Towards imperative modules: reasoning about invariants and sharing of mutable state

David A. Naumann* and Mike Barnett

naumann@cs.stevens-tech.edu, mbarnett@microsoft.com

Stevens Institute of Technology and Microsoft Research

*Partially supported by NSF CCR-0208984 and Microsoft.

Programmer's intro: object invariants

```
class Subject {
   private x,y: int := 0,1;
   invariant 𝓘(self)    where 𝓘(o) = o.x<o.y
   method m() {
      self.x := self.x+1;

      self.y := self.y+1; }    ...}
```

Programmer's intro: object invariants

```
class Subject {
   private x,y: int := 0,1; obs: Observer :=. . . ;
   invariant 𝓘(self)    where 𝓘(o) = o.x<o.y
   method m() {
      self.x := self.x+1;
      obs.notify();
      self.y := self.y+1; }    . . . }
```

Programmer's intro: object invariants

```
class Subject {
   private x,y: int := 0,1; obs: Observer :=. . . ;
   invariant 𝓘(self)    where 𝓘(o) = o.x<o.y
   method m() {
      self.x := self.x+1;
      obs.notify();
      self.y := self.y+1; }    . . . }
class Observer {
   z: Subject := . . . ;
   method notify() { z.m(); }    . . . }
```

Programmer's intro: object invariants

```
class Subject {
   private x,y: int := 0,1; obs: Observer :=. . . ;
   invariant 𝓘(self)    where 𝓘(o) = o.x<o.y
   method m() {
      self.x := self.x+1;
      obs.notify();
      self.y := self.y+1; }    . . . }
class Observer {
   z: Subject := . . . ;
   method notify() { z.m(); }    . . . }
```

When should $\mathcal{I}$ hold?

Programmer's intro (2): sharing

```
class Subject2 {
   private x: Integer := new Integer(0);
   private y: Integer := new Integer(1);
   invariant 𝓘(self)    where 𝓘(o) = o.x.val<o.y.val
   method m() { self.x.incr(); self.y.incr(); }
                                                    }
```

Programmer's intro (2): sharing

```
class Subject2 {
   private x: Integer := new Integer(0);
   private y: Integer := new Integer(1);
   invariant 𝓘(self)    where 𝓘(o) = o.x.val<o.y.val
   method m() { self.x.incr(); self.y.incr(); }
   method leak(): Integer { result := x; }     }
class Main {
   s: Subject2; i: Integer;
   . . . i := s.leak(); i.incr(); s.m() . . . }
```

Programmer's intro (2): sharing

```
class Subject2 {
   private x: Integer := new Integer(0);
   private y: Integer := new Integer(1);
   invariant 𝓘(self)    where 𝓘(o) = o.x.val<o.y.val
   method m() { self.x.incr(); self.y.incr(); }
   method leak(): Integer { result := x; }     }
class Main {
   s: Subject2; i: Integer;
   . . . i := s.leak(); i.incr(); s.m() . . . }
```

How to encapsulate not just fields but also referenced objects?

Logician's intro

$$\frac{\{\mathcal{P} \wedge \mathcal{I}\}\ \mathtt{body}\ \{\mathcal{Q} \wedge \mathcal{I}\} \qquad \mathcal{I} \text{ is encapsulated for } \mathtt{m}}{\{\mathcal{P}\}\ \textbf{call}\ \mathtt{m}()\ \{\mathcal{Q}\}}$$

Declaration: $\mathtt{m}()\{\mathtt{body}\}$. *Specification* $\mathcal{P}, \mathcal{Q}$ *involves public fields;* $\mathcal{I}$ *depends on the internal representation.*

Logician's intro

$$\frac{\{\mathcal{P} \wedge \mathcal{I}\}\ \mathtt{body}\ \{\mathcal{Q} \wedge \mathcal{I}\} \qquad \mathcal{I} \text{ is encapsulated for } \mathtt{m}}{\{\mathcal{P}\}\ \textbf{call}\ \mathtt{m}()\ \{\mathcal{Q}\}}$$

Declaration: $\mathtt{m}()\{\mathtt{body}\}$. *Specification* $\mathcal{P}, \mathcal{Q}$ *involves public fields;* $\mathcal{I}$ *depends on the internal representation.*

$$\frac{\{\mathcal{P}\}\ S\ \{\mathcal{Q}\} \qquad S \text{ does not interfere* with } \mathcal{I}}{\{\mathcal{P} \wedge \mathcal{I}\}\ S\ \{\mathcal{Q} \wedge \mathcal{I}\}}$$

Logician's intro

$$\frac{\{\mathcal{P} \wedge \mathcal{I}\}\ \mathtt{body}\ \{\mathcal{Q} \wedge \mathcal{I}\} \qquad \mathcal{I} \text{ is encapsulated for } \mathtt{m}}{\{\mathcal{P}\}\ \textbf{call}\ \mathtt{m}()\ \{\mathcal{Q}\}}$$

Declaration: $\mathtt{m}()\{\mathtt{body}\}$. *Specification* $\mathcal{P}, \mathcal{Q}$ *involves public fields;* $\mathcal{I}$ *depends on the internal representation.*

$$\frac{\{\mathcal{P}\}\ S\ \{\mathcal{Q}\} \qquad S \text{ does not interfere* with } \mathcal{I}}{\{\mathcal{P} \wedge \mathcal{I}\}\ S\ \{\mathcal{Q} \wedge \mathcal{I}\}}$$

* S *does not write variables read in* $\mathcal{I}$ *(hazard: aliasing)*

* S *does not update objects read in* $\mathcal{I}$ *(hazard: heap sharing)*

Logicians' intro (2)

$$\dfrac{\dfrac{}{\{\mathcal{R}\quad\}\, \mathrm{x} := \mathrm{E}\, \{\mathcal{P}\quad\}} \qquad \dfrac{\dfrac{\vdots}{\{\mathcal{P} \wedge \mathcal{I}\}\, \mathrm{body}\, \{\mathcal{Q} \wedge \mathcal{I}\}}}{\{\mathcal{P}\quad\}\, \textbf{call}\ \mathrm{m}\, \{\mathcal{Q}\quad\}}}{\{\mathcal{R}\quad\}\, \mathrm{x} := \mathrm{E}\, ;\ \textbf{call}\ \mathrm{m}\, \{\mathcal{Q}\quad\}}$$

$$\vdots$$

Logicians' intro (2)

$$\dfrac{\dfrac{\{\mathcal{R}\}\ \mathrm{x} := \mathrm{E}\ \{\mathcal{P}\} \quad \text{no interference}}{\{\mathcal{R}\wedge\mathcal{I}\}\ \mathrm{x} := \mathrm{E}\ \{\mathcal{P}\wedge\mathcal{I}\}} \qquad \dfrac{\vdots}{\dfrac{\{\mathcal{P}\wedge\mathcal{I}\}\ \mathrm{body}\ \{\mathcal{Q}\wedge\mathcal{I}\}}{\{\mathcal{P}\wedge\mathcal{I}\}\ \textbf{call}\ \mathrm{m}\ \{\mathcal{Q}\wedge\mathcal{I}\}}}}{\{\mathcal{R}\wedge\mathcal{I}\}\ \mathrm{x} := \mathrm{E}\ ;\ \textbf{call}\ \mathrm{m}\ \{\mathcal{Q}\wedge\mathcal{I}\}}$$

$$\vdots$$

Logicians' intro (2)

$$\dfrac{\dfrac{\{\mathcal{R}\}\ \mathrm{x} := \mathrm{E}\ \{\mathcal{P}\} \quad \text{no interference}}{\{\mathcal{R}\wedge\mathcal{I}\}\ \mathrm{x} := \mathrm{E}\ \{\mathcal{P}\wedge\mathcal{I}\}} \qquad \dfrac{\dfrac{}{\vdots}}{\dfrac{\{\mathcal{P}\wedge\mathcal{I}\}\ \mathrm{body}\ \{\mathcal{Q}\wedge\mathcal{I}\}}{\{\mathcal{P}\wedge\mathcal{I}\}\ \textbf{call}\ \mathrm{m}\ \{\mathcal{Q}\wedge\mathcal{I}\}}}}{\{\mathcal{R}\wedge\mathcal{I}\}\ \mathrm{x} := \mathrm{E}\ ;\ \textbf{call}\ \mathrm{m}\ \{\mathcal{Q}\wedge\mathcal{I}\}}$$

$$\vdots$$

$$\{\mathcal{R}\}\ \mathrm{init};\ \{\mathcal{R}\wedge\mathcal{I}\}\ \mathrm{x} := \mathrm{E}\ ;\ \textbf{call}\ \mathrm{m}\ \{\mathcal{Q}\wedge\mathcal{I}\} \Rightarrow \{\mathcal{Q}\}$$

Logicians' intro (2)

$$\dfrac{\dfrac{\{\mathcal{R}\}\,\mathrm{x} := \mathrm{E}\,\{\mathcal{P}\} \quad \text{no interference}}{\{\mathcal{R}\wedge\mathcal{I}\}\,\mathrm{x} := \mathrm{E}\,\{\mathcal{P}\wedge\mathcal{I}\}} \qquad \dfrac{\vdots}{\dfrac{\{\mathcal{P}\wedge\mathcal{I}\}\,\mathrm{body}\,\{\mathcal{Q}\wedge\mathcal{I}\}}{\{\mathcal{P}\wedge\mathcal{I}\}\,\textbf{call}\ \mathrm{m}\,\{\mathcal{Q}\wedge\mathcal{I}\}}}}{\{\mathcal{R}\wedge\mathcal{I}\}\,\mathrm{x} := \mathrm{E}\,;\ \textbf{call}\ \mathrm{m}\,\{\mathcal{Q}\wedge\mathcal{I}\}}$$

$$\vdots$$

$$\{\mathcal{R}\}\,\mathrm{init};\ \{\mathcal{R}\wedge\mathcal{I}\}\,\mathrm{x} := \mathrm{E}\,;\ \textbf{call}\ \mathrm{m}\,\{\mathcal{Q}\wedge\mathcal{I}\} \Rightarrow \{\mathcal{Q}\}$$

How generalize to multiple instantiation, i.e., $\mathcal{I}(\mathrm{o})$ for all o?

What about outcalls in body, i.e. method invocations on other objects?

How to express absence of interference, with heap sharing?

Contribution - a discipline

Problems:

- due to re-entrant callbacks, precondition $\mathcal{P} \wedge \mathcal{I}$ for $\mathtt{body}$ is unsound unless $\mathcal{I}$ re-established before outcalls

Contribution - a discipline

Problems:

- due to re-entrant callbacks, precondition $\mathcal{P} \wedge \mathcal{I}$ for $\mathtt{body}$ is unsound unless $\mathcal{I}$ re-established before outcalls
- need to protect $\mathcal{I}_{\mathrm{Subject}}(\mathrm{o})$ from interference —by code in other classes and by other instances $\mathrm{p} \neq \mathrm{o}$ of Subject

Contribution - a discipline

Problems:

- due to re-entrant callbacks, precondition $\mathcal{P} \wedge \mathcal{I}$ for $\mathtt{body}$ is unsound unless $\mathcal{I}$ re-established before outcalls
- need to protect $\mathcal{I}_{\mathrm{Subject}}(\mathrm{o})$ from interference —by code in other classes and by other instances $\mathrm{p} \neq \mathrm{o}$ of Subject

Contribution: formalize and prove sound a solution using a single everywhere-invariant, $\mathcal{PI}$. Rule: $\dfrac{\{\mathcal{PI} \wedge \mathcal{P}\}\ S\ \{\mathcal{Q}\}}{\{\mathcal{P}\}\ S\ \{\mathcal{Q}\}}$.

Contribution - a discipline

Problems:

- due to re-entrant callbacks, precondition $\mathcal{P} \wedge \mathcal{I}$ for $\mathtt{body}$ is unsound unless $\mathcal{I}$ re-established before outcalls
- need to protect $\mathcal{I}_{\mathrm{Subject}}(\mathrm{o})$ from interference —by code in other classes and by other instances $\mathrm{p} \neq \mathrm{o}$ of Subject

Contribution: formalize and prove sound a solution using a single everywhere-invariant, $\mathcal{PI}$. Rule: $\dfrac{\{\mathcal{PI} \wedge \mathcal{P}\}\ S\ \{\mathcal{Q}\}}{\{\mathcal{P}\}\ S\ \{\mathcal{Q}\}}$.

Handles transfer and sharing of objects across encapsulation boundaries. Can use with standard logics.

Auxiliary field to make explicit when invariant holds:

inv : **boolean** := false

Maintain program invariant $\mathcal{PI} : (\ \forall\, \mathrm{o} \bullet \mathrm{o}.\mathit{inv} \Rightarrow \mathcal{I}_{type(\mathrm{o})}(\mathrm{o})\ )$

Auxiliary field to make explicit when invariant holds:

inv : **boolean** := $false$

Maintain program invariant $\mathcal{PI} : (\forall o \bullet o.inv \Rightarrow \mathcal{I}_{type(o)}(o))$

class Subject { . . .

invariant $\mathcal{I}_{Subject}$(self) where $\mathcal{I}_{Subject}$(o) = o.x<o.y

method m() {

assert self.inv (* precondition *)

unpack self; (* self.inv := false *)

self.x := self.x+1; obs.notify(); self.y := self.y+1; (* $\mathcal{I}$(self) *)

pack self; (* self.inv := true *) } . . . }

class Main . . . **method** notify() { **assert** z.inv ?; z.m(); }

Auxiliary field to make explicit when invariant holds:

inv : **boolean** := $false$

Maintain program invariant $\mathcal{PI} : (\forall o \bullet o.inv \Rightarrow \mathcal{I}_{type(o)}(o))$

class Subject { . . .

invariant $\mathcal{I}_{Subject}$(self) where $\mathcal{I}_{Subject}$(o) = o.x<o.y

method m() {

assert self.inv (* precondition *)

unpack self; (* self.inv := false *)

self.x := self.x+1; obs.notify(); self.y := self.y+1; (* $\mathcal{I}$(self) *)

pack self; (* self.inv := true *) } . . . }

class Main . . . **method** notify() { **assert** z.inv ?; z.m(); }

Absence of interf., as a precond.: $\{\neg v.inv \wedge \mathcal{PI}\}\, v.f := E\, \{\mathcal{PI}\}$

Auxiliary field to delimit heap dependence:

$\mathrm{own} : \mathrm{Object} := \mathrm{null}$

Def: $\mathrm{o} \preceq \mathrm{p}$ *iff either* $\mathrm{o} = \mathrm{p.own}$ *or* $\mathrm{o} \preceq \mathrm{p.own}$.

Auxiliary field to delimit heap dependence:

$\mathrm{own} : \mathrm{Object} := \mathrm{null}$

Def: $\mathrm{o} \preceq \mathrm{p}$ *iff either* $\mathrm{o} = \mathrm{p.own}$ *or* $\mathrm{o} \preceq \mathrm{p.own}$.

Def: $\mathcal{I}_{\mathrm{C}}$ *is admissible iff*

when $\mathcal{I}_{\mathrm{C}}(\mathrm{o})$ *depends on* $\mathrm{p.f}$ *then either* $\mathrm{o} = \mathrm{p}$ *or* $\mathrm{o} \preceq \mathrm{p}$.

Auxiliary field to delimit heap dependence:

$\mathrm{own} : \mathrm{Object} := \mathrm{null}$

Def: $\mathrm{o} \preceq \mathrm{p}$ *iff either* $\mathrm{o} = \mathrm{p.own}$ *or* $\mathrm{o} \preceq \mathrm{p.own}$.

Def: $\mathcal{I}_C$ *is admissible iff*

when $\mathcal{I}_C(\mathrm{o})$ *depends on* $\mathrm{p.f}$ *then either* $\mathrm{o} = \mathrm{p}$ *or* $\mathrm{o} \preceq \mathrm{p}$.

Absence of interference, as a precondition:

$\{\neg v.\mathrm{inv} \wedge (\forall\, \mathrm{o} \bullet \mathrm{o} \preceq v \Rightarrow \neg \mathrm{o.inv}) \wedge \mathcal{PI}\}\, v.\mathrm{f} := \mathrm{E}\, \{\mathcal{PI}\}$

Ownership provides stateful encapsulation: $\neg \mathrm{o.inv}$ *means control is inside the boundary for* o.

Last auxiliary field for ownership discipline:

com : **boolean** $:= \mathit{false}$

$$\mathcal{PI}: \ (\forall \mathrm{o} \bullet \mathrm{o}.\mathit{inv} \Rightarrow \mathcal{I}_{\mathit{type}(\mathrm{o})}(\mathrm{o})\ ) \wedge$$
$$(\forall \mathrm{o} \bullet \mathrm{o}.\mathit{inv} \Rightarrow (\forall \mathrm{p} \bullet \mathrm{p}.\mathit{own} = \mathrm{o} \Rightarrow \mathrm{p}.\mathrm{com}\ )) \wedge$$
$$(\forall \mathrm{o} \bullet \mathrm{o}.\mathrm{com} \Rightarrow \mathrm{o}.\mathit{inv}\ )$$

Last auxiliary field for ownership discipline:

com : **boolean** $:= \mathtt{false}$

$$\mathcal{PI}: \ (\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow \mathcal{I}_{type(\mathrm{o})}(\mathrm{o})\ ) \wedge$$
$$(\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow (\forall \mathrm{p} \bullet \mathrm{p.own} = \mathrm{o} \Rightarrow \mathrm{p.com}\ )) \wedge$$
$$(\forall \mathrm{o} \bullet \mathrm{o.com} \Rightarrow \mathrm{o.inv}\ )$$

Absence of interf., as a precond.: $\{\neg v.\mathrm{inv} \wedge \mathcal{PI}\}\ v.\mathrm{f} := \mathrm{E}\ \{\mathcal{PI}\}$

Last auxiliary field for ownership discipline:

com : **boolean** := false

$\mathcal{PI}$: $(\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow \mathcal{I}_{type(\mathrm{o})}(\mathrm{o})\ ) \wedge$

$(\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow (\forall \mathrm{p} \bullet \mathrm{p.own} = \mathrm{o} \Rightarrow \mathrm{p.com}\ )) \wedge$

$(\forall \mathrm{o} \bullet \mathrm{o.com} \Rightarrow \mathrm{o.inv}\ )$

Absence of interf., as a precond.: $\{\neg \mathrm{v.inv} \wedge \mathcal{PI}\}\ \mathrm{v.f} := \mathrm{E}\ \{\mathcal{PI}\}$

Precondition and effect of **unpack** E:

assert $\mathrm{E.inv} \wedge \neg \mathrm{E.com}$;

$\mathrm{E.inv}$:= false; **forall** o **with** $\mathrm{o.own} = \mathrm{E}$ **do** $\mathrm{o.com}$:= false;

Last auxiliary field for ownership discipline:

com : **boolean** := false

$\mathcal{PI}$: $(\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow \mathcal{I}_{type(\mathrm{o})}(\mathrm{o})\ ) \wedge$

$(\forall \mathrm{o} \bullet \mathrm{o.inv} \Rightarrow (\forall \mathrm{p} \bullet \mathrm{p.own} = \mathrm{o} \Rightarrow \mathrm{p.com}\ )) \wedge$

$(\forall \mathrm{o} \bullet \mathrm{o.com} \Rightarrow \mathrm{o.inv}\ )$

Absence of interf., as a precond.: $\{\neg \mathrm{v.inv} \wedge \mathcal{PI}\}\ \mathrm{v.f} := \mathrm{E}\ \{\mathcal{PI}\}$

Precondition and effect of **unpack** E:

assert $\mathrm{E.inv} \wedge \neg \mathrm{E.com}$;

$\mathrm{E.inv}$:= false; **forall** o **with** $\mathrm{o.own} = \mathrm{E}$ **do** $\mathrm{o.com}$:= false;

Precondition and effect of **pack** E:

assert $\neg \mathrm{E.inv} \wedge \mathcal{I}_{type(\mathrm{E})}(\mathrm{E})$;

$\mathrm{E.inv}$:= true; **forall** o **with** $\mathrm{o.own} = \mathrm{E}$ **do** $\mathrm{o.com}$:= true;

Ownership transfer

$\{\neg v.\mathrm{inv} \wedge \neg \mathrm{E}.\mathrm{inv} \wedge \mathcal{PI}\}\ v.\mathrm{own} := \mathrm{E}\ \{\mathcal{PI}\}$

Special command **setowner** *to highlight that it only manipulates auxiliary state (like* **unpack/pack***.*

State-based encapsulation (vs. type systems):

- *avoids restriction on existence or reading of references*
- *allows transfer of objects across boundaries*

Main result: stateful encapsulation I.

Def: S is *properly annotated* iff each **pack**, **unpack**, **setowner**, and field update has stipulated precondition.

Theorem: $\{\mathcal{PI}\}\ S\ \{\mathcal{PI}\}$ for any properly annotated S

Rule: $$\frac{\{\mathcal{PI} \wedge \mathcal{P}\}\ S\ \{\mathcal{Q}\}}{\{\mathcal{P}\}\ S\ \{\mathcal{Q}\}}$$

Proof: using a straightforward denotational semantics for a sequential language with mutually recursive class declarations and methods etc.

Stateful encapsulation II.

A List owns its nodes. A node does not own its neighbors.

class List {
  head: ListNode;
  invariant self.head=null $\vee$ self.head.prev=null; ...}
class ListNode {
  next, prev: ListNode;
  invariant self.next=null
    $\vee$ (self.next.prev=self $\wedge$ self.next.own=self.own); ...}

Stateful encapsulation II.

A List owns its nodes. A node does not own its neighbors.

class List {
  head: ListNode;
  invariant self.head=null $\vee$ self.head.prev=null; ...}
class ListNode {
  next, prev: ListNode;
  invariant self.next=null
    $\vee$ (self.next.prev=self $\wedge$ self.next.own=self.own); ...}

Decentralized invariants express acyclicity without induction. Well behaved interaction but not ownership.

Absence of interference, as a precondition:

$\{\neg v.\mathrm{inv} \wedge$

$\quad (\,\forall\, o \bullet$ "$\mathcal{I}_C(o)$ depends on $v.f$" $\Rightarrow \neg o.\mathrm{inv} \vee \mathcal{I}_C(o)[E/v.f]\,)\}$

$v.f := E$

Absence of interference, as a precondition:

$\{\neg v.inv \wedge$

$\quad (\,\forall\, o \bullet$ "$\mathcal{I}_C(o)$ depends on $v.f$" $\Rightarrow \neg o.inv \vee \mathcal{I}_C(o)[E/v.f]\,)\}$

$v.f := E$

Auxiliary field for friendship discipline:

$deps$: **set of** $Object := \varnothing$

$(\,\forall\, o \bullet o \in v.deps \Rightarrow \neg o.inv \vee \mathcal{I}_C(o)[E/v.f]\,)$

Absence of interference, as a precondition:

$\{\neg v.\mathrm{inv} \wedge$

$\quad (\,\forall\, o \bullet$ "$\mathcal{I}_C(o)$ depends on $v.f$" $\Rightarrow \neg o.\mathrm{inv} \vee \mathcal{I}_C(o)[E/v.f]\,)\}$

$v.f := E$

Auxiliary field for friendship discipline:

$\mathrm{deps} : \textbf{set of } \mathrm{Object} := \varnothing$

$(\,\forall\, o \bullet o \in v.\mathrm{deps} \Rightarrow \neg o.\mathrm{inv} \vee \mathcal{I}_C(o)[E/v.f]\,)$

Admissibility: when $\mathcal{I}_C(o)$ *depends on* $p.f$ *then either* $o = p$, $o \preceq p$, *or* $p = o.g$ *for some declared pivot* g *and* $o \in p.\mathrm{deps}$

Absence of interference, as a precondition:

$\{\neg v.\mathrm{inv} \wedge$

$\quad (\,\forall\, o \bullet$ "$\mathcal{I}_C(o)$ depends on $v.f$" $\Rightarrow \neg o.\mathrm{inv} \vee \mathcal{I}_C(o)[E/v.f]\,)\}$

$v.f := E$

Auxiliary field for friendship discipline:

deps : **set of** $\mathrm{Object} := \varnothing$

$(\,\forall\, o \bullet o \in v.\mathrm{deps} \Rightarrow \neg o.\mathrm{inv} \vee \mathcal{I}_C(o)[E/v.f]\,)$

Admissibility: when $\mathcal{I}_C(o)$ *depends on* $p.f$ *then either* $o = p$, $o \preceq p$, *or* $p = o.g$ *for some declared pivot* g *and* $o \in p.\mathrm{deps}$

Abstract from $\mathcal{I}_C(o)[E/v.f]$ *as* $\mathcal{U}_C(o, v, E)$.

Obligation: $\{\mathcal{I}_C(\mathrm{self}) \wedge \mathcal{U}(\mathrm{self}, g, \mathrm{val})\}\ \mathrm{self}.g.f := \mathrm{val}\ \{\mathcal{I}_C(\mathrm{self})\}$

Conclusion

- Discipline for control of dependence for object invariants. Controls use of pointers rather than their existence.
- Handles difficult design patterns that are common in practice. No restrictions on heap structure.
- No committment to particular program logic or verification system.
- Uses verification conditions; not special type annotation but not fully automated.

Future work

- comparison with Separation Logic:

$$\frac{\{\mathcal{P}\}\ \mathtt{m}\ \{\mathcal{Q}\} \vdash \{\mathcal{P}'\}\ S\ \{\mathcal{Q}'\}}{\{\mathcal{P} * \mathcal{I}\}\ \mathtt{body}\ \{\mathcal{Q} * \mathcal{I}\} \vdash \{\mathcal{P}' * \mathcal{I}\}\ S\ \{\mathcal{Q}' * \mathcal{I}\}}$$

- multi-class patterns
- implementation and case studies —Boogie project
- implementation and full logic —Pierik and de Boer
- representation independence —Banerjee and Naumann
- friends and subclassing —Barnett and Naumann
- machine check soundness proof —Naumann

References

- Barnett, DeLine, Fähndrich, Leino, Wolfram Schulte: *Verification of object-oriented programs with invariants* (Journal of Object Technology '04)
- Leino and Müller: *Object invariants in dynamic contexts* (ECOOP'04)
- Barnett and Naumann: *Friends need a bit more: maintaining invariants over shared state* (MPC'04)
- Pierik, Clarke, de Boer: *Creational invariants* (FTfJP'04)
- O'Hearn, Yang, Reynolds: *Separation and Information Hiding* (POPL'04)