Lightweight Support for Magic Wands in an Automatic Verifier

Malte Schwerhoff and Alexander J. Summers

10th July 2015, ECOOP, Prague
Frame Problem

Modular, static verification of imperative programs

Frame problem: Which memory locations change?

Automated verification: Pre-/Postconditions, invariants, ghost code

Well-known approach: Permissions (≈ Separation Logic)
Permission Transfer

caller  callee
Permission Transfer

caller
callee
Permission Transfer

caller  callee

?  ?

[Diagram showing a process and a checkmark]

5
Permissions (≈ Separation Logic)

Logical properties

\[ \text{acc}(x.f) \ast x.f = 0 \]  (≈ \[ x.f \mapsto 0 \])

Disjointness: \( x \neq y \)

\[ \text{acc}(x.f) \ast \text{acc}(y.f) \]  (≈ \[ x.f \mapsto _\ast y.f \mapsto _ \])
Separating Conjunction

\[ A \star B \]

describes the current state in terms of disjoint substates
Separating Conjunction

A ★ B

describes the current state in terms of disjoint substates

Is at the heart of verifiers based on separation logic
Magic Wands

$A \rightarrow^* B$

describes hypothetical states

Read as a promise: “In any state, if you provide $A$, then you will get $B$”
Scenario: Iteratively traverse a recursively defined tree
(Verification Challenge at VerifyThis@FM’12)
Scenario: Iteratively traverse a recursively defined tree

Loop invariant: Describe partial data structure
Partial Data Structures as Magic Wands

Indirectly describe partial data structure as a promise
Modus-Ponens-like rule makes promise applicable

\[
\left( \begin{array}{c}
\Delta
\end{array} \right) \ast \left( \begin{array}{c}
\Delta
\end{array} \right)
\]
Partial Data Structures as Magic Wands

\[ \sigma \models A \rightarrow B \iff \forall \sigma' \cdot (\sigma' \models A \Rightarrow \sigma \uplus \sigma' \models B) \]
Used in various pen & paper proofs
- Partial data structures
- Usage protocols for data structures
- Synchronisation barriers
- ...

Typically* not supported in automatic verifiers
\[ \sigma \models A \rightarrow \! \! \! \# B \iff \forall \sigma' \cdot (\sigma' \models A \Rightarrow \sigma \cup \sigma' \models B) \]

* Only exception we are aware of is VerCors; developed in parallel
Entailment of magic wand formulas is undecidable

→ Lightweight user guidance to direct verification
Guidance: Ghost Operations + Specifications

Make a promise

Pass it around

Use it

package $A \rightarrow \ast B$

Opaque resource; Specifications

apply $A \rightarrow \ast B$
Challenge:
Ensure soundness of apply in any (future) state
Footprints of $A \rightarrow^* B$

Permissions guaranteeing that giving up $A \ast (A \rightarrow^* B)$ and obtaining $B$ is **sound**

![Diagram depicting the concept with footprints and an arrow]

carved out

effectively immutable
Footprints of $A \rightarrow B$

Footprints are **not unique**

or all available permissions
Footprints are not unique

How to choose a footprint?

or all available permissions

or
package A \rightarrow (B_1 \ast B_2 \ast \ldots \ast B_n)

0. given current state

Footprint Computation Algorithm: Setup
Footprint Computation Algorithm: Setup

\[ \text{package } A \rightarrow \ast (B_1 \ast B_2 \ast \ldots \ast B_n) \]

1. create LHS state

1. create current state
Footprint Computation Algorithm: Setup

package  $A \rightarrow^* (B_1 \ast B_2 \ast \ldots \ast B_n)$

2. create empty RHS state
package A →∗ (B₁ ∗ B₂ ∗ ... ∗ Bₙ)

3. iterate over Bᵢ’s:

LHS state

current state

RHS state
Footprint Computation Algorithm: Execution

package A →* (B₁ * B₂ * ... * Bₙ)

3. iterate over Bᵢ’s: If Bᵢ is acc(x.f) then transfer permissions and assumptions
3. iterate over $B_i$’s: If $B_i$ is a logical property $P$, e.g. $\mathbf{x} \cdot \mathbf{f} == \emptyset$, then check $P$.

Footprint Computation Algorithm: Execution

```java
package A ▸ (B_1 ▸ B_2 ▸ ... ▸ B_n)
```
package `acc(x.f) → acc(x.f)`

Examples
Examples

\texttt{package true \rightarrow acc(x.f)}
package true → acc(x.f) * x.f == 0

Examples
Examples

```
package acc(x.f) → acc(x.f) ∗ x.f == 0
```

LHS state

current state

RHS state

≠ x.f == 0
Abstract predicates for recursive data structures

x == null

x

Existing Features
Abstract predicates plus ghost operations

Existing Ghost Operations

unfold
tree(x)

fold
tree(x)

\[
\text{fold tree}(x) \rightarrow \text{X} \rightarrow \text{X} \leftarrow \text{X} \rightarrow \text{fold tree}(x)
\]

\[
x \times x \leftarrow \text{tree(x)} \rightarrow x \times x \leftarrow \text{fold tree}(x)
\]
Integrating Existing Ghost Operations

package →∗
Integrating Existing Ghost Operations

package → * (fold tree(x) in )
Integrating Existing Ghost Operations

package \rightarrow (\text{fold} \ \text{tree}(x) \ \text{in} \ \text{RHS state})
Integrating Existing Ghost Operations

package →∗ (fold tree(x) in )
Implementation

Part of **Viper** verification infrastructure
- Implementation based on symbolic execution
- Rich logic: unrestricted abstract predicates, abstraction functions, quantifiers, sets, sequences, custom mathematical domains, flexible permission model, ...

Set of *interesting examples*; 1.6 to 3 seconds

**Verification challenge** from VerifyThis’12
- Verifies in 3s
- VerCors:
  - 6 minutes (originally, using Chalice/Boogie)
  - 60 seconds (currently, using **Viper**)

38
Annotation Inference Heuristics

Simple **heuristics** to **infer** package and apply statements

Infers **all** package and apply statements in our examples

Verification time: +0.5s or less
VerifyThis’12 Challenge Revisited

Scenario: **Iteratively traverse a recursively defined tree**

Loop invariant: **Describe partial data structure**

Diagram:
- Node A
- Node B
- Path A → B
- Node A is marked with a star

Diagram illustrates a recursive traversal of a tree structure.
VerifyThis’12 Challenge Encoded
VerifyThis’12 Challenge Encoded

Named shorthand, could be inlined

Required in either case

Inferred by heuristics
Algorithm for computing wand footprints

- Sound (proof sketch)
- Permissive and predictable

Formalised verifier-independently

Implementation

- Co-first* to support magic wands in an automatic verifier
- Lightweight user annotations
- Convincing initial results (expressiveness, performance)

www.pm.inf.ethz.ch/research/viper.html

* VerCors