haha viper go hsssssss*

*working title: Automatic Predicate Inlining for Viper

Jonathan Chan, Braxton Hall, Alison Li, James Yoo

Overview

- (a very brief) Introduction to Viper
- Predicates
- Research Question
- Automatic Predicate Inlining
- The Story So Far…
- Questions?

Viper

Verification **I**nfrastructure for **Pe**rmission-based **R**easoning

- architecture for verification tools/prototypes
- **intermediate verification language**
- common use cases
  - sequential/concurrent programs
  - mutable state
  - separation logic (think Rust)
- provided verifiers
  - Boogie — verification condition generation
  - Z3 — symbolic execution
- compiles down to Z3

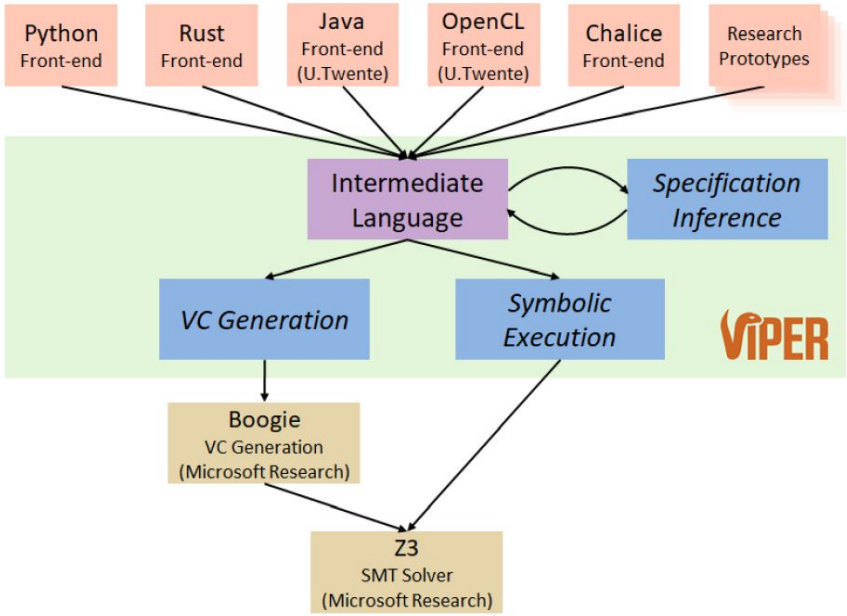


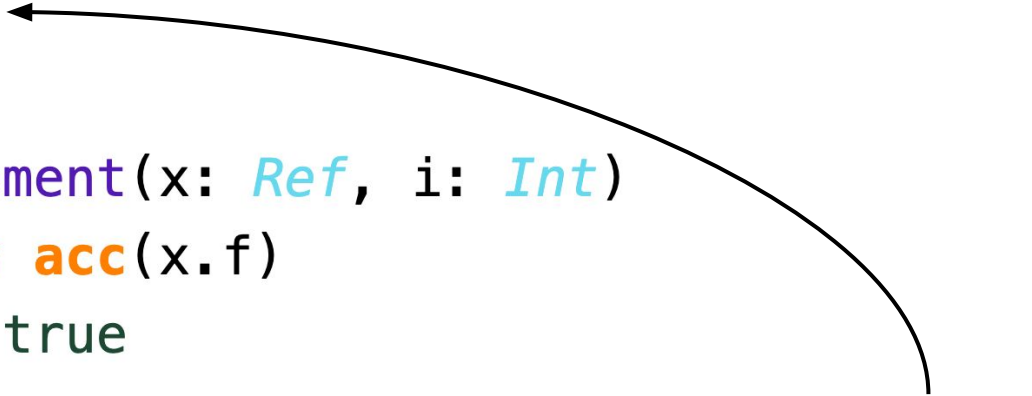
image from Programming Methodology Group | ETH Zurich

Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```

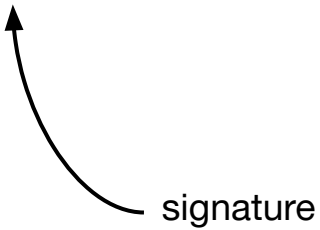
Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```



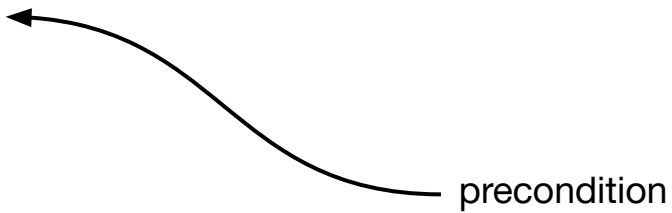
Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```



Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```



Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```

Viper in Action

```
field f: Int

method increment(x: Ref, i: Int)
    requires acc(x.f)
    ensures true
{
    x.f := x.f + i
}
```

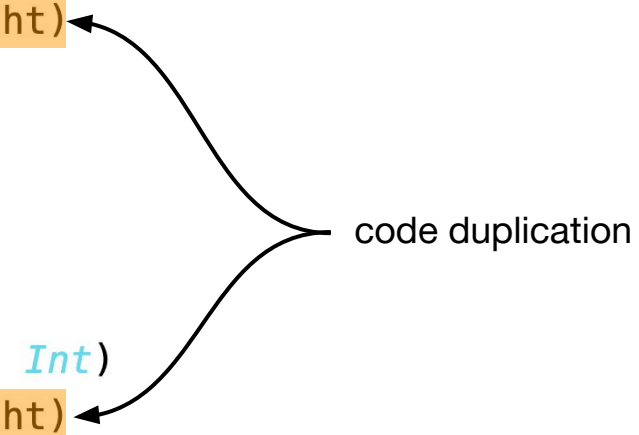
Predicates

Working with Tuples

```
method makeTuple(this: Ref, a: Int, b: Int)
    requires acc(this.left) && acc(this.right)
{
    this.left := a
    this.right := b
}

method incrementTuple(this: Ref, i: Int, j: Int)
    requires acc(this.left) && acc(this.right)
{
    this.left := this.left + i
    this.right := this.right + j
}
```

Working with Tuples

```
method makeTuple(this: Ref, a: Int, b: Int)
    requires acc(this.left) && acc(this.right)
{
    this.left := a
    this.right := b
}

method incrementTuple(this: Ref, i: Int, j: Int)
    requires acc(this.left) && acc(this.right)
{
    this.left := this.left + i
    this.right := this.right + j
}
```



Predicates

```
predicate allowAccess(this: Ref)
{
    acc(this.left) && acc(this.right)
}
```

```
method incrementTuple(this: Ref, i: Int, j: Int)
    requires allowAccess(this)
{
    unfold allowAccess(this)
    this.left := this.left + i
    this.right := this.right + j
    fold allowAccess(this)
}
```

Recursive Predicates: Lists

```
predicate list(this: Ref) {
  acc(this.elem) && acc(this.next) &&
  (this.next ≠ null ⟹ list(this.next))
}
```

```
method append(this: Ref, e: Int)
  requires list(this)
  ensures  list(this)
{
  unfold list(this);
  if (this.next == null) {…
  fold list(this);
}
```

Research Question

Problem(s)

- A lot of Viper code is automatically generated
  - can lead to redundancies that a human developer would detect and remove
- An unfold is an operation that changes the program state, replacing the predicate resource with the assertions specified by its body. [1]
  - this is a lot of work performed at ***runtime***

[1] ETH Zurich

How do inlined predicates impact the performance of Viper?

Automatic Predicate Inlining

Automatic Predicate Inlining

```
inline predicate allowAccess(this: Ref)
{
    acc(this.left) && acc(this.right)
}
```

predicates quantified with the **`inline`** keyword will be automatically expanded when they're used

```
inline predicate allowAccess(this: Ref)
{
    acc(this.left) && acc(this.right)
}

method makeTuple(this: Ref, a: Int, b: Int)
    requires allowAccess(this)
{
    unfold allowAccess(this)
    this.left := a
    this.right := b
    fold allowAccess(this)
}
```

rewrite methods

```
method makeTuple(this: Ref, a: Int, b: Int)
    requires acc(this.left) && acc(this.right)
{
    this.left := a
    this.right := b
}
```

Dealing with (Mutually) Recursive Predicates

```
predicate list(this: Ref) {
  acc(this.elem) && acc(this.next) &&
  (this.next ≠ null ⟹ list(this.next))
}
```

infinite inline!

```
list(this)

acc(this.elem) && acc(this.next) &&
(this.next ≠ null ⟹ list(this.next))

acc(this.elem) && acc(this.next) &&
(this.next ≠ null ⟹
  acc(this.elem) && acc(this.next) &&
  (this.next ≠ null ⟹ list(this.next)))

acc(this.elem) && acc(this.next) &&
(this.next ≠ null ⟹
  acc(this.elem) && acc(this.next) &&
  (this.next ≠ null ⟹
    acc(this.elem) && acc(this.next) &&
```

options:

1. ~~trust the programmer~~
2. you may inline… once ☝️
3. detect cycles & error out

programmer is sus

Eventually: *Automatic* Automatic Predicate Inlining

- Insert the **`inline`** keyword wherever possible
  - requires the detection of recursive/mutually-recursive cycles

```
predicate allowAccess(this: Ref)
{
    acc(this.left) && acc(this.right)
}

predicate list(this: Ref)
{
    acc(this.elem) && acc(this.next) &&
        (this.next != null => list(this.next))
}
```


```
inline predicate allowAccess(this: Ref)
{
    acc(this.left) && acc(this.right)
}

predicate list(this: Ref)
{
    acc(this.elem) && acc(this.next) &&
        (this.next != null => list(this.next))
}
```

Validation (Future Goals)

- Industrial Benchmarks
  - Provided by Prof. Alex Summers (SPL)
- Run two different versions of Viper
  - one with inlining
  - one without

The Story So Far...

Milestones

Decide on a language construct/keyword for inlining ✔️

Implement PoC that inlines all simple predicates ✔️

Recursive cycle detection ✔️

Extend Viper parser with the `inline` keyword

Extend PoC to inline only predicates marked with `inline`

Benchmark performance of Viper programs with inlined predicates

Merge with upstream Viper project 🔜

Questions?