Viper
A Verification Infrastructure for Permission-based Reasoning

Peter Müller, ETH Zurich

Joint work with
Pietro Ferrara, Uri Juhasz, Ioannis Kassios, Milos Novacek, Malte Schwerhoff, and Alex Summers
Automatic Program Verification

- Most automatic deductive verifiers use solvers for first order logic (Z3, CVC4)
- Verification conditions are computed via translation to intermediate verification language (Boogie, Why)
- Many success stories: Corral, Dafny, Frama-C, Spec#, VCC
Verifiers for Permission Logics

- Separation Logic (and other permission logics) use custom logics to reason about heap-manipulating programs
- Custom verification engines (jStar, Smallfoot, VeriFast)
Viper Infrastructure

- Frontend
- Intermediate Language
- Specification Inference
- Boogie Encoding
- Symbolic Execution
- Boogie VC Generation (Microsoft Research)
- Z3 SMT solver (Microsoft Research)
Permissions

- Permissions denoted by accessibility predicates

- Expressions e may depend on the heap

- Support for fractional permissions

- Conjunction is multiplicative (as in in separation logic)

\[
\text{acc}(e.f)
\]

\[
\text{acc}(x.f) \land x.f > 0
\]

\[
\text{acc}(x.f, \frac{1}{2})
\]

\[
\text{acc}(x.f, \frac{1}{2}) \land \text{acc}(x.f, \frac{1}{2})
\]
Inhale and Exhale

- **inhale** A means:
  - all permissions required by A are obtained
  - all logical constraints (e.g., x.f > 0) are assumed

- **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)

- Analogues of assume and assert
Example: Modeling Locks

class C {
    @GuardedBy("this") int[] data;

    void Foo() {
        acquire this;
        int i = data.length;
        while (0 < i) {
            ...;
            i = i - 1;
        }
        release this;
    }
}
domain Array {
  function loc( a: Array, i: Int ): Ref
  function length( a: Array ): Int

  axiom all_diff {
    forall a1: Array, a2: Array, i: Int, j: Int ::
    ( a1 != a2 || i != j ) ==> loc( a1, i ) != loc( a2, j )
  }

  axiom length_nonneg {
    forall a: Array :: length( a ) >= 0
  }
}
Symbolic Read Permissions

```c
int eval( State s )
requires acc( s.map, read ) && s.map != null
ensures acc( s.map, read )
```

```c
int eval( State s ) {
  leftTk := fork left.eval( s );
  rightTk := fork right.eval( s );
  return ( join leftTk ) + ( join rightTk );
}
```
class List {
    int value;
    List next;

    int len() {
        if (next == null) return 1;
        else return 1 + next.len();
    }

    int itemAt(int i) {
        if (i == 0) return value;
        else return next.itemAt(i - 1);
    }
}
Inhale-Exhale Pairs

- Proof principles
  - $\forall n \in \mathbb{N} \cdot n > 0 \land P(n-1) \Rightarrow P(n)$
  - $\forall n \in \mathbb{N} \cdot P(n)$

- Properties justified elsewhere

- Context-dependent proof obligations
  - type( this ) <: T

- $\forallallrefs \text{[ holds ]} x :: \text{false}$
  - true
Viper – Frontends

- **Chalice Frontend**
- **Scala Frontend**
- **Java Frontend**
  - (University of Twente)
- **OpenCL Frontend**
  - (University of Twente)

Intermediate Language

Specification Inference

Boogie Encoding

Symbolic Execution

Boogie

VC Generation
  - (Microsoft Research)

Z3

SMT solver
  - (Microsoft Research)

Peter Müller – ETH Zurich, October 02, 2015
Conclusion

- Viper is useful to
  - Develop verifiers based on permission-logics
  - Prototype new verification techniques
  - Experiment with and integrate different back-ends

- Current work
  - Increase expressiveness of intermediate language
  - Encode more languages and program logics
  - Improve inference

- Try Viper online at viper.ethz.ch
Intermediate Language

- Top-level declarations
  - Fields
  - Methods
  - Heap-dependent functions
  - Predicates
  - Domains

- Types
  - Int, Bool, Ref, Perm
  - Set[T], Seq[T]
  - Types declared in domains

- Statements
  - Assignments, calls, conditionals, loops
  - inhale, exhale
  - fold, unfold

- Assertions
  - Permissions
  - Predicates
  - Magic wands
  - Quantifiers
Example: Leak Check

class C {
    @GuardedBy("this") int[] data;

    void Foo() {
        acquire this;
        int i = data.length;
        while (0 < i) {
            ...
            i = i - 1;
        }
        release this;
    }
}

Peter Müller – ETH Zurich, October 02, 2015
Example: Two-State Invariants

class C {
    @GuardedBy("this") int[] data;
    @GuardedBy("this") int count;

    monitor invariant count == old(count) + 1;

    void Foo() {
        acquire this;
        count++; // Incorrect
        int i = data.length;
        while (0 < i) { ...; i = i - 1; }
        release this;
    }
}