Inferring Object Invariants

Bor-Yuh Evan Chang
University of California, Berkeley

K. Rustan M. Leino
Microsoft Research

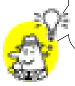
January 21, 2005

AIOOL 2005
Paris, France

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
}
```



Treat “this.x” as one variable

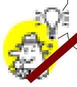
Used to show divide by zero impossible

Too Naïve

```
class B {
  int x;
  B() { this.x := 8; }
  void M(B b) {
      this.x := b.x + 1;
  }

  invariant this.x ≥ 8;
}
```

this.x == ?



~~Treat “this.x” as one variable~~

Too Imprecise

Too Naïve

```
class C {
  int x;
  C() { this.x := 8; }
  void M(C c) {
      c.x := 7;
  }

  ~~invariant this.x == 8;~~
}
```

this.x == 8

~~Treat “this.x” as one variable~~

Unsound

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

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 $\mathrm{T} \mapsto \mathcal{C}(\mathcal{P})$
    - e.g., $\mathrm{B} \mapsto \mathrm{sel}(\mathrm{H},\mathrm{t},\mathrm{x}) \geq 8$, which concretizes to $(\forall \mathrm{t}{:}\mathrm{B} \bullet (\forall \mathrm{H} \bullet\ \mathrm{sel}(\mathrm{H},\mathrm{t},\mathrm{x}) \geq 8))$

Abstract Transition

- Write the abstract transition for a statement s

$$s\colon \langle I \fatsemi C \rangle \longrightarrow \langle I' \fatsemi C' \rangle$$

  - 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**

$$\frac{P \;=\; C \uparrow \mathsf{sel}(H, o, *_T)}{\textbf{pack}\ o\ \textbf{as}\ T\colon \langle I \fatsemi C \rangle \to \langle I[T \mapsto I(T) \mathbin{\nabla} [t/o]P] \fatsemi C \rangle}$$

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**

$$\frac{}{\textbf{unpack}\ o\ \textbf{from}\ T\colon \langle I \fatsemi C \rangle \to \langle I \fatsemi C \sqcap [o/t]I(T) \rangle}$$

  - 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

```
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 B;
        this.x := d.x + d.y;
        pack this as B;
   }
}
```

$\top$

I: D ↦ ⊥ , B ↦ ⊥

Example

```
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 B;
        this.x := d.x + d.y;
        pack this as B;
   }
}
```

sel(H,this,y) = 1

I: D ↦ ⊥ , B ↦ ⊥

Example

```
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 B;
        this.x := d.x + d.y;
        pack this as B;
   }
}
```

sel(H,this,y) = 1

I: D ↦ sel(H,t,y) = 1 , B ↦ ⊥

Example

```
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 B;
        this.x := d.x + d.y;
        pack this as B;
   }
}
```

sel(H,this,y) = 1

sel(H,this,x) = 8

I: D ↦ sel(H,t,y) = 1 , B ↦ sel(H,t,x) = 8

Example

```
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 B;
        this.x := d.x + d.y;
        pack this as B;
   }
}
```

Assume d is valid

sel(H,this,y) = 1

sel(H,this,x) = 8

sel(H,d,y) = 1 ∧ sel(H,d,x) = 8

I: D ↦ sel(H,t,y) = 1 , B ↦ sel(H,t,x) = 8

Example

```
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;
   }
}
```

sel(H,this,y) = 1

sel(H,this,x) = 8

sel(H,d,y) = 1 ∧ sel(H,d,x) = 8

sel(H,d,y)=1∧sel(H,d,x)=8∧sel(H,this,x)=8

I: D ↦ sel(H,t,y) = 1 , B ↦ sel(H,t,x) = 8

Example

```
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 B;
         this.x := d.x + d.y;
         pack this as B;
   }
}
```

sel(H,this,y) = 1

sel(H,this,x) = 8

sel(H,d,y) = 1 ∧ sel(H,d,x) = 8

sel(H,d,y)=1∧sel(H,d,x)=8∧sel(H,this,x)=8

sel(H,this,x) = 9

I: D ↦ sel(H,t,y) = 1 , B ↦ sel(H,t,x) = 8

Example

```
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 B;
         this.x := d.x + d.y;
         pack this as B;
   }
}
```

sel(H,this,y) = 1

Re-examine where previous object invariant for B was instantiated

sel(H,this,x) = 8

sel(H,d,y) = 1 ∧ sel(H,d,x) = 8

sel(H,d,y)=1∧sel(H,d,x)=8∧sel(H,this,x)=8

sel(H,this,x) = 9

I: D ↦ sel(H,t,y) = 1 , B ↦ sel(H,t,x) ≥ 8

Example: Remarks

- Weakening of object invariant must be with widening (like for loops)

$9 \geqslant \mathsf{sel}(H,t,\mathsf{x}) \geqslant 8 \quad 10 \geqslant \mathsf{sel}(H,t,\mathsf{x}) \geqslant 8 \quad 11 \geqslant \mathsf{sel}(H,t,\mathsf{x}) \geqslant 8 \quad \ldots$

- Needed the object invariant of D to obtained the desired object invariant for B
- Incorporating information into object invariant determined by **unpack**/**pack**, not class context

---

Conclusion and Future Work

- Designed a technique to infer more precise object invariants
  - based on distinguishing “valid” and “mutable” objects
  - for class-based object-oriented langauges
    - works with inheritance
    - can obtain invariants on fields of a field using a notion of ownership (see paper)
- Combined with a methodology (strengthening both)
- Fits nicely in an abstract interpretation framework (e.g., Logozzo)

- Future Work: implementation and experiments

Thank you!

Questions? Comments?