Reasoning about modules: data refinement and simulation

David Naumann

naumann@cs.stevens-tech.edu

Stevens Institute of Technology

Objectives of talk

- Introduce collaborative projects with
  - Paulo Borba, *Ana Cavalcanti*, Augusto Sampaio; Federal U. of Pernambuco, Recife.
  - Uday Reddy; University of Birmingham.
  - Hongseok Yang; Korean Advanced Inst. of Sci. and Tech. (ROPAS).
  - *Anindya Banerjee*; Kansas State University.
- Survey results on simulation for data refinement.
- Suggest an approach to pointer confinement.

Outline

- project overview
- simulation for alias-free programs
- simulation and aliasing (cf. POPL talk)
- confinement and patterns
- sidelights

Refinement calculus for OOP

Specification statements: $x : [pre, post]$ modifies "only x" and establishes $post$ if pre initially; else aborts.
Assertion: $\emptyset : [P, true]$.
Assumption: $\emptyset : [true, P]$.
Correctness: $x : [pre, post] \sqsubseteq prog$

Refinement calculus for OOP

Specification statements: $x : [pre, post]$ modifies "only x" and establishes $post$ if pre initially; else aborts.
Assertion: $\emptyset : [P, true]$.
Assumption: $\emptyset : [true, P]$.
Correctness: $x : [pre, post] \sqsubseteq prog$
Applications:

- Formalize stepw ise development (to guide semi-informal methodology or etc.).
  Proof rules as refinement laws, e.g.,
  $x, y : [pre, post] \sqsubseteq x : [pre, mid];\ y : [mid, post]$
- Specifications, esp. design patterns with callbacks.
- Normal-form compilation.

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.
- Weakest-precondition semantics *without requiring specifications or behavioral subclassing*.

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.
- Weakest-precondition semantics *without requiring specifications or behavioral subclassing*.
- Aim to extend Morgan's imperative refinement calculus, e.g., initially used predicates=formulas.

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.
- Weakest-precondition semantics *without requiring specifications or behavioral subclassing*.
- Aim to extend Morgan's imperative refinement calculus, e.g., initially used predicates=formulas.
- Use semantics to prove basic laws.

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.
- Weakest-precondition semantics *without requiring specifications or behavioral subclassing*.
- Aim to extend Morgan's imperative refinement calculus, e.g., initially used predicates=formulas.
- Use semantics to prove basic laws.
- Compiler status: normal form and VM defined. . .

Project overview

- Sequential fragment of Java including: dynamic dispatch, recursion, visibility control.
  Omit: *pointers*, exceptions, concurrency.
- Weakest-precondition semantics *without requiring specifications or behavioral subclassing*.
- Aim to extend Morgan's imperative refinement calculus, e.g., initially used predicates=formulas.
- Use semantics to prove basic laws.
- Compiler status: normal form and VM defined. . .
- Design laws: unnesting etc., *complete* for rewriting to imperative normal form where classes are used only for type tagging and testing [BS01].

Class refinement

```
class A {
  private Boolean g;  // rep object
  unit init(){ g := new Boolean();
               g.set(~true); }
  unit setg(bool x){ g.set(~x); }
  bool getg(){ return ~g.get(); } }
```

Equivalent or refined behavior for all clients, e.g.

```
z:= new A(); z.setg(true); b:= z.getg();

if ~(pt instanceof ColorPt) skip else abort
```

Soundness of forward simulation

- Uses: info flow, theory underlying behavioral subclassing, data refinement...

Soundness of forward simulation

- Uses: info flow, theory underlying behavioral subclassing, data refinement...
- Prior work – imperative: single variable (vs. **new**), non-recursive type, fixed state space, no pointers, no expressions except in [Nau01b]; functional: deterministic, continuous (no specs)

Soundness of forward simulation

- Uses: info flow, theory underlying behavioral subclassing, data refinement...
- Prior work – imperative: single variable (vs. **new**), non-recursive type, fixed state space, no pointers, no expressions except in [Nau01b]; functional: deterministic, continuous (no specs)
- Soundness of healthy forward simulation for class refinement (private fields) [CN01].

Healthy couplings

- Early work required simulations to be, e.g., surjective functions, but this proved unnecessary and incomplete.

Healthy couplings

- Early work required simulations to be, e.g., surjective functions, but this proved unnecessary and incomplete.
- $[\![\mathbf{var}\ x : T \bullet c]\!]\psi = \forall t \in [\![T]\!] \bullet [\![c]\!]\psi$
  surjective: demonic choice of any concrete initial value t must be matched by some abstract choice.

Healthy couplings

- Early work required simulations to be, e.g., surjective functions, but this proved unnecessary and incomplete.
- $[\![\mathbf{var}\ x : T \bullet c]\!]\psi = \forall t \in [\![T]\!] \bullet [\![c]\!]\psi$
  surjective: demonic choice of any concrete initial value t must be matched by some abstract choice.
- $[\![\mathbf{avar}\ x : T \bullet c]\!]\psi = \exists t \in [\![T]\!] \bullet [\![c]\!]\psi$
  total: e.g., angelic choice makes
  $(\mathbf{avar}\ x : T \bullet \emptyset : [x = v, true])$ behave as $skip$
  but if v has no concrete counterpart, it behaves as
  $(\mathbf{avar}\ x : T \bullet \emptyset : [false, true])$, i.e., $abort$

Healthy couplings

- Early work required simulations to be, e.g., surjective functions, but this proved unnecessary and incomplete.
- $[\![\mathbf{var}\ x : T \bullet c]\!]\psi = \forall t \in [\![T]\!] \bullet [\![c]\!]\psi$
  surjective: demonic choice of any concrete initial value t must be matched by some abstract choice.
- $[\![\mathbf{avar}\ x : T \bullet c]\!]\psi = \exists t \in [\![T]\!] \bullet [\![c]\!]\psi$
  total: e.g., angelic choice makes
  $(\mathbf{avar}\ x : T \bullet\ \emptyset : [x = v, true])$ behave as $skip$
  but if v has no concrete counterpart, it behaves as
  $(\mathbf{avar}\ x : T \bullet\ \emptyset : [false, true])$, i.e., $abort$
- $[\![x : [pre, post]]\!]\psi = pre \wedge \forall x \bullet post \Rightarrow \psi$

Healthy couplings

- Early work required simulations to be, e.g., surjective functions, but this proved unnecessary and incomplete.
- $[\![\mathbf{var}\ x : T \bullet c]\!]\psi = \forall t \in [\![T]\!] \bullet [\![c]\!]\psi$
  surjective: demonic choice of any concrete initial value t must be matched by some abstract choice.
- $[\![\mathbf{avar}\ x : T \bullet c]\!]\psi = \exists t \in [\![T]\!] \bullet [\![c]\!]\psi$
  total: e.g., angelic choice makes $(\mathbf{avar}\ x : T \bullet \emptyset : [x = v, true])$ behave as $skip$
  but if v has no concrete counterpart, it behaves as $(\mathbf{avar}\ x : T \bullet \emptyset : [false, true])$, i.e., $abort$
- $[\![x : [pre, post]]\!]\psi = pre \wedge \forall x \bullet post \Rightarrow \psi$
- A solution: object invariant, choices among satisfying values.

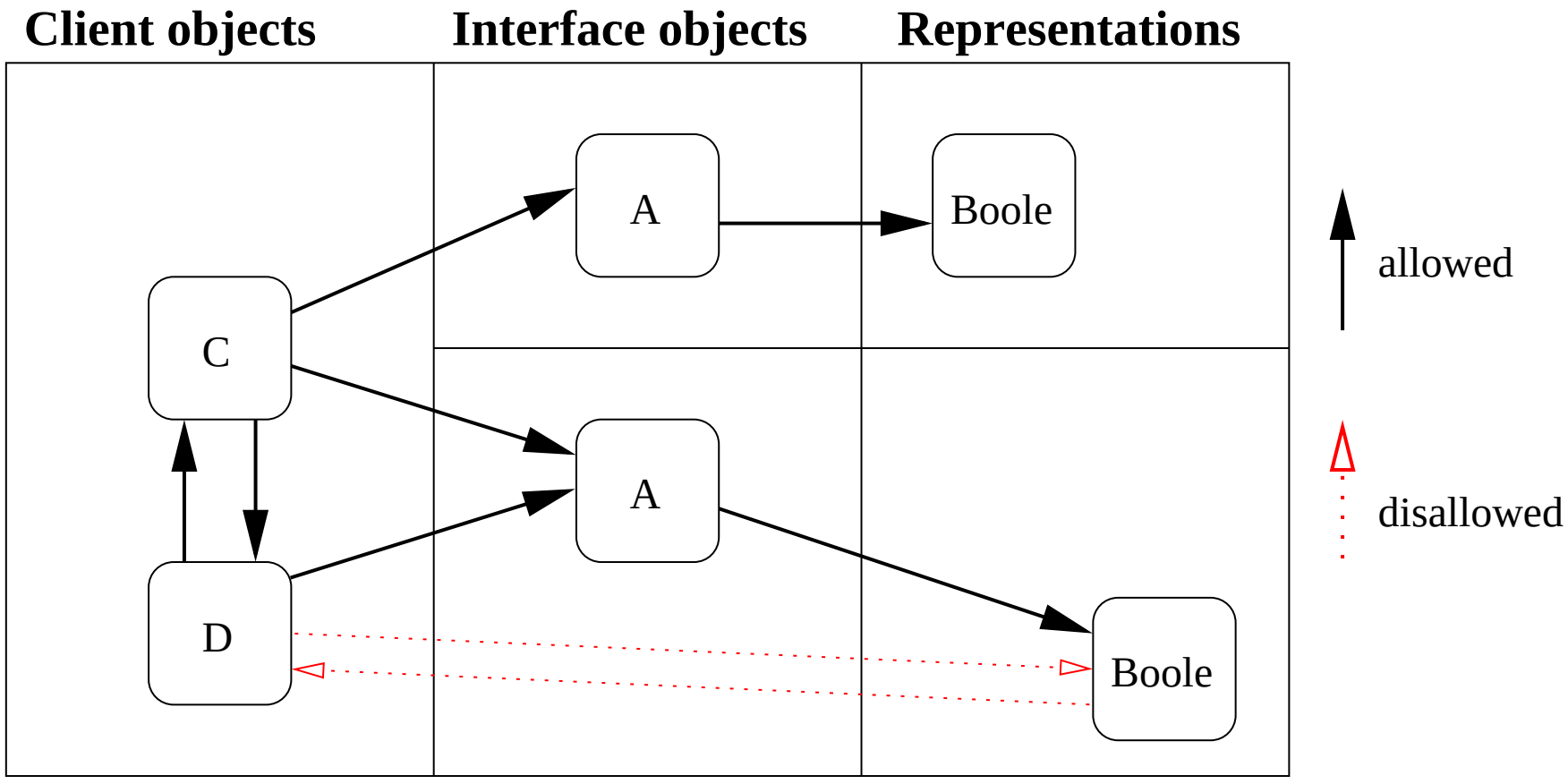
Representation exposure

```
class A {
  private Boolean g;  // rep object
  unit init(){ g := new Boolean();
               g.set(~true); }
  unit setg(bool x){ g.set(~x); }
  bool getg(){ return ~g.get(); }
  Object bad(){ return g; } }
```

Client behavior depends on representation:

```
  z := new A(); w := (Boolean) z.bad();
  if (w.get()) skip else diverge;
```

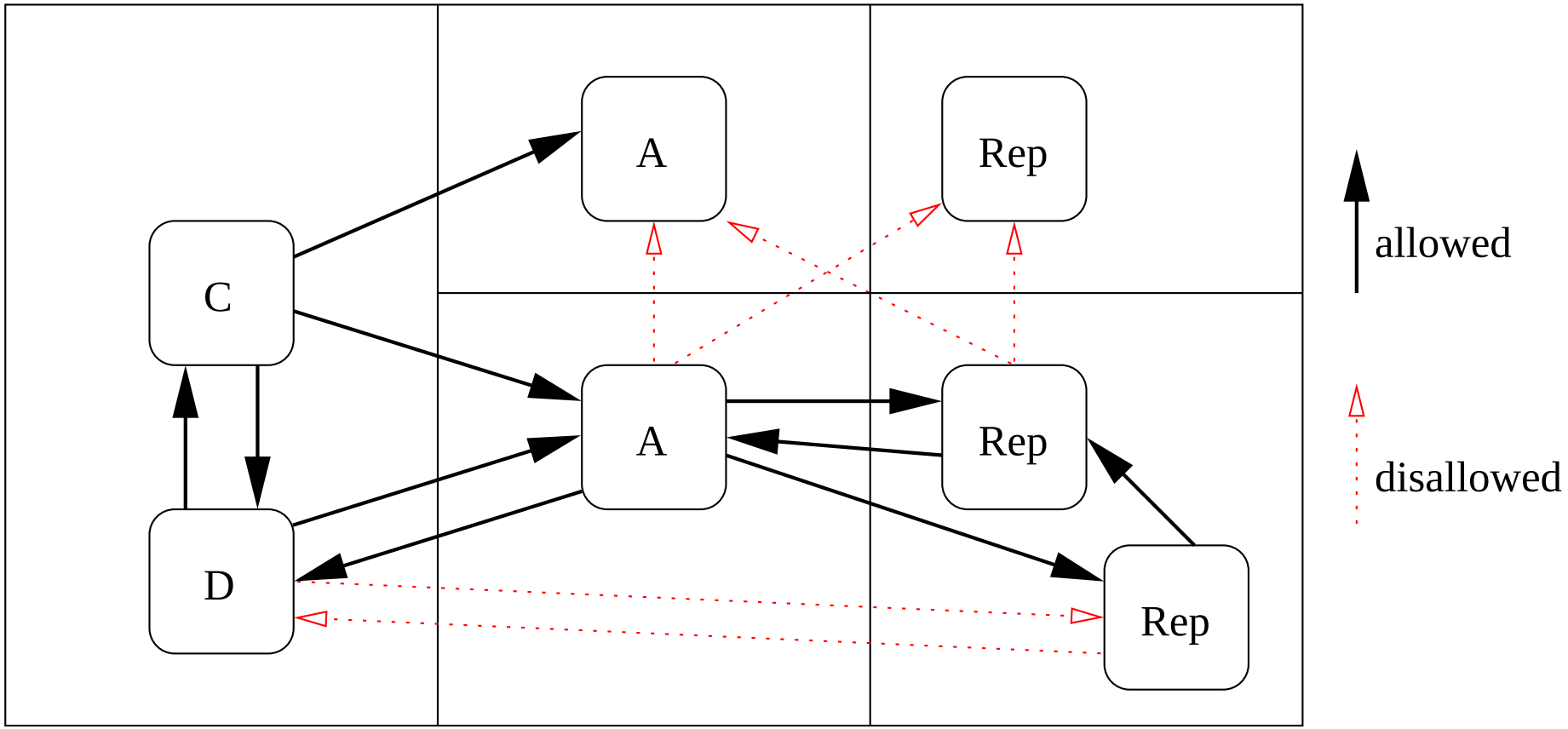
Confinement



Confinement

- Confinement is an end in itself [Boyland,Potter,Clark,Noble,Vitek,Bokowski...]
- Modular reasoning about modification of extended state [Leino,Poetzsch-Heffter,Müller...]
- Reasoning about modules: soundness of forward (bi)simulation for sequential Java (sans specifications and nondeterminacy) [BN02, BN].

Heap confinement for A, Rep



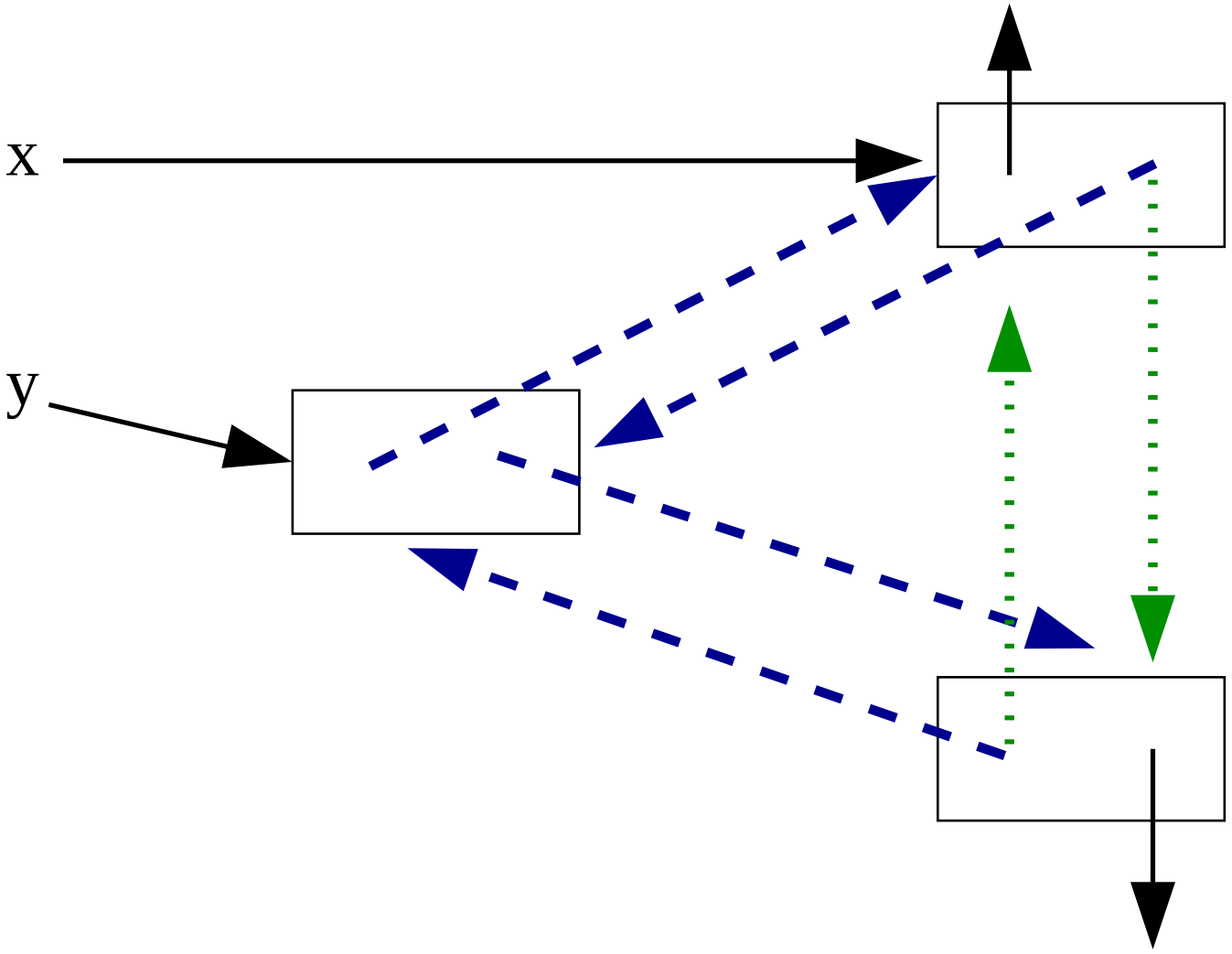
$conf\ h$ iff h has admissible partition

$h = hOut * hA_1 * hRep_1 * \ldots * hA_n * hRep_n$ with

$hOut \not\rightsquigarrow hRep_k$, $hRep_k \not\rightsquigarrow hOut$, and

$hA_k * hRep_k \not\rightsquigarrow hA_j * hRep_j$ for $k \neq j$

Pointers



patvar a, b, c, h

turn $(x \rightarrow a, b) * (y \rightarrow \text{-}, \text{-}) * (b \rightarrow x, c) * h$

into $(x \rightarrow a, y) * (y \rightarrow x, b) * (b \rightarrow y, c) * h$

Explicit confinement

Alternative to type system for tracking confinement.
To maintain $hOut \not\rightarrow hRep$ it suffices to check assignments to cells in $hOut$.

$//heap = hOut * (x \rightarrow \text{-}, b) * hRep$ with $hOut \not\rightarrow hRep$

$y := new\ Rep();$

$//heap = hOut * (x \rightarrow \text{-}, b) * hRep * (y \rightarrow \text{-}, nil)$ with

$//\qquad hOut \not\rightarrow hRep \wedge hOut \not\rightarrow (y \rightarrow \text{-}, nil)$

...

$//heap = hOut * (x \rightarrow \text{-}, y) * hRep * (y \rightarrow \text{-}, b) *$ with

$//\qquad hOut \not\rightarrow hRep \wedge hOut \not\rightarrow (y \rightarrow \text{-}, nil)$

Useful for tools that treat types as predicates in an untyped logic?

Sidelights

- Established theories of parametricity using existential types: apply to instance-based visibility [Red98] [elaborating Reynolds]

Sidelights

- Established theories of parametricity using existential types: apply to instance-based visibility [Red98] [elaborating Reynolds]
- Semantics of specification statements: sets-of-state-transformers corresponds to conjunctive predicate transformers [YR00, Nau01a] [Leino&Manohar]

References

[BN] Anindya Banerjee and David A. Naumann. A static analysis for instance-based confinement in Java. Submitted.

[BN02] Anindya Banerjee and David A. Naumann. Representation independence, confinement and access control. In *POPL*, 2002. to appear.

[BS01] P. H. M. Borba and A. C. A. Sampaio. A normal form reduction strategy for ROOL: an object-oriented language. Submitted to ESOP, 2001.

[CN01] Ana Cavalcanti and David A. Naumann. Forward simulation for data refinement of classes. submitted, 2001.

[Nau01a] David A. Naumann. Calculating sharp adaptation rules. *Information Processing Letters*, 77:201–208, 2001.

[Nau01b] David A. Naumann. Soundness of data refinement for a higher order imperative language. *Theoretical Computer Science*, 2001. To appear.

[Red98] U. S. Reddy. Objects and classes in Algol-like languages. In *Fifth Intern. Workshop on Foundations of Object-oriented Languages*, Jan 1998. Full version to appear in Information and Computation.

[YR00] H. Yang and U. S. Reddy. On the semantics of refinement calculi. In *Foundations of Software Science and Computation Structures*, volume 1784, pages 359–374. Springer-Verlag, 2000.

Appendix: static confinement

Signatures: $C \leq Rep \Rightarrow U \leq A \vee U \leq Rep$ for all $U \in \overline{T}$

Phrases:

$$\frac{C \leq A \Rightarrow U \not\leq A \qquad \Gamma;\ C \rhd e : U}{\Gamma;\ C \rhd x.f := e} \qquad \frac{C \neq A \Rightarrow B \not\leq Rep \qquad C \leq A \Rightarrow B \not\leq A}{\Gamma;\ C \rhd x := \mathsf{new}\ B(\ )}$$

$$\frac{\mathrm{mtype}(m, D) = (\overline{x} : \overline{T}) \to T \qquad C \leq A \Rightarrow T \not\leq A \qquad \Gamma;\ C \rhd e : D \quad \Gamma;\ C \rhd \overline{e} : \overline{U}}{\Gamma;\ C \rhd e.m(\overline{e}) : T}$$

These suffice for semantic condition stronger than needed for abstraction theorem.