Enabling Multi- and Cross-Language Verification with LLVM

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

SOFTWARE ANALYSIS RESEARCH LABORATORY

Jack J. Garzella, Marek Baranowski, Shaobo He, Zvonimir Rakamarić

Problem

- Many verifiers are made to support only C programming language
- Nowadays programs are written in many different languages and their combinations
- New languages are being invented (e.g., Rust, Swift) to improve ease of development, security, and/or performance
- Verification tooling for these new languages does not exist

Solution

- Leverage LLVM IR to lower development cost of supporting multiple input languages
- SMACK software verifier already uses LLVM IR
- Explore feasibility of such approach

Methodology

1. Include SMACK headers
   - LLVM IR models
   - Verification primitives
2. Compile to LLVM IR
   - Emit IR
   - Identify top function
3. Model missing features
   - Language/IR features
   - Standard libraries

Microbenchmarks

```
int cap(int x) {
  int y = x;
  if (10 < x) { y = 10; }
  return y;
}

int main(void) {
  assert(cap(2) == 2);
  assert(cap(15) == 10);
  int x = nondet_int();
  assert(cap(x) <= 10);
}
```

```
fn cap(x: usize) -> usize {
  let mut y = x;
  if 10 < x {
    y = 10;
  }
  return y;
}


fn main() {
  let two = cap(2);
  let ten = cap(15);
  assert!(two == 2);
  assert!(ten == 10);
  let x = nondet_int();
  assert!(x <= 10);
}
```

```
func cap(_ x: Int) -> Int {
  var y = x
  if 10 < x {
    y = 10
  }
  return y
}

assert(cap(2) == 2)
assert(cap(15) == 10)
let x = Int(nondet_int())
assert(cap(x) <= 10)
```

```
pure function cap(x)
  integer, intent(in) :: x
  integer :: cap, y
  y = x
  if (10 < y) then
    y = 10
  end if
  cap = y
end function

program main
  integer :: cap, x
  call assert(cap(2) == 2)
  call assert(cap(15) == 10)
  x = nondet_int()
  call assert(cap(x) <= 10)
end program main
```

Conclusions

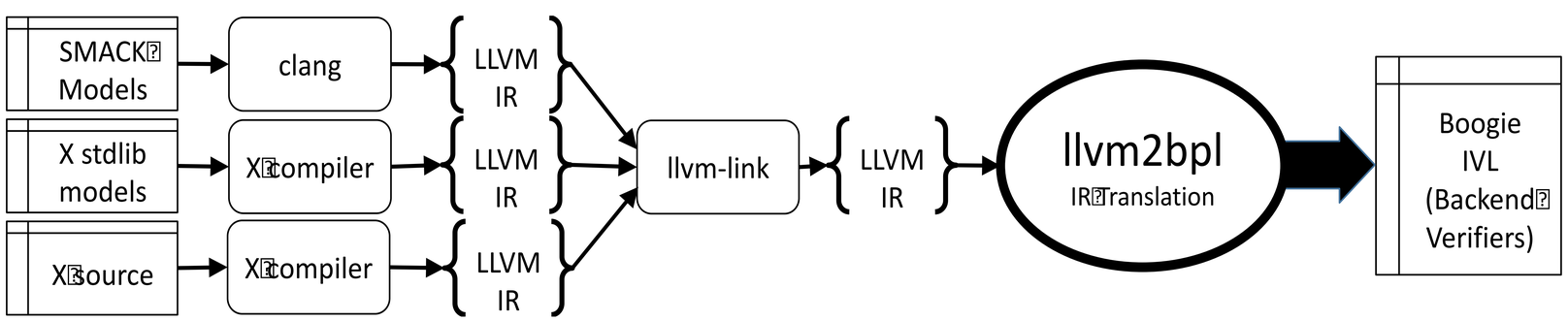
Strengths

- Modular architecture that uses LLVM avoids the need for front-end development
- New language can be added with modest amount of development effort

Challenges

- Modeling of standard libraries and large runtimes is challenging and time consuming

SMACK Toolflow Diagram



Microbenchmark Results

Benchmark	C	C++	Obj-C	Rust	Fortran	D	Swift	Kotlin
basic								
compute								
function								
forloop								
fib								
compound								
array								
pointer							N/A	N/A
inout								N/A
method	N/A				N/A			
dynamic	N/A				N/A			

This work was supported by the University of Utah Office of Undergraduate Research and National Science Foundation