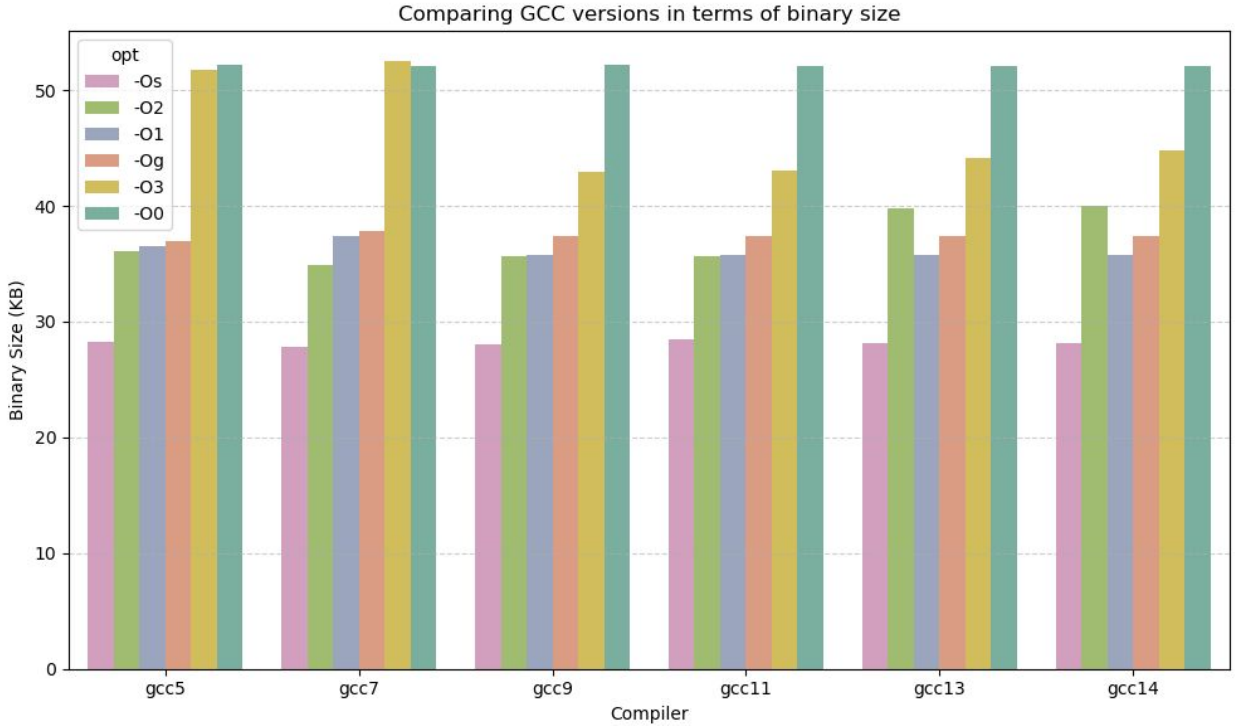
# Evaluation - CS4: The Evolution of gcc



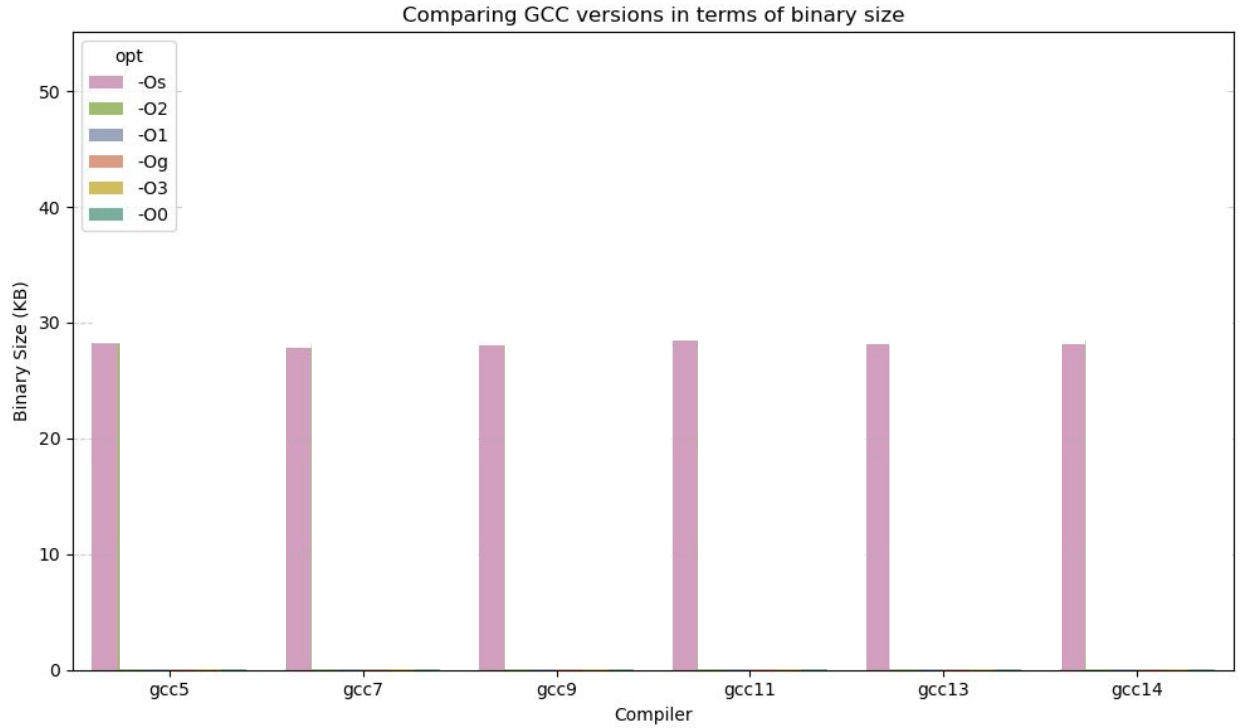
# Evaluation - CS4: The Evolution of gcc



# Evaluation - CS4: The Evolution of gcc



# Evaluation - CS4: The Evolution of gcc



# Conclusion

DCC

- L-System benchmark generator

# Conclusion

DCC

- L-System benchmark generator
- Multi-language support
- Diverse practical applications
- Generated programs are free of memory leaks

# Conclusion

DCC

- L-System benchmark generator
- Multi-language support
- Diverse practical applications
- Generated programs are free of memory leaks
- Heap memory limitation for large programs
- L-grammar vocabulary is very restricted
- Structural scalability is not equivalent to semantic diversity

# Contact

DCC

Vinicius Francisco da Silva - silva.vinicius@dcc.ufmg.br

Fernando Magno Quintão Pereira - fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

Project repository

https://github.com/lac-dcc/BenchGen/