



南京大學
NANJING UNIVERSITY

Unveiling Compiler Faults via Attribute-Guided Compilation Space Exploration

Jiangchang Wu, Yibiao Yang, Maolin Sun, Yuming Zhou

State Key Laboratory for Novel Software Technology

Nanjing University

07/09/2025

Compilers are fundamental to software system



**Operating
System Kernel**



**Database
Management System**



**android
Android
Application**



**Deep Learning
Framework**



.....

But compilers are prone to bugs

The screenshot shows the GCC Bugzilla interface. At the top, a navigation bar includes 'Home', 'New', 'Browse', and 'Search'. Below this, a search bar is visible. A large red text overlay in the upper center reads '120K GCC bug reports'. Below the search bar, a table of bug reports is displayed. A second red text overlay, '80K LLVM issue reports', is positioned over the table. The table columns include 'ID', 'Product', 'Comp', 'Assignee', 'Status', 'Resolution', 'Summary', and 'Changed'. The 'Summary' column contains several entries, such as 'Bitcode and textual IR have different BB predecessor order/LoopVectorizer output is sensitive to predecessor order' and '[clang] Assertion !Fang.isAmbiguous() && "Cannot handle ambiguities here yet" failed in clang::NamedDecl* clang::Sema::FindFirstQualifierInScope(...)'. The 'Changed' column shows a vertical timeline of timestamps from 17:24:11 to 07:11:57. A sidebar on the left lists bug IDs from 120952 down to 120923. A central banner asks 'Want to contribute to llvm/llvm-project?' with a 'Dismiss' button. Below the banner, there are filters for 'is:issue', 'Labels', 'Milestones', and 'New issue'. A summary bar shows 'Open 25,744' and 'Closed 59,093' bugs. The main content area lists individual bug reports with their IDs, titles, and authors.

120K GCC bug reports

80K LLVM issue reports

Want to contribute to llvm/llvm-project?

is:issue

Open 25,744 Closed 59,093

- Bitcode and textual IR have different BB predecessor order/LoopVectorizer output is sensitive to predecessor order **llvm:core**
- [flang][OpenMP] flang core dump when using cos or sin **flang**
- [flang][OpenMP] Ptx assembly aborted when calling exp **flang**
- [mir] Error message for optional properties is confusing **mir**
- [BOLT][AArch64] After BOLT optimization, -so crashes and the call information cannot be viewed. **BOLT**
- lld produces elf with corrupt string table index **lld**
- [clang] Assertion !Fang.isAmbiguous() && "Cannot handle ambiguities here yet" failed in clang::NamedDecl* clang::Sema::FindFirstQualifierInScope(...) **clang:frontend** **crash-on-invalid**
- [clang] Assertion Type <= 3 && "unexpected type" failed in bool {anonymous}::IntExprEvaluator::VisitBuiltinCallExpr(const clang::CallExpr*, unsigned int) **clang:frontend**

But compilers are prone to bugs

The screenshot shows the GCC Bugzilla interface. At the top, it says "GCC Bugzilla - Bug List". Below that, there are navigation links: "Home | New | Browse | Search". A search bar is present. The main content area displays a list of bugs. A large red text overlay reads "120K GCC bug reports". Another large red text overlay reads "80K LLVM issue reports". The bug list includes columns for ID, Product, Comp, Assignee, Status, Resolution, Summary, and Changed. The first bug listed is #120952, titled "Bitcode and textual IR have different BB predecessor order/LoopVectorizer output is sensitive to predecessor order". Other bugs listed include issues with flang, mir, BOLT, and clang. A sidebar on the right shows a list of times from 17:24:11 to 07:11:57.

Code complexity is high

Support a wide range of platforms

Optimization algorithm are complex

.....

But compilers are prone to bugs

The screenshot shows the GCC Bugzilla interface. At the top, it says "GCC Bugzilla - Bug List". Below that, there's a search bar and navigation links. A large red text overlay reads "120K GCC bug reports". Below this, there's a table of bug reports with columns for ID, Product, Comp, Assignee, Status, Resolution, and Summary. A second red text overlay reads "80K LLVM issue reports". The table lists several bugs, including:

- Bitcode and textual IR have different BB predecessor order/LoopVectorizer output is sensitive to predecessor order (llvm.core)
- [flang][OpenMP] flang core dump when using cos or sin (flang)
- [flang][OpenMP] Ptx assembly aborted when calling exp (flang)
- [mir] Error message for optional properties is confusing (mir)
- [BOLT][AArch64] After BOLT optimization, -so crashes and the call information cannot be viewed. (BOLT)
- lld produces elf with corrupt string table index (lld)
- [clang] Assertion !Found.isAmbiguous() && "Cannot handle ambiguities here yet" failed in clang::NamedDecl* clang::Sema::FindFirstQualifierInScope(clang::Scope*, clang::NestedNameSpecifier*) (clang:frontend, crash-on-invalid)
- [clang] Assertion Type <= 3 && "unexpected type" failed in bool (anonymous)::IntExprEvaluator::VisitBuiltinCallExpr(const clang::CallExpr*, unsigned int) (clang:frontend)

Code complexity is high

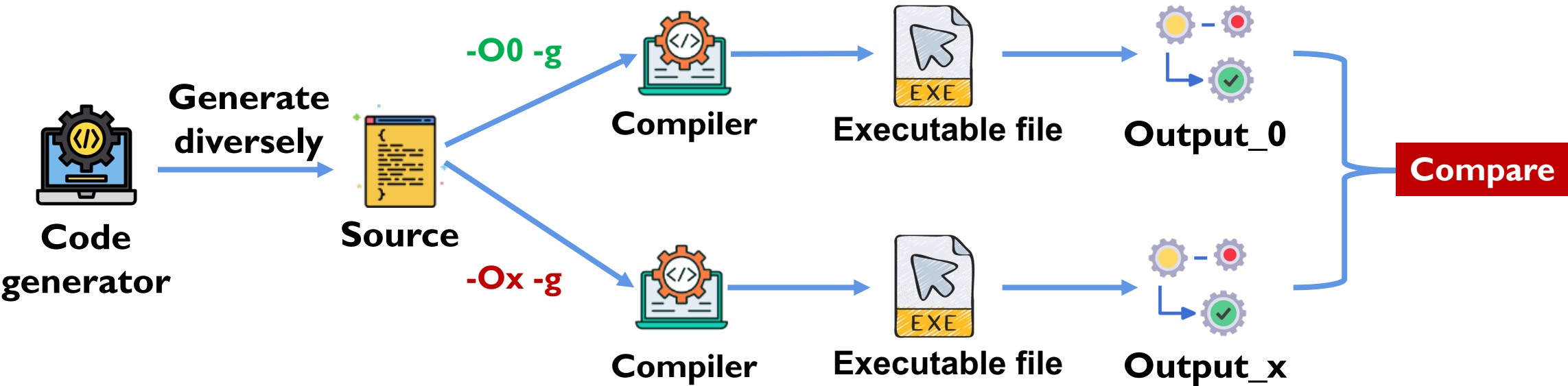
Support a wide range of platforms

Optimization algorithm are complex

.....

Compiler bugs can seriously impact system correctness, ensuring compiler reliability and correctness is crucial

Existing approaches for compiler testing



Existing approaches for compiler testing

gcc -O0 source.c -o source && ./source $\xrightarrow{\text{output}}$ 1

gcc -O2 source.c -o source && ./source $\xrightarrow{\text{output}}$ 0



Existing approaches for compiler testing

```
gcc -O0 source.c -o source && ./source  $\xrightarrow{\text{output}}$  1
```

```
gcc -O2 source.c -o source && ./source  $\xrightarrow{\text{output}}$  1
```

```
gcc -O2 -ftree-loop-distribution source.c  $\xrightarrow{\text{output}}$  0
```

```
-o source && ./source
```



Existing approaches for compiler testing

```
gcc -O0 source.c -o source && ./source  $\xrightarrow{\text{output}}$  1
```

```
gcc -O2 source.c -o source && ./source  $\xrightarrow{\text{output}}$  1
```

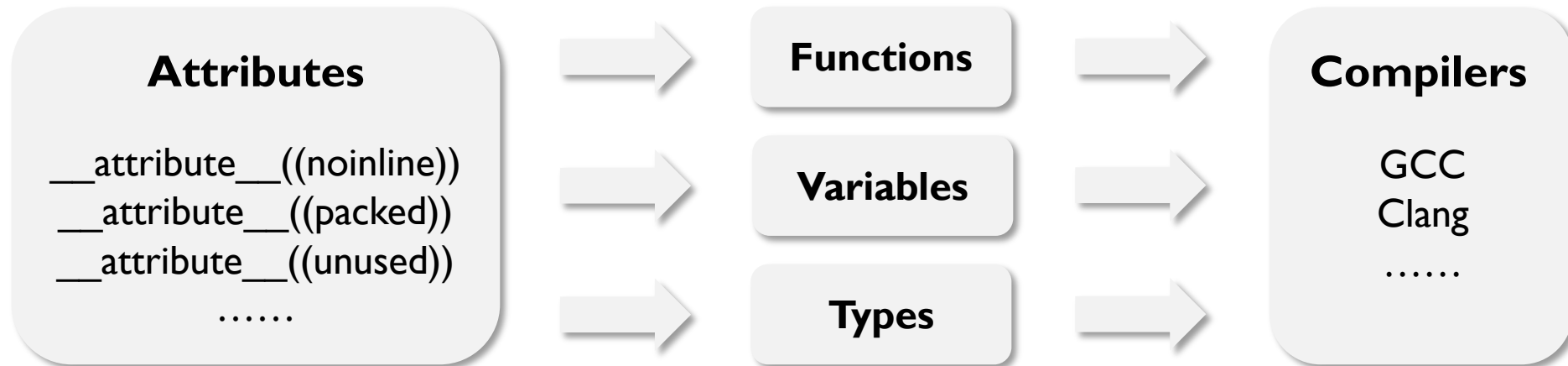
```
gcc -O2 -ftree-loop-distribution source.c  $\xrightarrow{\text{output}}$  0  
-o source && ./source
```



Compiler options are typically applied to the entire program

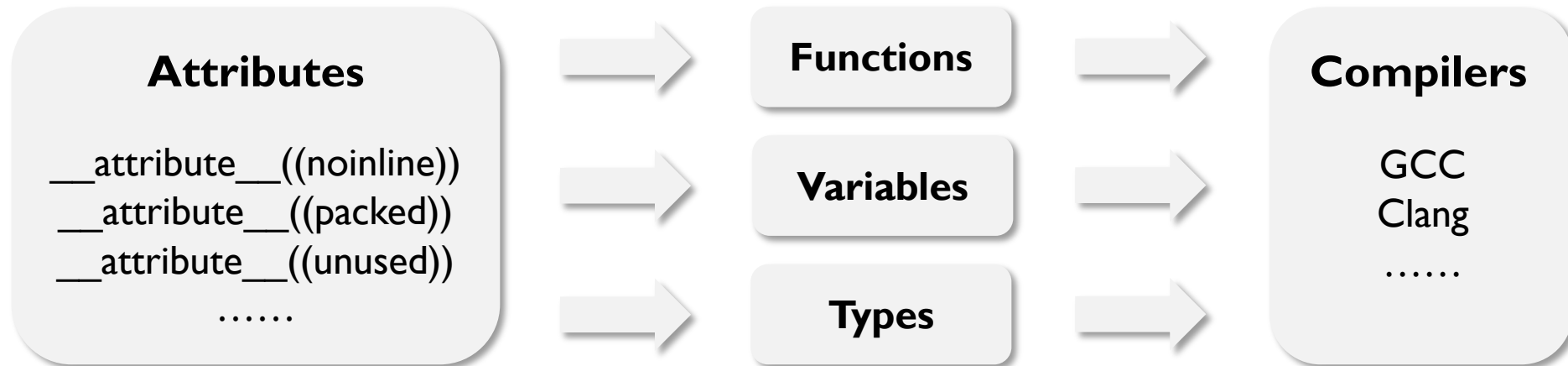
What is attribute?

Attributes give the compiler extra information about specific code elements



What is attribute?

Attributes provide the compiler extra information about specific code elements



Attributes enable fine-grained control over compilation behavior

A motivation example

```
struct S {  
    int i;  
};  
inline struct S bar() {  
    struct S s = { 0 };  
    return s;  
}  
void foo() {  
    bar();  
}
```

Source Program

A motivation example

```
struct S {  
    int i;  
};  
inline struct S bar() {  
    struct S s = { 0 };  
    return s;  
}
```

```
__attribute__((optimize("-ftree-loop-distribution")))
```

```
void foo() {  
    bar();  
}
```

GCC Bug 43234

**GCC crashed when
compiled it with `-O1`**

Another motivation example

```
int **a;
__attribute__((always_inline)) int *foo(void) {
    int b[1];
    (void) b;
}
__attribute__((no_sanitize_address)) static char
bar(void) {
    *a = foo();
}
void baz(void) {
    bar();
}
```

GCC Bug 114956

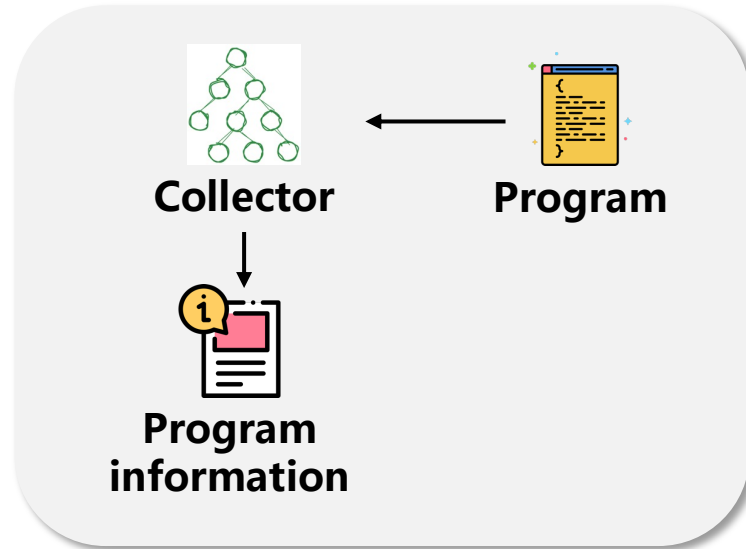
GCC crashed when
compiled it with `-O2`
`-fsanitize=address`

Atlas

Attribute-Guided Compilation Space Exploration

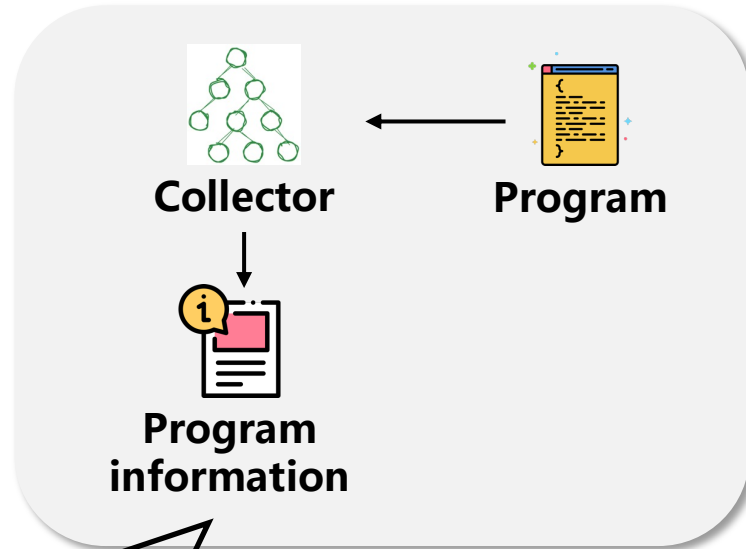
Atlas workflow

Information collection



Atlas workflow

Information collection



Functions info: position, return type...

Variables info: position, variable type...

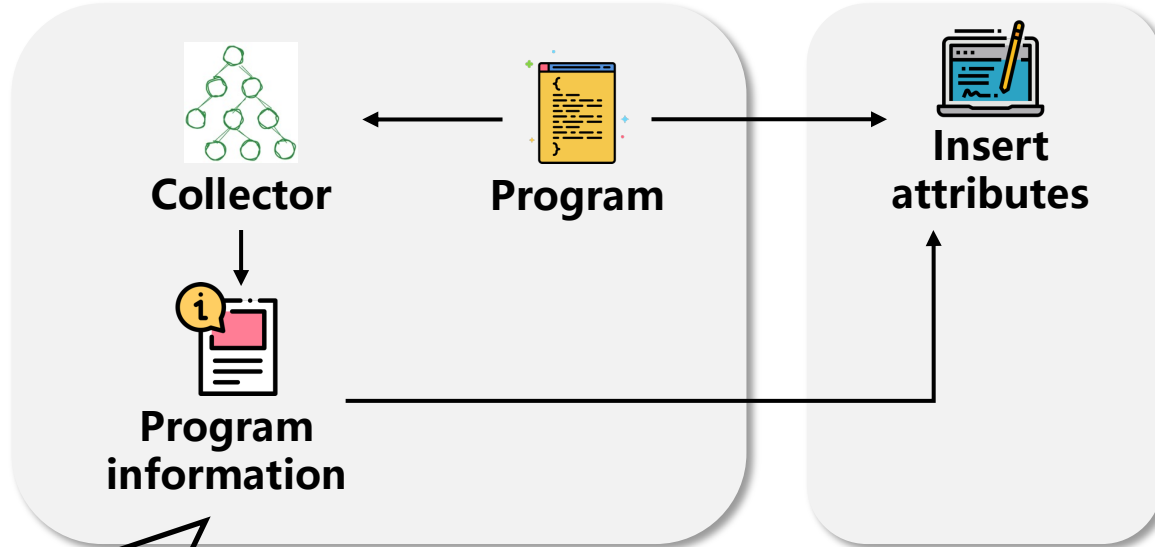
Statements info: position, variables ...

...

Atlas workflow

Information collection

Attributes insertion



Functions info: position, return type...

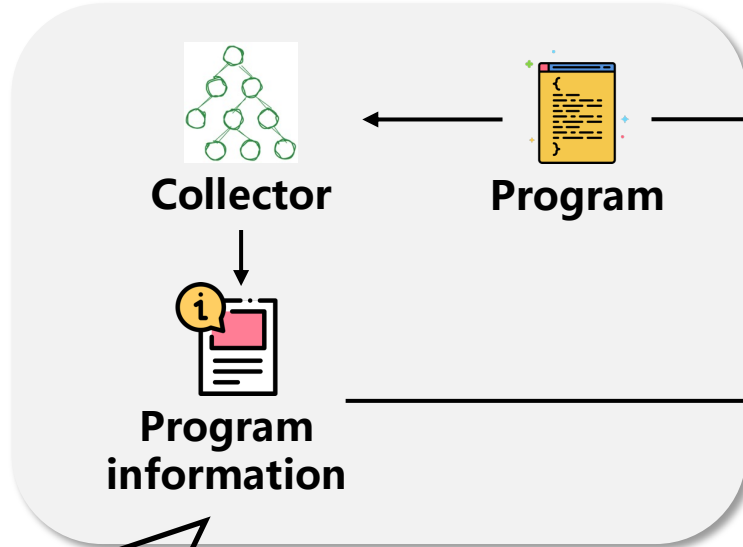
Variables info: position, variable type...

Statements info: position, variables ...

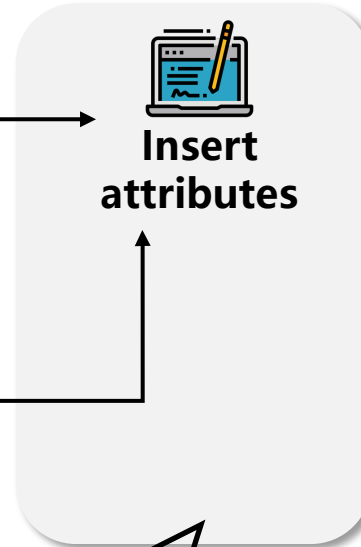
...

Atlas workflow

Information collection



Attributes insertion



Functions info: position, return type...

Variables info: position, variable type...

Statements info: position, variables ...

...

Function attributes

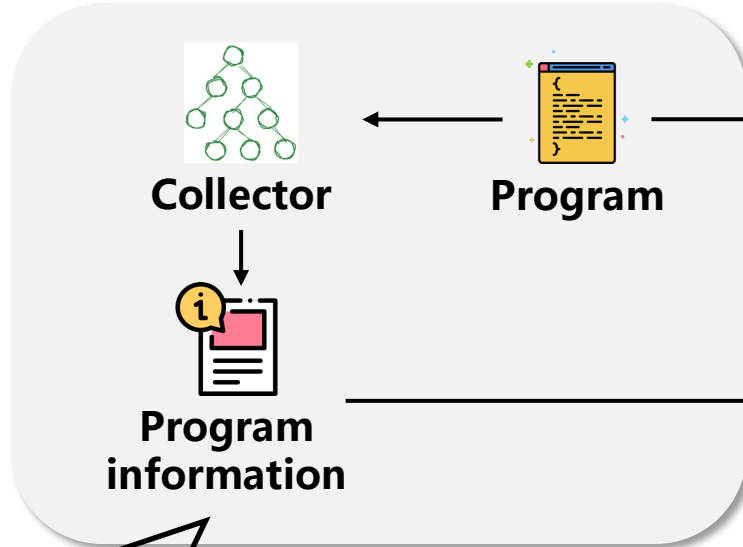
Variable attributes

Statements attributes

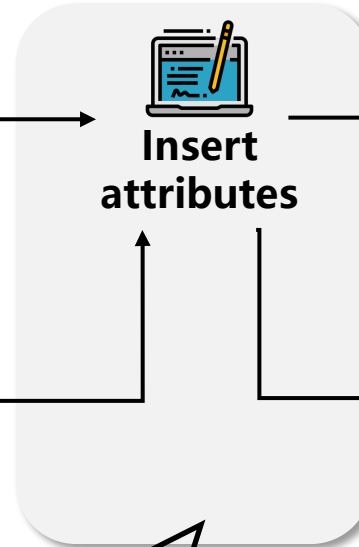
...

Atlas workflow

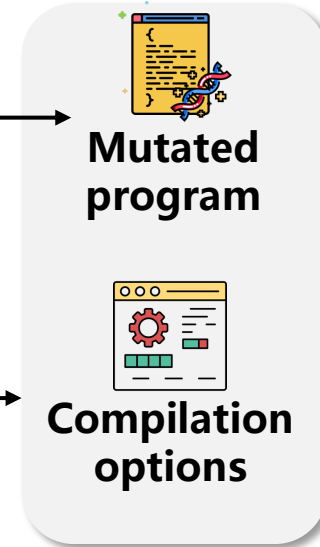
Information collection



Attributes insertion



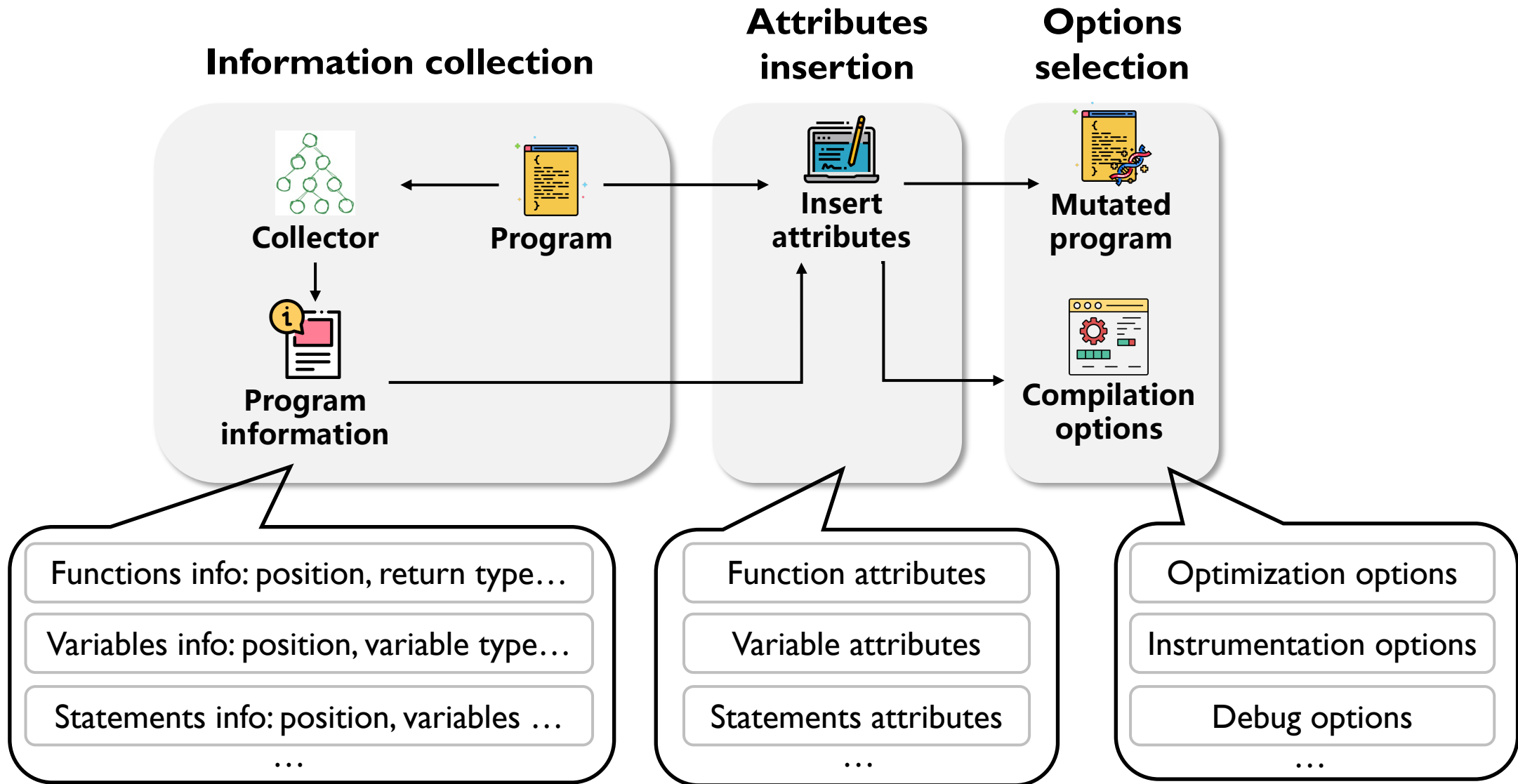
Options selection



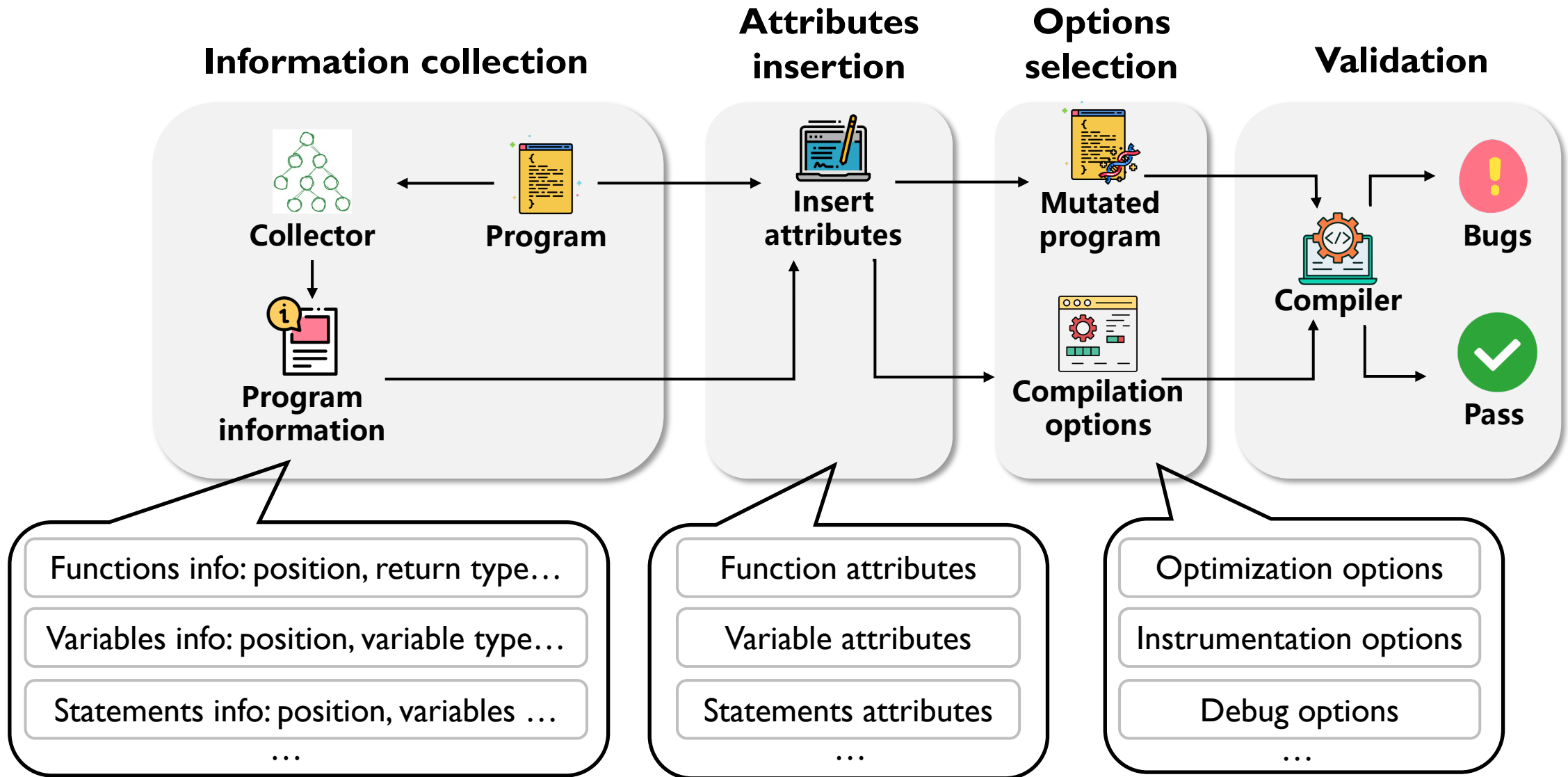
- Functions info: position, return type...
- Variables info: position, variable type...
- Statements info: position, variables ...
- ...

- Function attributes
- Variable attributes
- Statements attributes
- ...

Atlas workflow



Atlas workflow



Atlas generated case I

```
int a;  
int b(int);  
<func> int c() {  
    a = 1;  
    while (a)  
        a = 2;  
    return a;  
}  
<func> int e(void) {  
    int d = c();  
    b(d);  
}
```

Original Program

Atlas generated case 1

```
int a;  
int b(int);  
<func> int c() {  
    a = 1;  
    while (a)  
        a = 2;  
    return a;  
}  
<func> int e(void) {  
    int d = c();  
    b(d);  
}
```

Original Program

```
int a;  
int b(int);  
__attribute__((pure,  
    returns_twice))  
int c() {  
    a = 1;  
    while (a)  
        a = 2;  
    return a;  
}  
int e(void) {  
    int d = c();  
    b(d);  
}
```

Mutated Program

Atlas generated case 1

```
int a;  
int b(int);  
<func> int c() {  
    a = 1;  
    while (a)  
        a = 2;  
    return a;  
}  
<func> int e(void) {  
    int d = c();  
    b(d);  
}
```

Original Program

```
int a;  
int b(int);  
__attribute__((pure,  
    returns_twice))  
int c() {  
    a = 1;  
    while (a)  
        a = 2;  
    return a;  
}  
int e(void) {  
    int d = c();  
    b(d);  
}
```

Mutated Program

GCC crashed when
compiled it with `-O1`
`-fsanitize=address`

Atlas generated case 2

```
<var> int a;  
void b();  
int e(int *c, struct d {  
<var> char an[*c]} *) {  
    sizeof(struct d);  
}  
void f() {  
    if (e(&a, 0))  
        b();  
}
```

Original Program

Atlas generated case 2

```
<var> int a;  
void b();  
int e(int *c, struct d {  
<var> char an[*c]} *) {  
    sizeof(struct d);  
}  
void f() {  
    if (e(&a, 0))  
        b();  
}
```

Original Program

```
int a;  
void b();  
int e(int *c, struct d {  
    __attribute__((  
        vector_size(4)))  
        char an[*c]} *) {  
    sizeof(struct d);  
}  
void f() {  
    if (e(&a, 0))  
        b();  
}
```

Mutated Program

Atlas generated case 2

```
<var> int a;
void b();
int e(int *c, struct d {
<var> char an[*c]} *) {
    sizeof(struct d);
}
void f() {
    if (e(&a, 0))
        b();
}
```

Original Program

```
int a;
void b();
int e(int *c, struct d {
__attribute__((
    vector_size(4)))
    char an[*c]} *) {
    sizeof(struct d);
}
void f() {
    if (e(&a, 0))
        b();
}
```

GCC Bug 117145

GCC crashed when
compiled it with -O2

Atlas generated case 3

```
void a (int &x, int &y) {  
    <stmt>  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Original Program

Atlas generated case 3

```
void a (int &x, int &y) {  
    <stmt>  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Original Program

```
void a (int &x, int &y) {  
    #pragma omp simd  
        linear(uval (x): y+1)  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Mutated Program

Atlas generated case 3

```
void a (int &x, int &y) {  
    <stmt>  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Original Program

```
void a (int &x, int &y) {  
    #pragma omp simd  
    linear(1) (x): y+1  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Mutated Program

Atlas generated case 3

```
void a (int &x, int &y) {  
    <stmt>  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Original Program

```
void a (int &x, int &y) {  
    #pragma omp simd  
        linear(uval (x): y+1)  
    for (i=0; i<10; i++) {  
        x += y + 1;  
    }  
}
```

Clang Bug 108166

Clang crashed when
compiled it with `-O0`

Research Question

- **RQ1: How many compiler bugs have been found by Atlas?**
- **RQ2: How efficient is Atlas in uncovering compiler bugs?**
- **RQ3: Can Atlas improve code coverage?**

RQ1: Effectiveness of Atlas

Reported bugs

	Reported	Confirmed	Fixed
GCC	44	31	13
LLVM	29	27	4
Totl	73	58	17

Distribution of confirmed bugs

Component	GCC	LLVM	Total
Front-End	17	6	23
Middle-End	13	8	21
Optimization	11	5	16
Back-End	3	10	13

✓ **1 GCC P1 bug, 14 GCC P2 bugs**

✓ **Span a wide range of compiler components**

RQ1: Effectiveness of Atlas

Valuable attributes of GCC

Attribute	Type	#	Attribute	Type	#
vector_size	var.	7	alias	func.	1
return_twice	func.	7	no_reorder	func.	1
sanitize	func.	5	noipa	func.	1
simd	func.	4	destructor	func.	1
optimize	func.	4	malloc	func.	1
pure	func.	4	strub	func.	1
const	func.	3	deprecated	func.	1
malloc	func.	2	copy	func.	1
target	func.	2	sentinel	func.	1
aligned	var.	2	nonstring	var.	1
aligned	func.	2	copy	func.	1
flatten	func.	2	always_inline	func.	1
vector_size	func.	2	used	func.	1
constructor	func.	1	interrupt	func.	1

Valuable attributes of LLVM

Attribute	Type	#	Attribute	Type	#
noline	func.	3	speculative	func.	1
vector_size	var.	3	interrupt	func.	1
simd	stat.	2	atomic	func.	1
vectorcall	func.	2	optnone	func.	1
preserve_none	func.	1	nodebug	func.	1
minsize	func.	1	preserve_most	func.	1
sanitize	func.	1	packed	var.	1
opencl	func.	1	regcall	var.	1
loop_vectorize	stmt.	1	target	func.	1
cpu_specific	func.	1	register	func.	1
cold	func.	1	address_space	var.	1
alias	func.	1	constructor	func.	1
noderef	var.	1	-	-	-

Each case contains at least one attribute

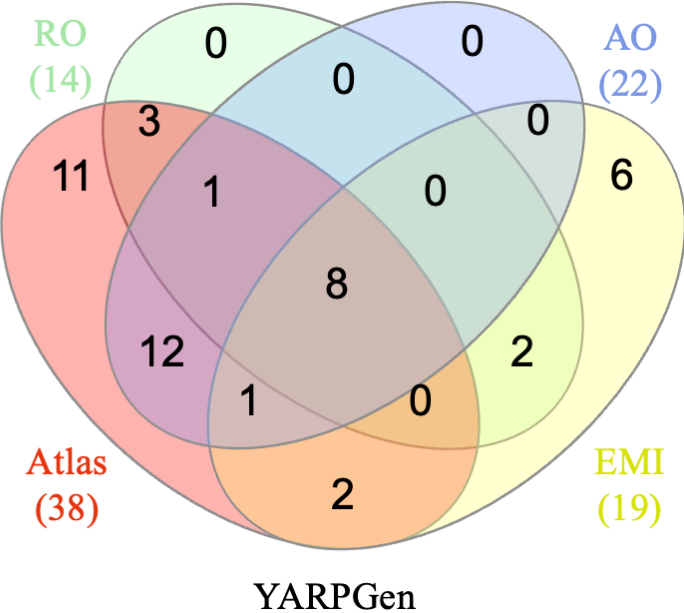
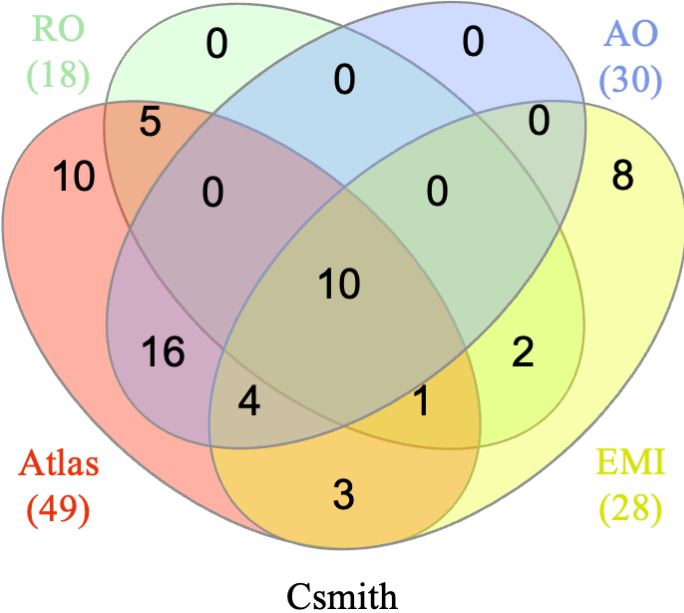
RQ2: Efficiency of Atlas

Detected bugs and generated programs within 10 days

Generators	Techniques	GCC Bugs(#)	LLVM Bugs(#)	Total Bugs(#)	Compilable Programs(#)	Total Programs(#)
Csmith	RO	13	5	18	53,458	53,481
	AO	19	11	30	32,043	47,825
	EMI	17	11	28	48,765	49,279
	ATLAS	32	17	49	30,170	46,416
YARPGen	RO	8	6	14	123,087	123,192
	AO	12	10	22	70,361	108,085
	EMI	10	9	19	101,734	103,628
	ATLAS	25	13	38	62,142	99,439

- ✓ Atlas finds more bugs
- ✓ More than **60%** of program generated by Atlas are compilable

RQ2: Efficiency of Atlas

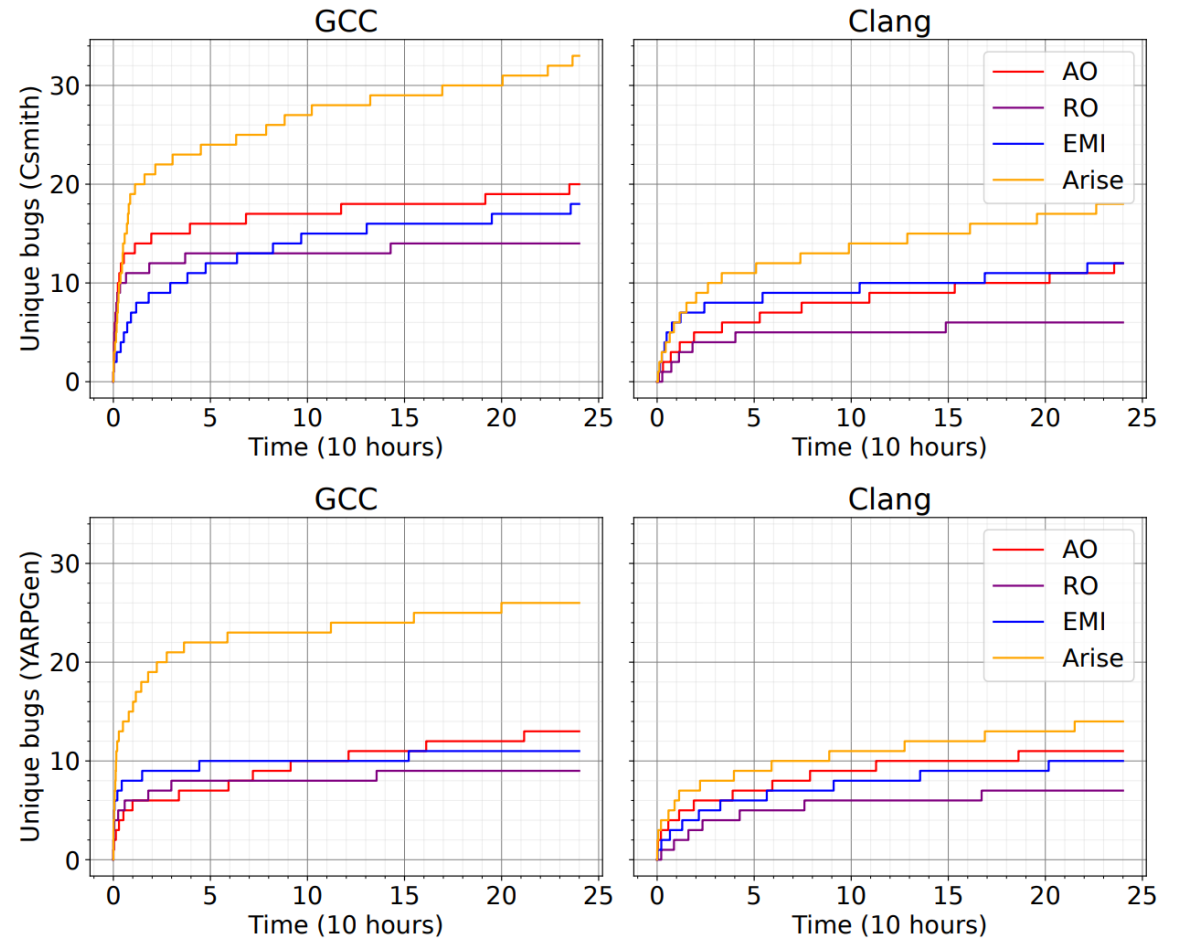


Unique bug found by different approaches

✓ More unique bugs
 ✓ **109.1%** more bugs than other approaches

RQ2: Efficiency of Atlas

- ✓ Atlas continues to uncover bugs over a period of 10 days
- ✓ Atlas maintains a more consistent discovery rate



The earliest time each unique bug was discovered

RQ3: Capability of Atlas

Code coverage

Compiler	Techniques	Line	Function	Branch
GCC	RO	33.5%	35.2%	21.1%
	AO	34.7%(+13,080)	36.9%(+419)	22.0%(+10,697)
	EMI	35.3%(+19,619)	36.8%(+314)	22.4%(15,450)
	ATLAS	36.3%(+30,519)	37.6%(+1,152)	23.2%(+24,958)
LLVM	RO	34.4%	25.6%	20.0%
	AO	35.6%(+20,324)	26.5%(+849)	20.9%(+6,472)
	EMI	35.1%(+12,435)	26.9%(+1,307)	21.3%(+9,592)
	ATLAS	36.8%(+41,806)	27.6%(+1,953)	21.7%(+12,954)

- ✓ Atlas achieves the highest code coverage in all metrics
- ✓ Atlas brings the largest improvements

Conclusion

- **Target:** achieve fine-grained control over compilation for compiler testing
- **Our solution:**
 - ✓ Insert attribute into code target specific elements such as functions, variables, and statements
 - ✓ Combine various options that activate the attributes during compilation
- **Take away:** attributes enable precise control over the compilation strategies applied to individual code elements

Thank You !