UFMG

A **Range Analysis** Algorithm for the **Cangjie Compiler**



Bernardo Borges
bernborgess@gmail.com

Range Analysis

DCC

Value range analysis

Range Analysis

DCC

Value range analysis is a type of data flow analysis

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution.

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution. Ex.:

```
#include <stdio.h>

int main() {
   int i = 0;
   while (i < 4) {
       i++;
   }
   if (i != 4) {
       printf("I will never print!\n");
   }
   return 0;
}
```

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution. Ex.:

```
#include <stdio.h>

int main() {
    int i = 0;
    while (i < 4) {
        i++;
    }
    if (i != 4) {
        printf("I will never print!\n");
    }
    return 0;
}
```

$i \in \{0\}$

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution. ex:

```c
#include <stdio.h>

int main() {
    int i = 0;
    while (i < 4) {
        i++;
    }
    if (i != 4) {
        printf("I will never print!\n");
    }
    return 0;
}
```

Learn information about `i` in this branch

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution. ex:

```
#include <stdio.h>

int main() {
    int i = 0;
    while (i < 4) {
        i++;
    }
    if (i != 4) {
        printf("I will never print!\n");
    }
    return 0;
}
```

$i \in [0,3]$

Range Analysis

DCC

Value range analysis is a type of data flow analysis that tracks the range (interval) of values that a numeric variable can take on at each point of a program's execution. ex:

```c
#include <stdio.h>

int main() {
    int i = 0;
    while (i < 4) {
        i++;
    }
    if (i != 4) {
        printf("I will never print!\n");
    }
    return 0;
}
```

Learning $i \in \{4\}$ lets us conclude that this code is dead!

Range Analysis

DCC

Why do we need a new algorithm?

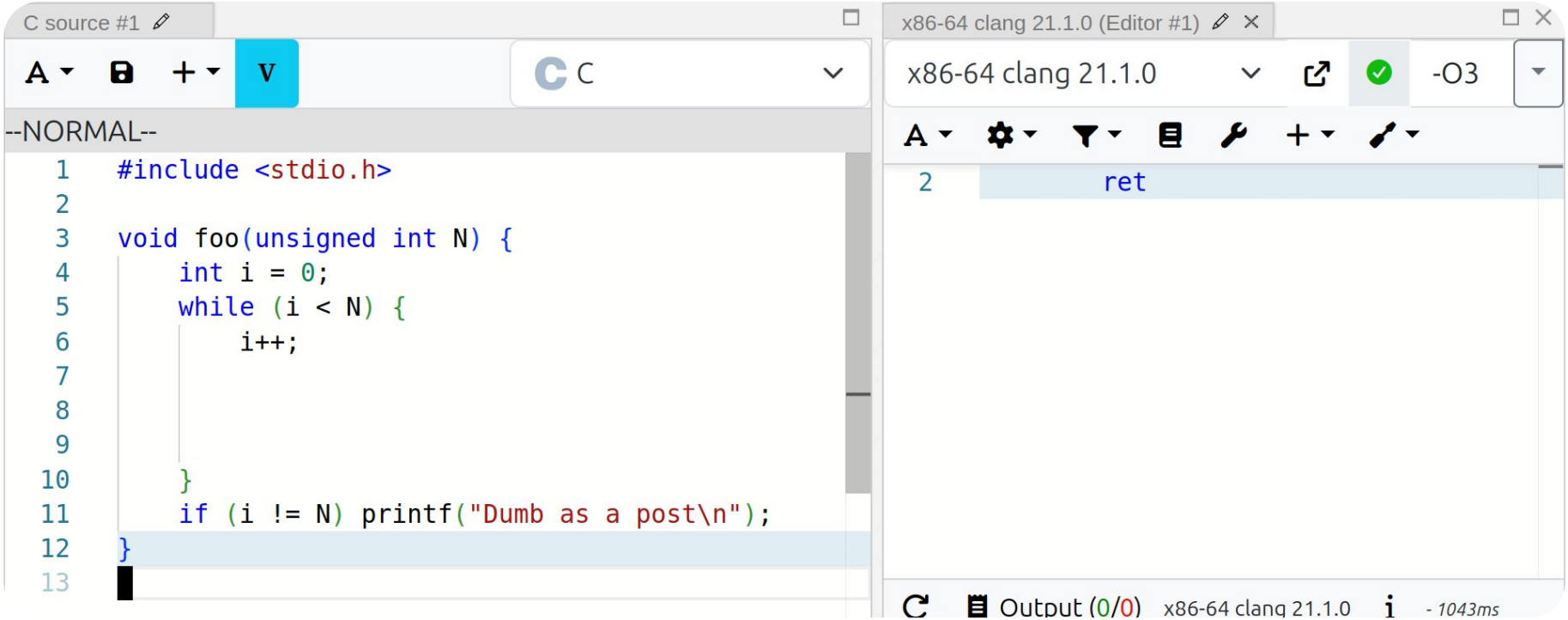
Range Analysis

DCC

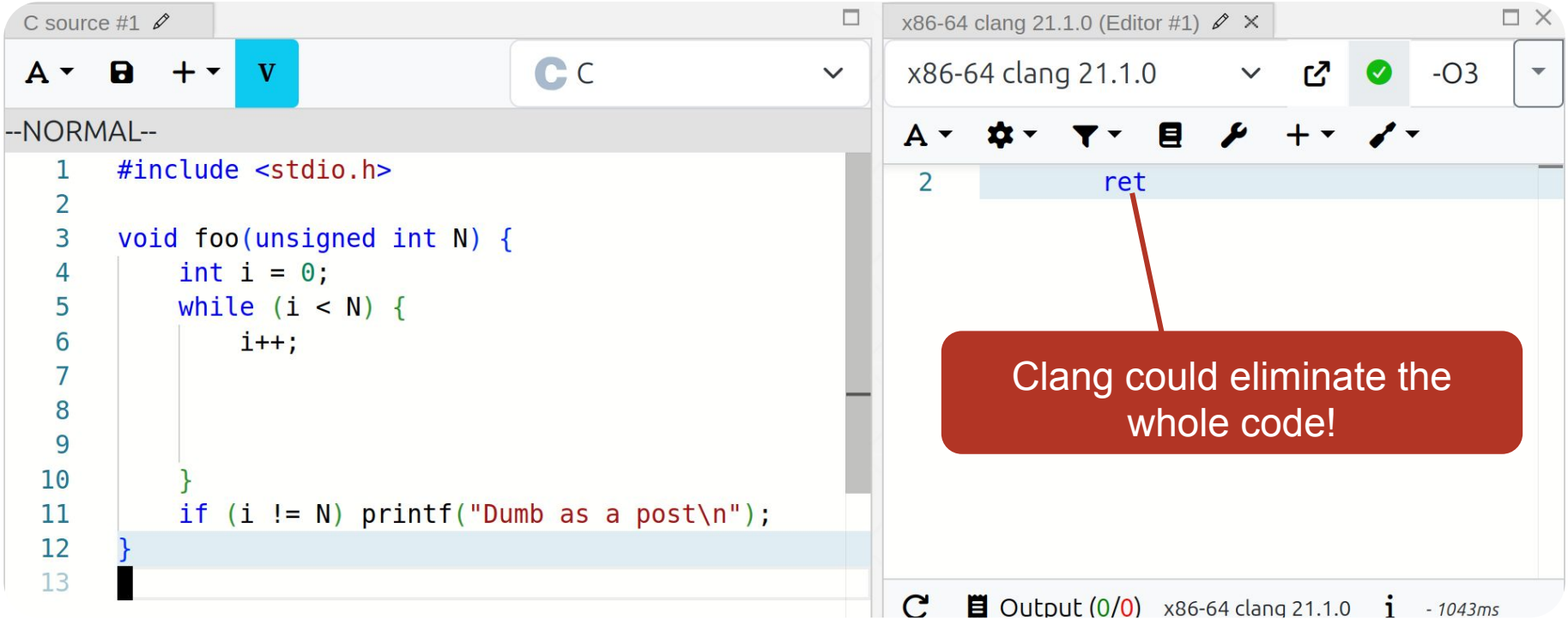
Why do we need a new algorithm? Clang/GCC can’t solve some cases!

Range Analysis

DCC



Range Analysis



Range Analysis

DCC

C source #1

C

--NORMAL--

```c
#include <stdio.h>

void foo(unsigned int N) {
    int i = 0;
    while (i < N) {
        i++;
        if (i < N) {
            i++;
        }
    }
    if (i != N) printf("Dumb as a post\n");
}
```

x86-64 clang 21.1.0 (Editor #1)

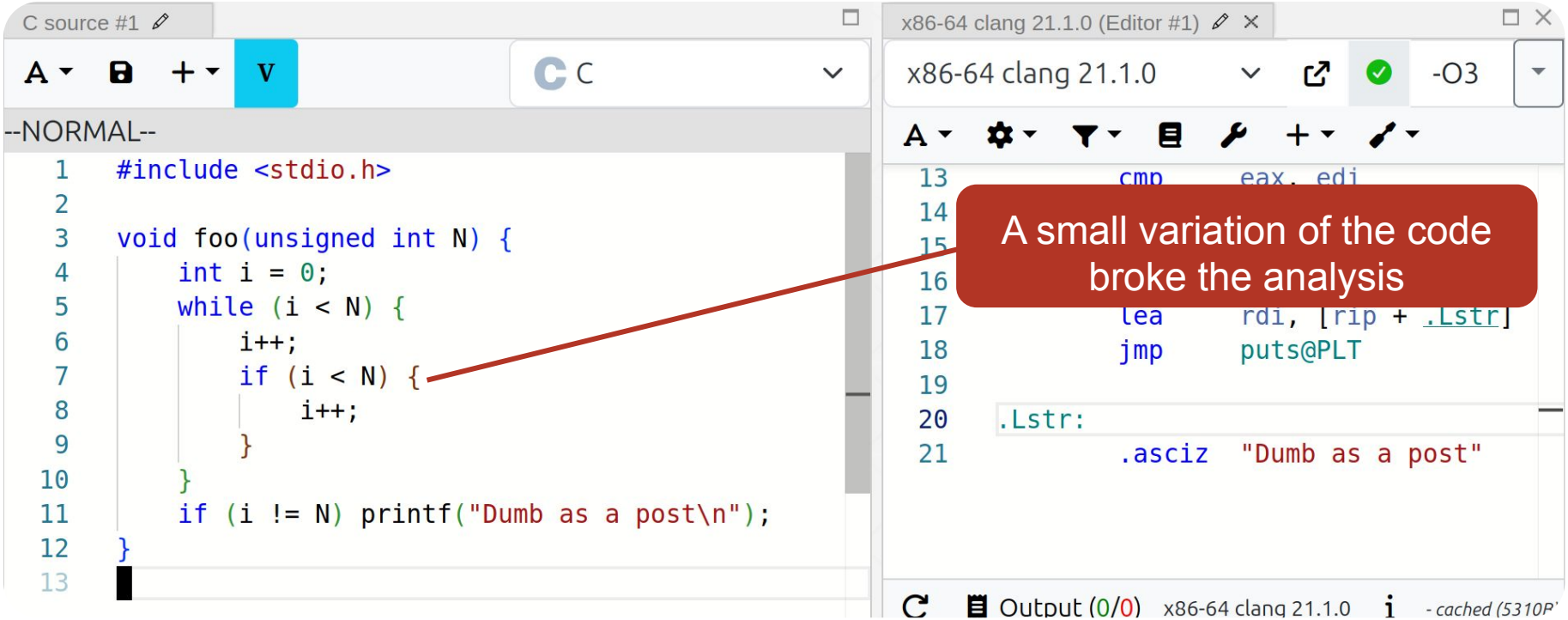
x86-64 clang 21.1.0 -O3

```
        cmp     eax, edi
        jne     .LBB0_4
        ret
.LBB0_4:
        lea     rdi, [rip + .Lstr]
        jmp     puts@PLT

.Lstr:
        .asciz  "Dumb as a post"
```

Output (0/0) x86-64 clang 21.1.0 - cached (5310P

Range Analysis

DCC



Cangjie Compiler

DCC

Cangjie (Chinese: 仓颉; pinyin: cāngjié)

Cangjie Compiler

DCC

Cangjie (Chinese: 仓颉; pinyin: cāngjié) is a high-level, **compiled** programming language developed by Huawei.

Cangjie Compiler

DCC

Cangjie (Chinese: 仓颉; pinyin: cāngjié) is a high-level, **compiled** programming language developed by Huawei.



https://github.com/cangjielanguage/cangjie_compiler

https://gitcode.com/Cangjie/cangjie_compiler



Cangjie Compiler

DCC



Cangjie Compiler

DCC

06. Huawei BiSheng Cup Cangjie Code Fragment Semantic Checks	Challenge
07. Huawei BiSheng Cup Cangjie High-Performance Range Analysis	Challenge
08. Huawei BiSheng Cup Cangjie Static Data-Race Detection	Challenge

Our Implementation

DCC

- The Lattice: Hybrid Abstract Domain
  - **Strided Intervals (> N):** Represent a range by the lower and upper bounds and a step
    - Ex: **[1, +∞:2]** *All positive odd numbers*

R. E. Rodrigues, V. H. Sperle Campos and F. M. Quintão Pereira
A fast and low-overhead technique to secure programs against integer overflows - CGO 2013

Our Implementation

DCC

- The Lattice: Hybrid Abstract Domain
  - **Strided Intervals (> N):** Represent a range by the lower and upper bounds and a step
    - Ex: **[1, +∞:2]** *All positive odd numbers*
  - **Finite Sets (≤ N):** Store a set with at most **N** exact reachable constants
    - Ex: **{ 1, 3, 8 }**

Example

DCC

```
func foo() {
    var x = 0

    while (x < 10) {

        x++
    }

    if (x != 10) {
        println("dead")
    }

    return i;
}
```

Example

DCC

- Convert Cangie programs to Extended Static Single Assignment form

```
func foo() {
    var x = 0

    while (x < 10) {

        x++
    }

    if (x != 10) {
        println("dead")
    }

    return x;
}
```

Example

DCC

- Convert Cangie programs to Extended Static Single Assignment form

```
func foo() {
    var x = 0

    while (x < 10) {

        x++
    }

    if (x != 10) {
        println("dead")
    }

    return x;
}
```



Example

DCC

- Convert Cangie programs to Extended Static Single Assignment form

```
func foo() {
    var x = 0

    while (x < 10) {

        x++
    }

    if (x != 10) {
        println("dead")
    }

    return x;
}
```



```
func foo() {
    var x0 = 0
    x1 = φ(x3, x0)
    while (x1 < 10) {
        x2 = σ(x1 ∩ [-∞,9])
        x3 = x2 + 1
    }
    x4 = σ(x1 ∩ [10,+∞])
    if (x4 != 10) {
        println("dead")
    }

    return x4;
}
```

Example

DCC

- Extract constraints from the e-SSA form program

```
func foo() {
    var x0 = 0
    x1 = φ(x3, x0)
    while (x1 < 10) {
        x2 = σ(x1 ∩ [-∞,9])
        x3 = x2 + 1
    }
    x4 = σ(x1 ∩ [10,+∞])
    if (x4 != 10) {
        println("dead")
    }

    return x4;
}
```

Example

DCC

- Extract constraints from the e-SSA form program

```
func foo() {
    var x0 = 0
    x1 = φ(x3, x0)
    while (x1 < 10) {
        x2 = σ(x1 ∩ [-∞,9])
        x3 = x2 + 1
    }
    x4 = σ(x1 ∩ [10,+∞])
    if (x4 != 10) {
        println("dead")
    }

    return x4;
}
```



Example

DCC

- Extract constraints from the e-SSA form program

```
func foo() {
    var x0 = 0
    x1 = φ(x3, x0)
    while (x1 < 10) {
        x2 = σ(x1 ∩ [-∞,9])
        x3 = x2 + 1
    }
    x4 = σ(x1 ∩ [10,+∞])
    if (x4 != 10) {
        println("dead")
    }

    return x4;
}
```



```
x0 = Constant(0)
x1 = φ(x3, x0)

x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```

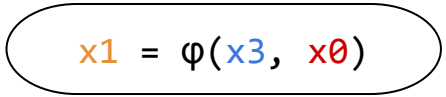
Example

DCC

- Build a constraint graph, sorted by Strongly Connected Components

```
x0 = Constant(0)

x1 = φ(x3, x0)
x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```









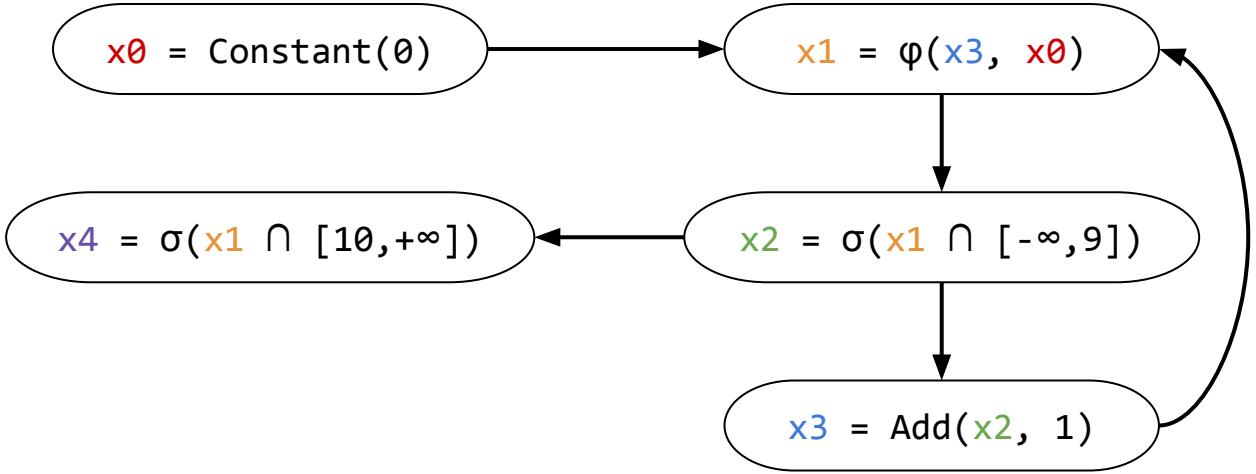


Example

DCC

- Build a constraint graph, sorted by Strongly Connected Components

```
x0 = Constant(0)

x1 = φ(x3, x0)
x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```

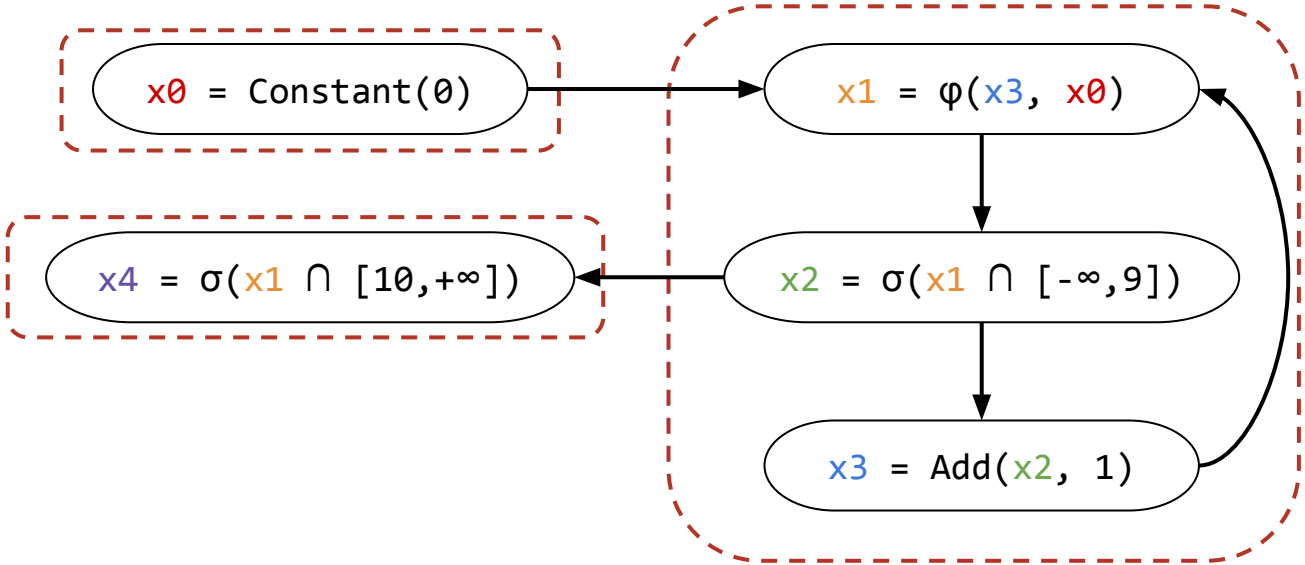


Example

DCC

- Build a constraint graph, sorted by Strongly Connected Components

```
x0 = Constant(0)

x1 = φ(x3, x0)
x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```

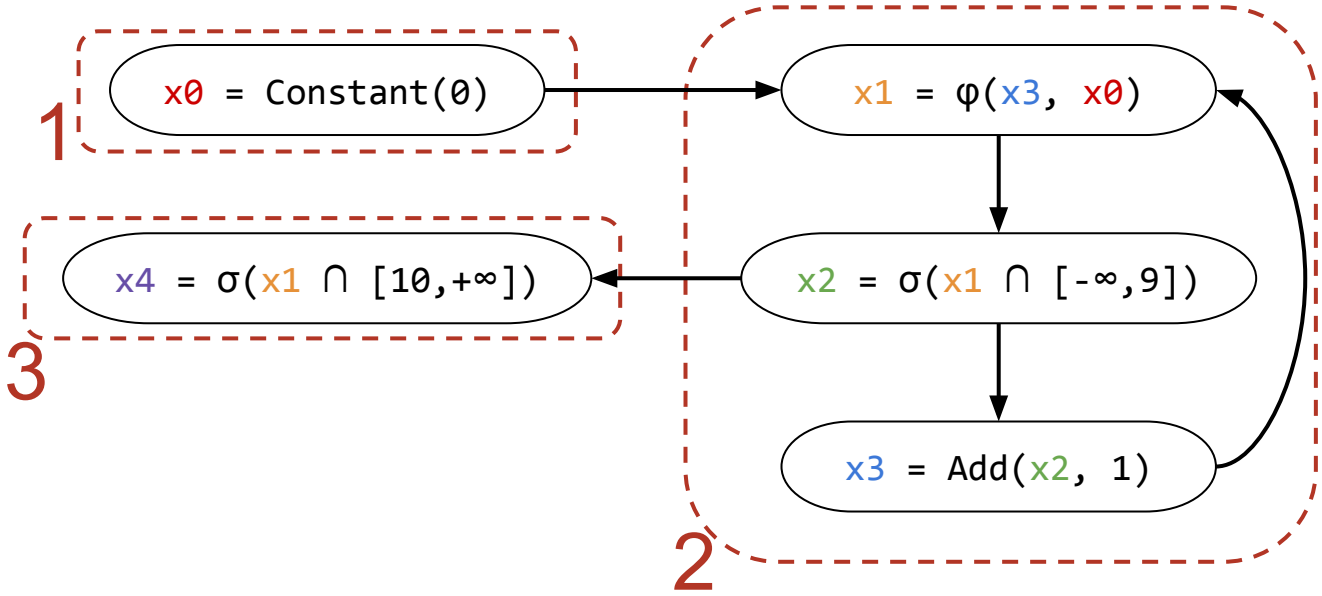


Example

DCC

- Build a constraint graph, sorted by Strongly Connected Components

```
x0 = Constant(0)

x1 = φ(x3, x0)
x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```



Example

DCC

- Solve the each SCC in topological order

```
x0 = Constant(0)              x0 ∈ {0}

x1 = φ(x3, x0)
x2 = σ(x1 ∩ [-∞,9])
x3 = Add(x2, 1)

x4 = σ(x1 ∩ [10,+∞])
```

Example

DCC

- Solve the each SCC in topological order

```
x0 = Constant(0)              x0 ∈ {0}

x1 = φ(x3, x0)                x1 ∈ [-∞, 10:1]
x2 = σ(x1 ∩ [-∞,9])           x2 ∈ [-∞, 9:1]
x3 = Add(x2, 1)               x3 ∈ [-∞, 10:1]

x4 = σ(x1 ∩ [10,+∞])
```

Example

DCC

- Solve the each SCC in topological order

```
x0 = Constant(0)              x0 ∈ {0}

x1 = φ(x3, x0)                x1 ∈ [-∞, 10:1]
x2 = σ(x1 ∩ [-∞,9])           x2 ∈ [-∞, 9:1]
x3 = Add(x2, 1)               x3 ∈ [-∞, 10:1]

x4 = σ(x1 ∩ [10,+∞])          x4 ∈ {10}
```

Example

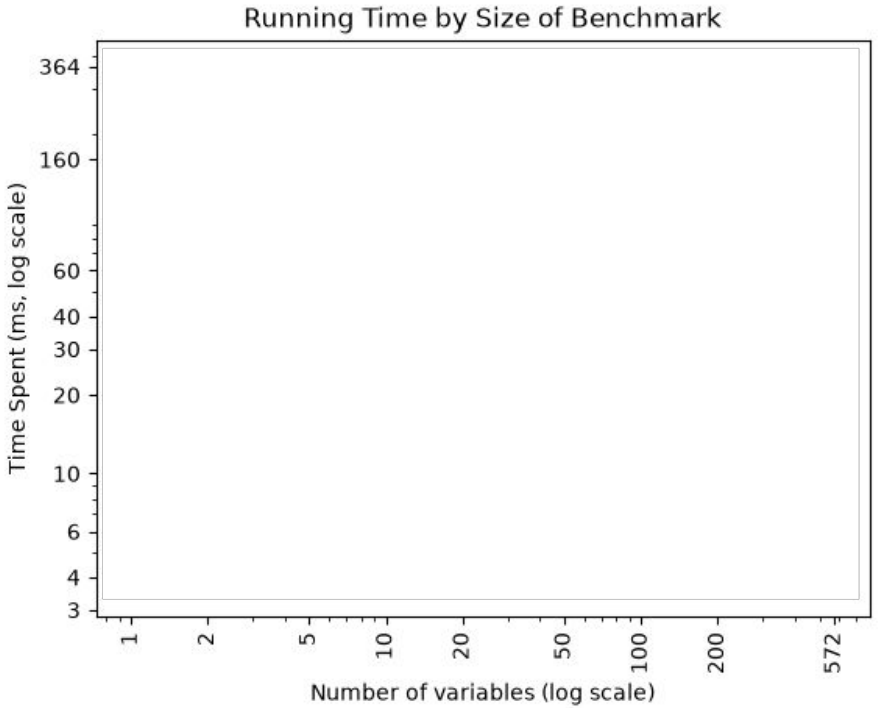
- Solve the each SCC in topological order

```
x0 = Constant(0)              x0 ∈ {0}

x1 = φ(x3, x0)                x1 ∈ [-∞, 10:1]
x2 = σ(x1 ∩ [-∞,9])           x2 ∈ [-∞, 9:1]
x3 = Add(x2, 1)               x3 ∈ [-∞, 10:1]

x4 = σ(x1 ∩ [10,+∞])          x4 ∈ {10}
```

```
if (x4 != 10) {
    println("dead")
}
```

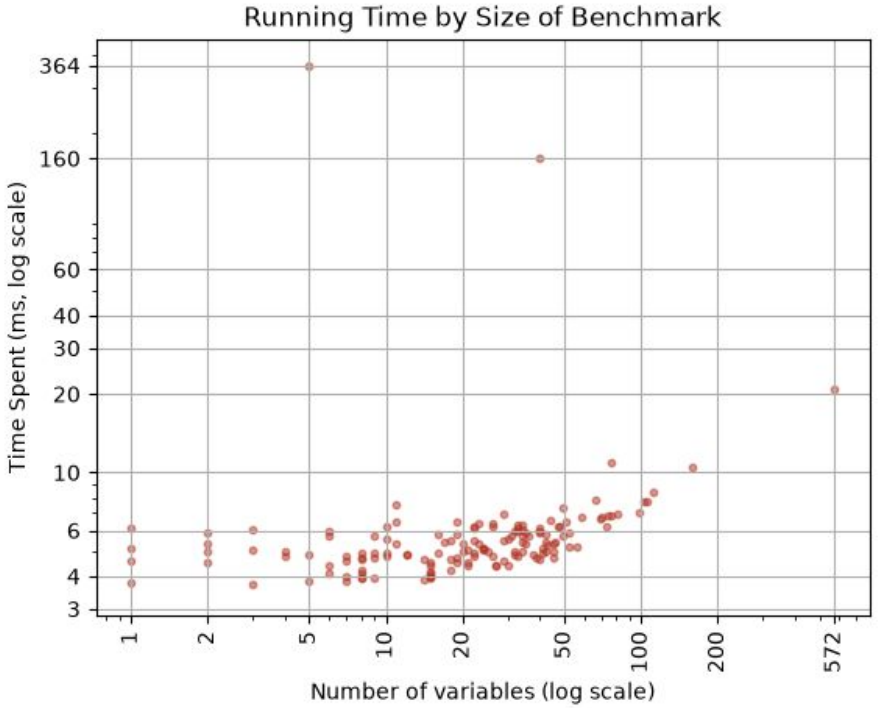
This condition can’t be true now!
We conclude this is dead code!

Benchmark



Code samples from github.com/cjhCoder7/CangjieBench

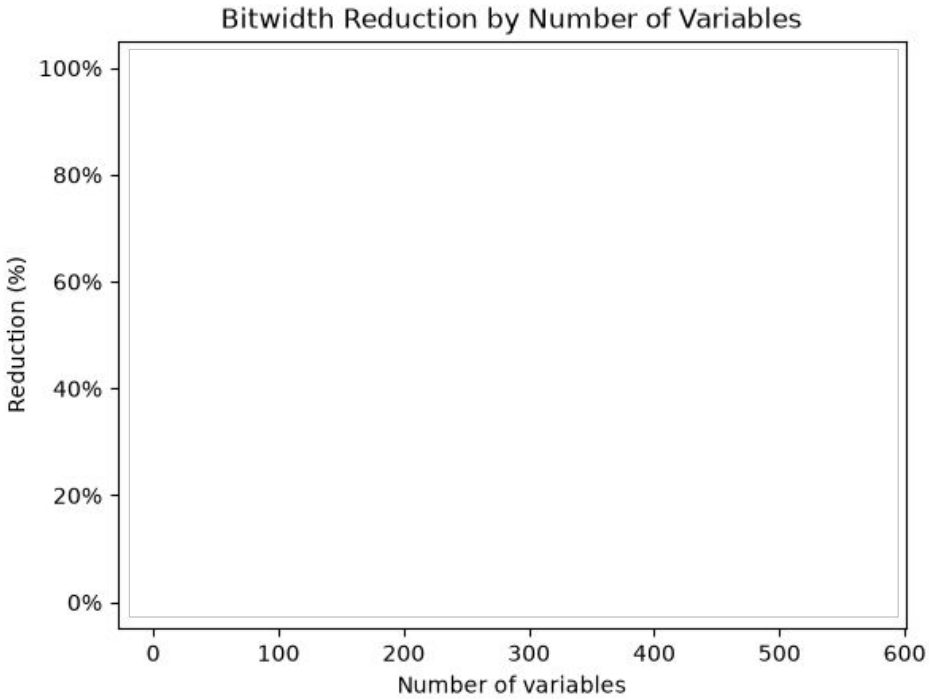
Benchmark



Code samples from github.com/cjhCoder7/CangjieBench

Benchmark

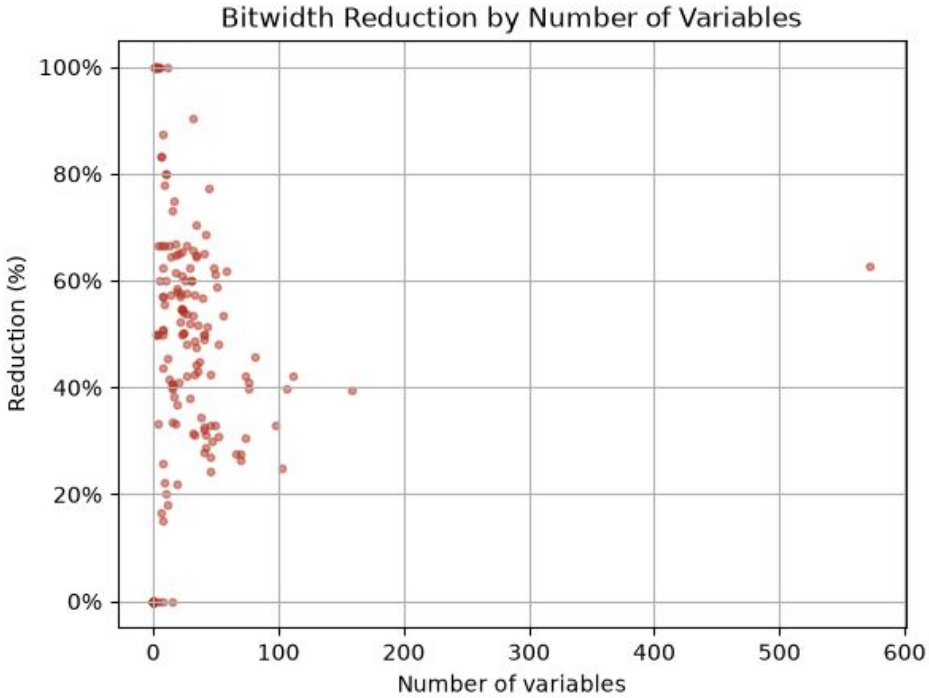
Obtained Reduction



Benchmark

DCC

Obtained Reduction



Future work

DCC

- Affine relations between μ-functions (φ-functions in loop headers)

```
void foo(const int N) {
    int i, j = 0;
    for (i = 0; i < N; ++i) {
        j += 3;
        if (N % 2) {
            i++;
            j += 3;
        }
    }
    if (3 * i != j) {
        printf("Dumb as a post\n");
    }
}
```

Capture the *affine relation* between `i` and `j`, that is preserved through the loop

References & Contact

DCC

References

Ernani et al.
A fast and low-overhead technique to secure programs [..].
CGO 2013

Tarjan et al.
A fast algorithm for finding dominators in a flowgraph.
TOPLAS 1979.

Cytron et al.
Efficiently computing SSA form [..] control dependence graph.
TOPLAS 1991

Value range analysis - Wikipedia
Cangjie (programming language) - Wikipedia

Contact

Bernardo Nogueira Borges

bernardoborges@dcc.ufmg.br

https://bernborgess.github.io

More Resources

GitHub repository