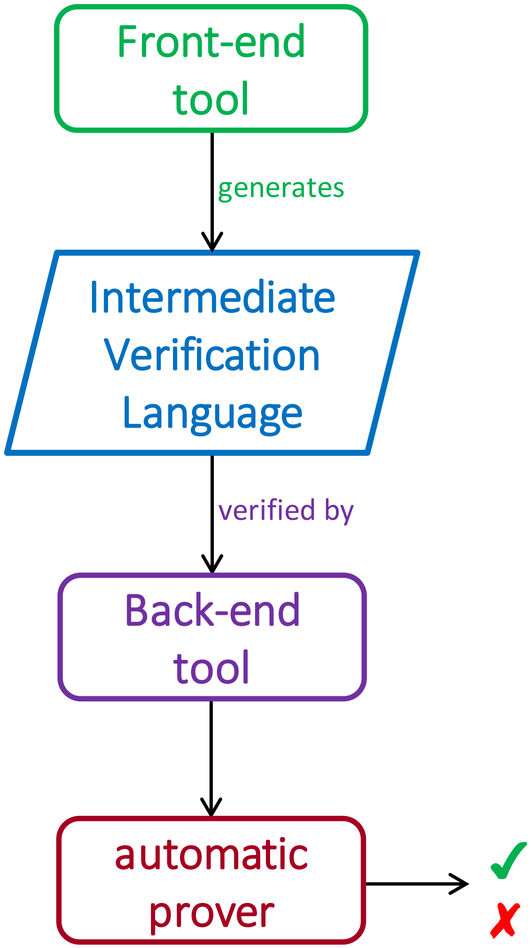
Viper
A Verification Infrastructure for Permission-Based Reasoning

Alex Summers, ETH Zurich

Joint work with Uri Juhasz, Ioannis Kassios,
Peter Müller, Milos Novacek, Malte Schwerhoff
(and many students)

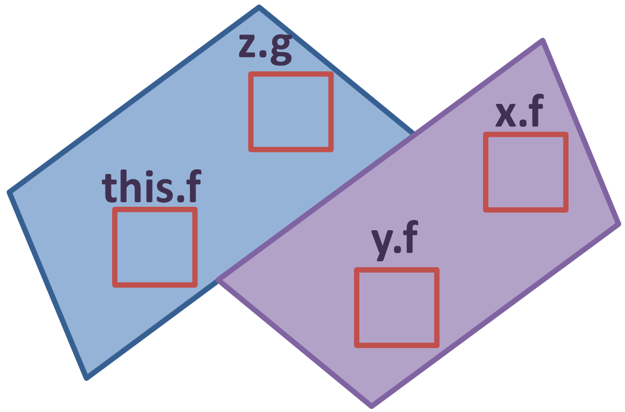
16th July 2015, Imperial Concurrency Workshop

Verification via Automatic Provers



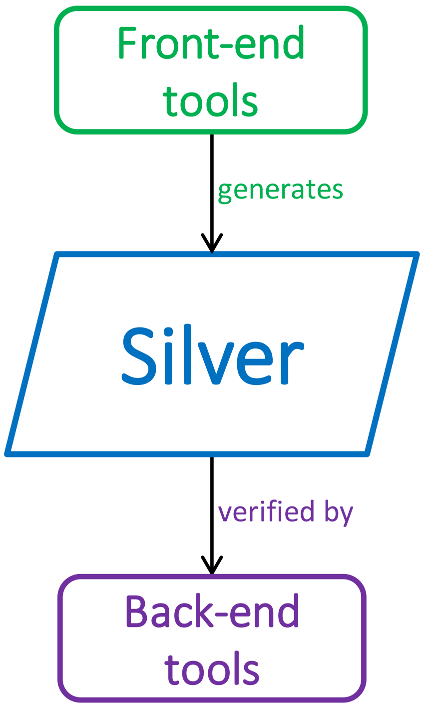
- Last 10 years: rapid progress in automatic tools for first-order logics (SMT solvers, provers)
- Intermediate Verification Languages: e.g. Boogie and Why
- Provide common infrastructures for building program verifiers
- Many success stories and tools
  - Microsoft Hypervisor (VCC)
  - Device-drivers (Corral)
  - .. and many more, e.g., Why3, GPUVerify, Spec#, Dafny, Vericool, Krakatoa, etc....

Permission-Based Reasoning



- Separation Logic (and others): custom logics for heap reasoning
- control of ownership/sharing of *partial heaps* (heap fragments)
- First-order prover technology *difficult* to directly exploit
  - Custom verification engines (usually symbolic execution)
  - Lots of work to implement
  - Hard to reuse for new work
  - … fewer tools available ☹

The Viper Project



- We have designed **Silver**: a new intermediate verification language
  - Reusable *native support* for permission-based heap reasoning
  - Few, expressive constructs
- The tool infrastructure is called Viper
  - Includes back-ends (two verifiers)
- Some front-end tools also available:
  - Proof-of-concept translators for
    - *Chalice*, *Scala* (fragment), etc..
    - used for various other projects

Basic Assertion Language

- Based on *Implicit Dynamic Frames* [Smans et al. '09]
- Permission assertions: *accessibility predicates* acc(e.f)
  - exclusive: similar to $e.f \mapsto \_$ in separation logics
- Expressions e may depend directly on the heap
  - e.g. acc(x.f) && x.f > 0
- *Fractional permissions* [Boyland'03], e.g. acc(x.f, ½)
  - allow reading (and framing), not writing
- Conjunction && is multiplicative for permissions
  - e.g. acc(x.f, ½) && acc(x.f, ½) ≡ acc(x.f, 1)

Silver primitives: Inhale and Exhale

- A statement **`inhale A`** means:
  - all permissions required by are **`A`** gained
  - all logical constraints (e.g. x.f > 0) are *assumed*
- A statement **`exhale A`** means:
  - check, and remove all permissions required by **`A`**
  - all logical constraints (e.g. x.f > 0) are *asserted*
  - any locations to which all permissions is lost are implicitly *havoced* (their values are no-longer known)
- Can be seen as the *permission-aware analogues* of assume/assert statements used in first-order verification
  - used to model ownership transfer of partial states
  - cf. “produce” and ”consume” in symbolic execution

Example : Encoding Locks (CSL-style)

```
class C {
  int[ ] data; int count = 0;

  monitor invariant this.data↦_ * this.count↦_


  void Foo( ) {
    acquire this;
    int i = data.length;
    while( 0 < i )
      invariant this.data↦_ * this.count↦_
      invariant holds( this );
    { ... ; i = i - 1; }
    count = count + 1; release this;
  }
}
```

A few powerful Viper features….

- *Paired assertions* **`[A,B]`**
  - **`A`** used when inhaled, **`B`** used when exhaled
  - mismatches: external justification / proof obligations
- *Quantification over local state* **`forallrefs[f] x ::`**
  - non-standard for separation logics (but handy)

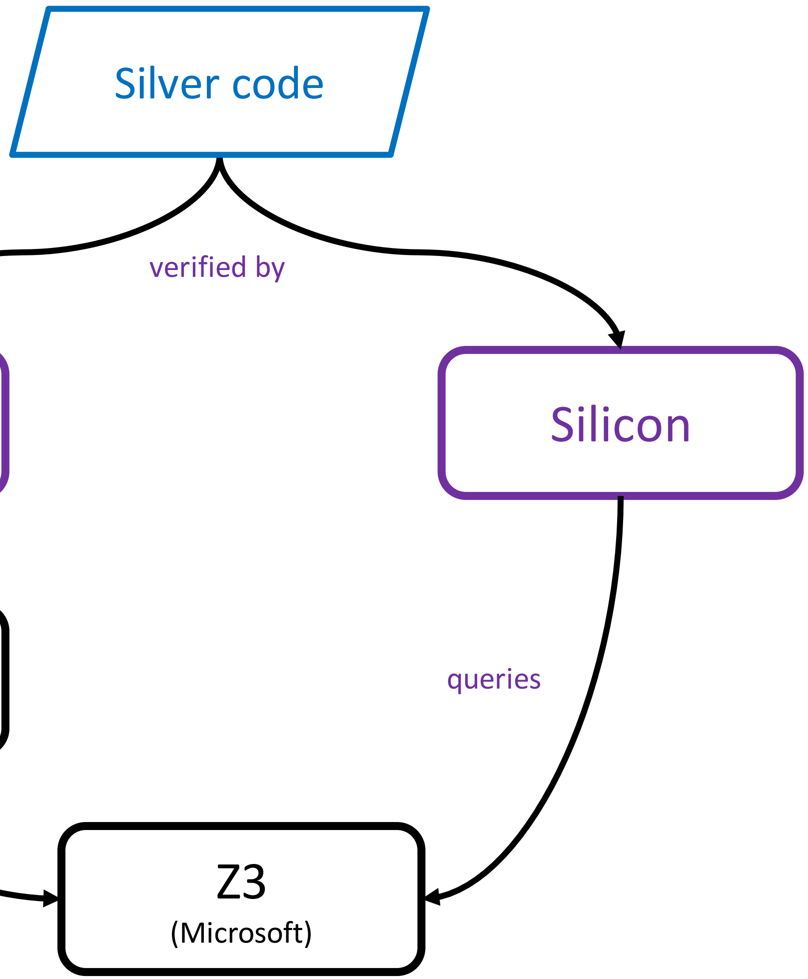
Example : Two-state invariants

```
class C {
  int[ ] data; int count = 0;

  monitor invariant this.data↦_ * this.count↦_
                    && this.count > old(this.count)

  void Foo( ) {
    acquire this;
    int i = data.length;
    while( 0 < i )
      invariant this.data↦_ * this.count↦_
      invariant holds( this );
    { ... ; i = i - 1; }
    count = count + 1; release this;
  }
}
```

A few powerful Viper features....

- *Paired assertions* **`[A,B]`**
  - **`A`** used when inhaled, **`B`** used when exhaled
  - mismatches: external justification / proof obligations
- *Quantification over local state* **`forallrefs[f] x ::`**
  - non-standard for separation logics
- *State snapshots, labelled "old" expressions*
- *Custom predicates, heap-dependent functions* [ECOOP'13]
  - fold/unfold for predicates, functions mostly automatic
- *Constrainable permissions* [VMCAI'13, FTfJP'14]
  - Alternative to fractional permissions (angelic amounts)
- *"Magic wand" support* [ECOOP'15]
  - Powerful connective from separation logic
- *Custom domains, sets and sequences, quantifiers*

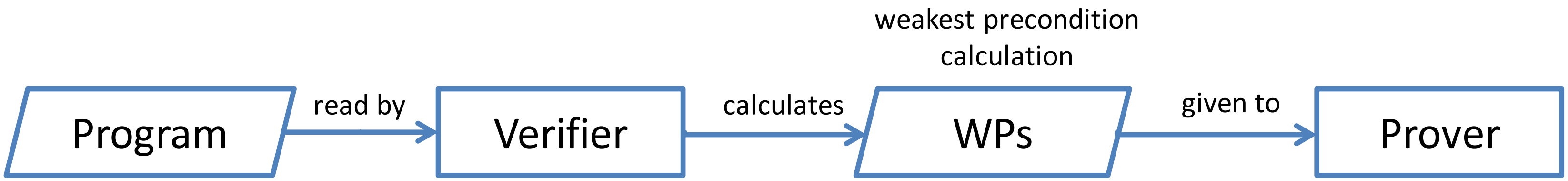
Verification of Silver Code (back-ends)



Verification Condition Generation

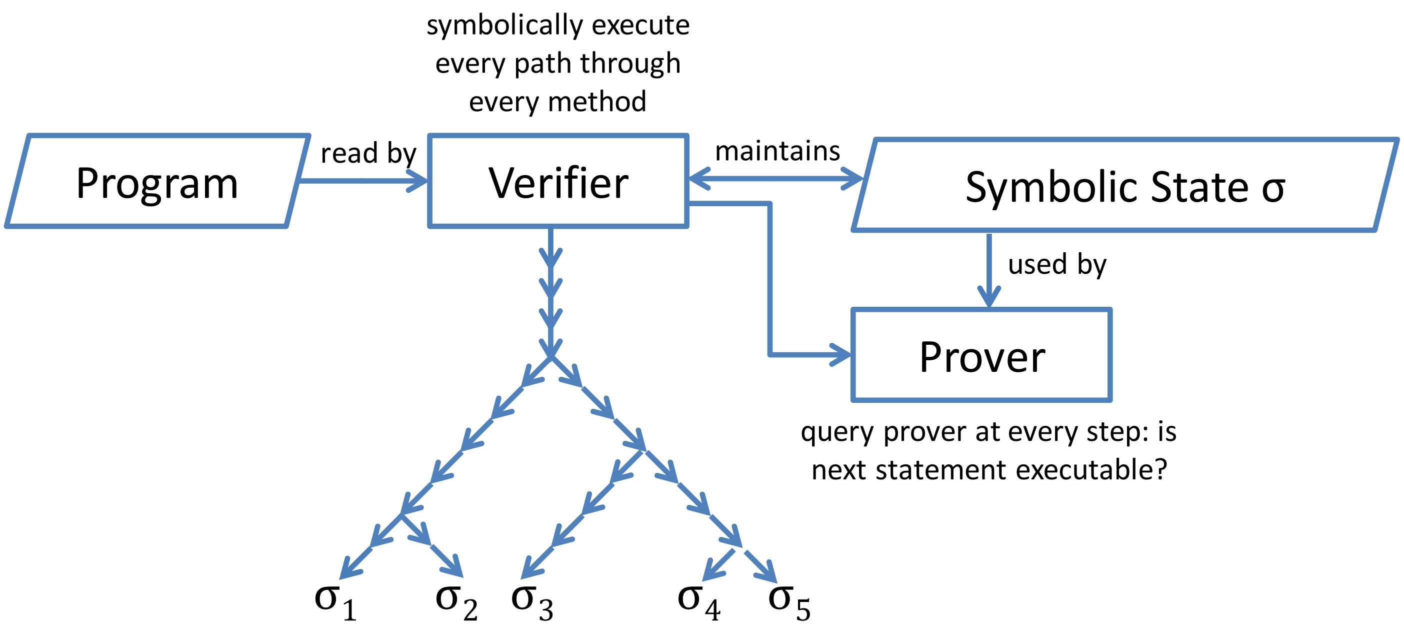
Symbolic Execution

Verification of Silver Code (back-ends)



VCG

Query prover once with full information (encode heap)



SE

Query prover often with limited information (no heap)

Tool Availability and Future Work

- Core tools released (open-source) in September 2014:

  `http://www.pm.inf.ethz.ch/research/viper.html`

  `https://bitbucket.org/viperproject`

  - we have (public!) issue trackers for known problems
  - Some advanced features are in the pipeline (but ask)

- Building / supporting new tools by translations into Silver
  - SL, dynamic frames, invariants, rely-guarantee, types
  - More-advanced program logics? Weak memory? …
  - Also interested in work we *cannot* encode (yet …)

- Make tools to implement your cool research with Viper ☺
  - coalesces much formal and practical past research
  - users can focus on the aspects relevant to their work

Any questions?