Inferring Object Invariants

Bar-Yeh Evan Chang
University of California, Berkeley
January 21, 2005

K. Rustan M. Leino
Microsoft Research

AIOL 2005
Paris, France

Too Naive

class B {
    int x;
    B() { this.x := 8; }
    void M(B b) {
        this.x := b.x + 1;
        // this.x = ?
    }
    invariant this.x >= 8;
}

Object Invariant Methodology

- Methodology distinguishes between “valid” and “mutable” objects
- Unpack/pack block where the object invariant may be temporarily violated
  - Unpack o from T
  - Pack o as T
  - Object invariant may be violated
  - Fields may be modified
  - Object invariant holds
  - Fields cannot be modified

Outline

- Overview
- Obtaining Object Invariants
  - Abstract State
  - Abstract Transition for Unpack/Pack
- Example
- Concluding Remarks

Too Naive

class C {
    int x;
    C() { this.x := 8; }
    void M(C c) {
        c.x := 7;
        this.x := 8;
        // this.x = ?
    }
    invariant this.x >= 8;
}

What to Infer?

class A {
    int x;
    A() { this.x := 8; }
    void M() {
        this.x := this.x + 1;
    }
    int N(int y) { return y / this.x; }
    invariant this.x >= 8
}
Overview of Contributions

- To extend standard reasoning to fields
  - use parameterized abstract interpretation framework [VMCAI 2005]
- To simultaneously track an unbounded number of objects
  - treat “valid” objects collectively following object invariant methodology [JOT 2004]
- To capture interaction between multiple object invariants
  - separate analysis into flow-sensitive and flow-insensitive parts
  - interaction given by object invariant methodology

Abstract State

- Kind of invariants obtained given by policy domain $\mathcal{P}$
  - e.g., linear arithmetic if $\mathcal{P}$ is Polyhedra
- Standard local flow-sensitive abstract interpretation using $\mathcal{C}(\mathcal{P}, \mathcal{S})$
  - policy domain extended to work with fields
- Global flow-insensitive part captured by $\mathcal{I}$
  - mappings of the form $\mathcal{T} \mapsto \mathcal{C}(\mathcal{P})$
    - e.g., $B \mapsto \text{sel}(H, t, x) \geq 8$, which concretizes to $(\forall t: B \cdot (\forall H \cdot \text{sel}(H, t, x) \geq 8))$

Abstract Transition

- Write the abstract transition for a statement $s$
  $$s: \langle I \triangleright C \rangle \rightarrow \langle I' \triangleright C' \rangle$$
  - one $C$ per program point
  - one global $I$
- All statements except pack/unpack affect only the local state $C$
  - including field updates because methodology says that only “mutable” objects can be modified

Abstract Transition: Pack

- Incorporate the information obtained from the local analysis on pack
  $$P = C \uparrow \text{sel}(H, o, s_T)$$
  pack $o$ as $T$: $\langle I \triangleright C \rangle \rightarrow \langle I'[T \mapsto \{t/o\} \triangleright C \rangle$
  Extract constraints that involve fields of $o$ that are declared in $T$ or a superclass of $T$
  Rename $o$ to $t$ and widen into the current object invariant for $T$
  - note: does not depend on class context

Abstract Transition: Unpack

- Instantiate current object invariant on unpack
  $$\text{unpack } o \text{ from } T: \langle I \triangleright C \rangle \rightarrow \langle I \triangleright C \cap [o/t] I(T) \rangle$$
  - methodology says that the object satisfies the object invariant right before the unpack
  - note: does not depend on class context
- Also instantiate object invariant for, say, “valid” method arguments

Outline

- Overview
- Obtaining Object Invariants
  - Abstract State
  - Abstract Transition for Unpack/Pack
- Example
- Concluding Remarks
Example

```java
class D extends B {
    int y;
    D() { this.y = 1; pack this as D; }
}
class B {
    int x;
    B() { this.x = 8; pack this as B; }
    void M(D d) {
        unpack this from
        this.x := d.x + d;
        pack this as B;
    }
    I: D → sel(H,t,y) = 1, B → sel(H,t,x) = 8
}
```

Example

```java
class D extends B {
    int y;
    D() { this.y = 1; pack this as D; }
}
class B {
    int x;
    B() { this.x = 8; pack this as B; }
    void M(D d) {
        unpack this from
        this.x := d.x + d;
        pack this as B;
    }
    I: D → sel(H,t,y) = 1, B → sel(H,t,x) ≥ 8
```