Symbolic Verification via Program Transformation

Henrich Lauko

Masaryk University

Brno, Czech Republic

14th May 2018

Outline of the Talk

1. Introspection to my diploma thesis
2. Designing a better approach

Problem Introduction

Problem: how to deal with inputs of a program in the verification?

Problem Introduction

Problem: how to deal with inputs of a program in the verification?

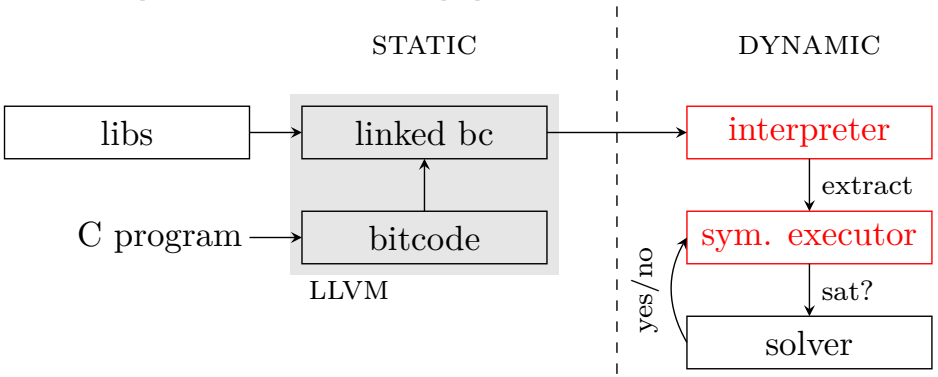
Solution: symbolic representation of data and computation

Problem Introduction

Problem: how to deal with inputs of a program in the verification?

Solution: symbolic representation of data and computation

Interpretation approach:

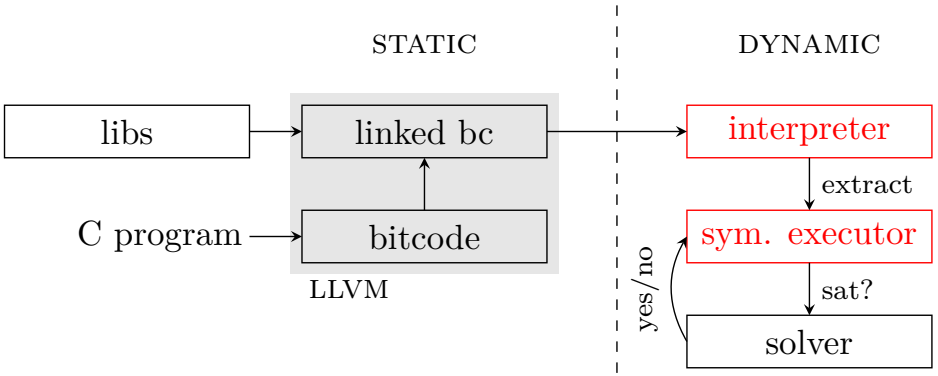


Problem Introduction

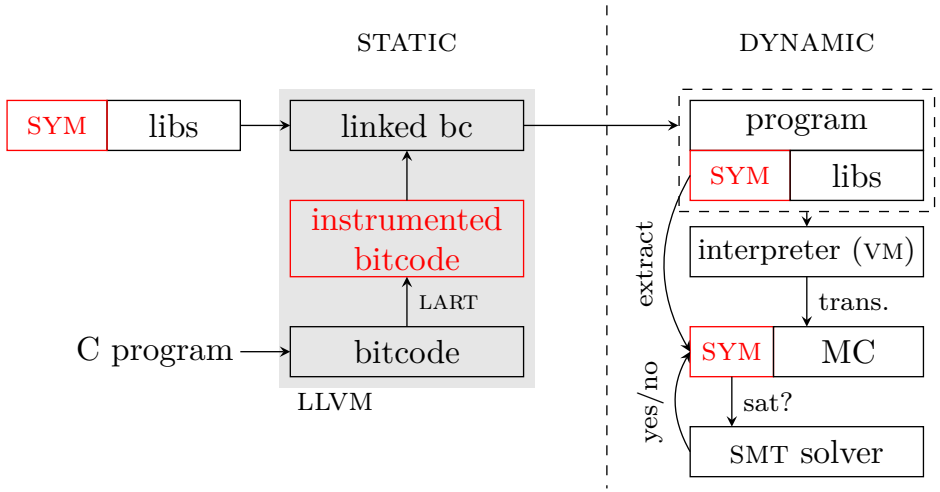
Problem: how to deal with inputs of a program in the verification?

Solution: symbolic representation of data and computation

Interpretation approach:



Compilation approach:

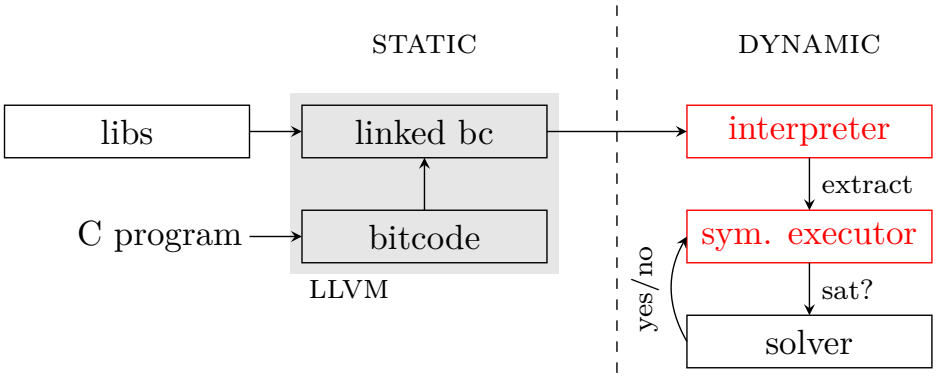


Problem Introduction

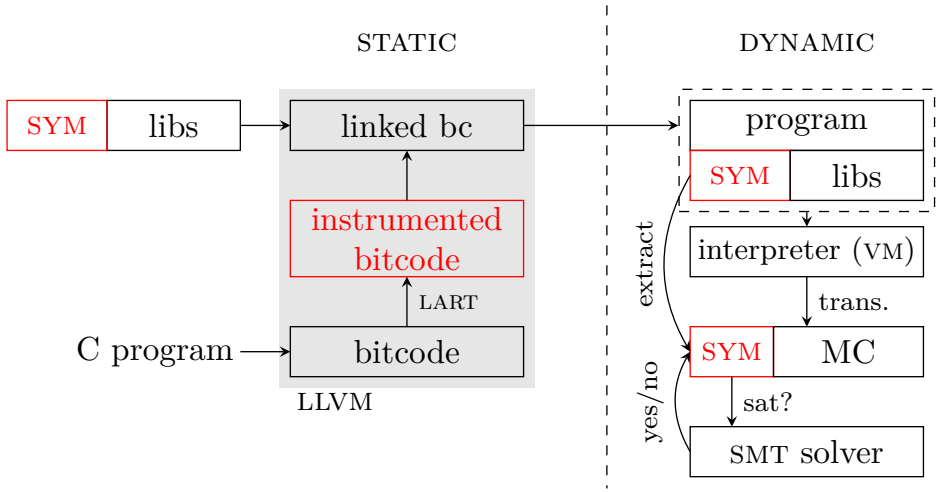
Problem: how to deal with inputs of a program in the verification?

Solution: symbolic representation of data and computation

Interpretation approach:



Compilation approach:



- compilation approach does not complicate an interpreter

Transformation of a Program

Let **x** be marked as symbolic:

```
bool foo(int x) {
  int n  = factorial(7);
  bool y = x + n;
  return y;
}
```

```
SymInt foo(SymInt x) {
  int n = factorial(7);
  SymInt y = sym_add(x, lift(n));
  return y;
}
```

- transform instructions, types, functions
- preserve concrete computation
- lift concrete values

Transformation of a Program

Let **x** be marked as symbolic:

```
bool foo(int x) {
  int n  = factorial(7);
  bool y = x + n;
  return y;
}
```

```
SymInt foo(SymInt x) {
  int n = factorial(7);
  SymInt y = sym_add(x, lift(n));
  return y;
}
```

- transform instructions, types, functions
- preserve concrete computation
- lift concrete values

State After My Diploma Thesis:

- functional prototype
- each value needs to know whether it is concrete or symbolic

Analyzing Drawbacks and Designing a Better Approach

Aggregate Types and Arrays

Problem: types needs to be either concrete or symbolic

```
int arr[ 3 ] = { 1, 2, 3 };
arr[ 1 ] = input();
```

- similar problem with aggregates, recursive structures
- aggregates requires shape analysis

Aggregate Types and Arrays

Problem: types needs to be either concrete or symbolic

```
int arr[ 3 ] = { 1, 2, 3 };
arr[ 1 ] = input();
```

- similar problem with aggregates, recursive structures
- aggregates requires shape analysis

Solution: use discriminated union type

```
UnionInt arr[ 3 ] = { 1, 2, 3 }; //either int or SymInt
arr[ 1 ] = input();
```

Aggregate Types and Arrays

Problem: types needs to be either concrete or symbolic

```
int arr[ 3 ] = { 1, 2, 3 };
arr[ 1 ] = input();
```

- similar problem with aggregates, recursive structures
- aggregates requires shape analysis

Solution: use discriminated union type

```
UnionInt arr[ 3 ] = { 1, 2, 3 }; //either int or SymInt
arr[ 1 ] = input();
```

- use taints to determine in what state is the union (thanks to Adam)

Aggregate Types and Arrays

Problem: types needs to be either concrete or symbolic

```
int arr[ 3 ] = { 1, 2, 3 };
arr[ 1 ] = input();
```

- similar problem with aggregates, recursive structures
- aggregates requires shape analysis

Solution: use discriminated union type

```
UnionInt arr[ 3 ] = { 1, 2, 3 }; //either int or SymInt
arr[ 1 ] = input();
```

- use taints to determine in what state is the union (thanks to Adam)

Improve:

- use bits of concrete value, do not change anything!

Size of Symbolic Values

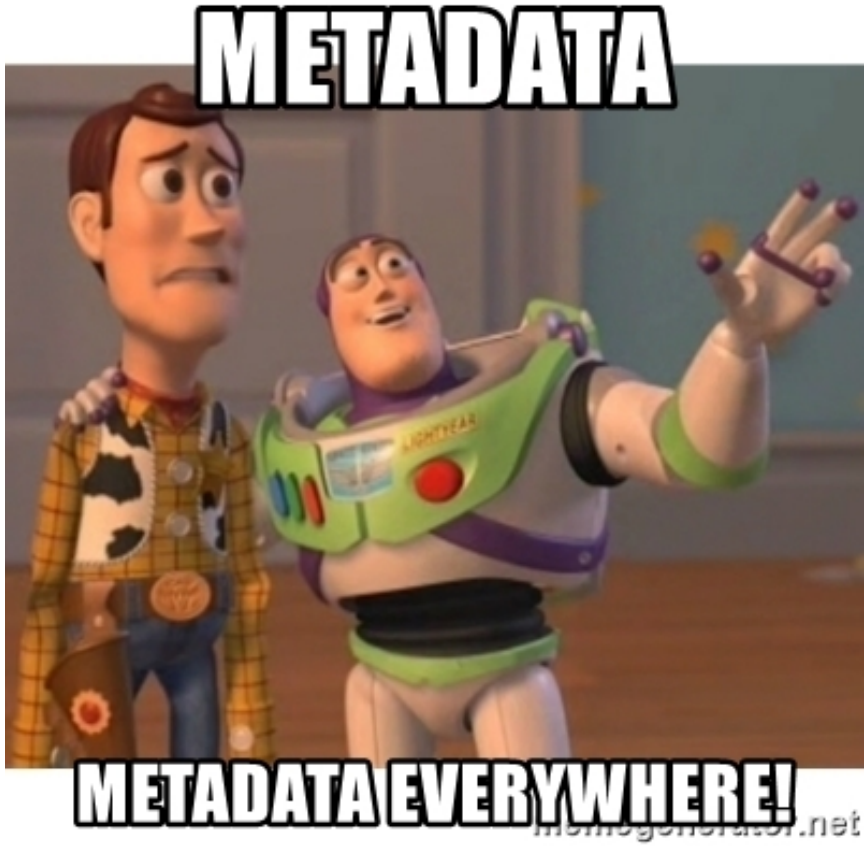
Problem: symbolic value may not fit into the memory of concrete value

Size of Symbolic Values

Problem: symbolic value may not fit into the memory of concrete value

Solution: use metadata to store the symbolic value (thanks to Adam)

- if the value is tainted pick symbolic value from the metadata



Computation with Unions

Problem: instruction needs to decide whether to do concrete or symbolic operation

Computation with Unions

Problem: instruction needs to decide whether to do concrete or symbolic operation

Solution: based on taints decide what operation to do

Before transformation:

```
int y = x + 10;
```

Computation with Unions

Problem: instruction needs to decide whether to do concrete or symbolic operation

Solution: based on taints decide what operation to do

Before transformation:

```
int y = x + 10;
```

After transformation:

```
int y = lifter_add(x, 10);
```

- `y` gets taint if `x` is tainted

Values Lifter

- lifter decides to do either concrete or symbolic operation

```
int lifter_add(int a, int b) {
    if (!is_tainted(a) && !is_tainted(b))
        return a + b;
    else if (!is_tainted(a))
        a = lift(a);
    else if (!is_tainted(b))
        b = lift(b);
    return sym_add(a, b); // tainted value
}
```

Calling of Functions

Problem: we need to create a clone of function for each possible combination of concrete and symbolic arguments

```
int foo(int a, int b, int c);
```

Calling of Functions

Problem: we need to create a clone of function for each possible combination of concrete and symbolic arguments

```
int foo(int a, int b, int c);
```

- may produce exponentially many symbolic duplicates:

```
int foo(SymInt a, int b, int c);
int foo(int a, SymInt b, int c);
int foo(int a, int b, SymInt c);
int foo(SymInt a, SymInt b, int c);
...
```

- resolve return type

Calling of Functions

Problem: we need to create a clone of function for each possible combination of concrete and symbolic arguments

```
int foo(int a, int b, int c);
```

- may produce exponentially many symbolic duplicates:

```
int foo(SymInt a, int b, int c);
int foo(int a, SymInt b, int c);
int foo(int a, int b, SymInt c);
int foo(SymInt a, SymInt b, int c);
...
```

- resolve return type

Solution: solved by union types

Future Work

Future Work

Problem: pointers to symbolic values
Solution: requires alias analysis, insertion of lifters accordingly

Future Work

Problem: pointers to symbolic values
Solution: requires alias analysis, insertion of lifters accordingly

Problem: symbolic pointers
Solution: ???

Future Work

Problem: pointers to symbolic values
Solution: requires alias analysis, insertion of lifters accordingly

Problem: symbolic pointers
Solution: ???

Problem: memory of symbolic size
Solution: ???

Future Work

Problem: pointers to symbolic values
Solution: requires alias analysis, insertion of lifters accordingly

Problem: symbolic pointers
Solution: ???

Problem: memory of symbolic size
Solution: ???

Thanks for attention