A Formal Semantics for Isorecursive and Equirecursive State Abstractions

Alex Summers
ETH Zurich, Switzerland

Sophia Drossopoulou
Imperial College London, UK

Background / Area

- Specification/Verification logics for heap-based programming languages
- Integration of permission-based verification with general recursive definitions (predicates, functions, etc.)
- Many verification approaches make use of some kind of *permission*, governing access to the heap
  - e.g. separation logics, implicit dynamic frames,
- Need to specify recursive datatypes commonly leads to logics which support recursive definitions
  - e.g. abstract predicates, pure methods/functions

Implicit Dynamic Frames (IDF)

Smans et al.'08

- Extend first-order assertions to additionally include "accessibility predicates": **`acc(x.f)`**
  - meaning: *permission* to access location **`x.f`**
- Assertions can also include heap-dependent expressions: e.g., **`x.f > 4`**
  - only allowed when permission to location is held
- Also includes a separating conjunction **`*`**
  - related to that of separation logic; expresses disjointness of permissions, *but* allows heap reads : **`acc(x.f) * (x.f ≠ null ⇒ acc(x.f.f))`**
- *self-framing assertions* are those which require permission to all heap locations they depend on

Recursive Predicates

- Recursive predicates may specify permissions to unbounded data structures

```
predicate list  ≡
    acc(this.val) * acc(this.next) *
    (this.next ≠ null ⇒ this.next.list)
```

- Predicate bodies are assertions; they describe permissions and other properties (eg `this.val ≠ 4`)
- Predicate *instances* are written e.g. `x.list` and can be used in assertions

Abstraction Functions

- Implicit Dynamic Frames (IDF) also provides *abstraction functions* in its expression syntax
  - similar to allowing pure methods in specifications
- User-defined functions can abstract over data

```
function length() : int
  { next == null ? 1 : 1 + next.length() }
```

  - Function body is a (heap-dependent) expression
  - Functions can have assertions as pre-conditions
- We give a semantics to IDF recursive definitions

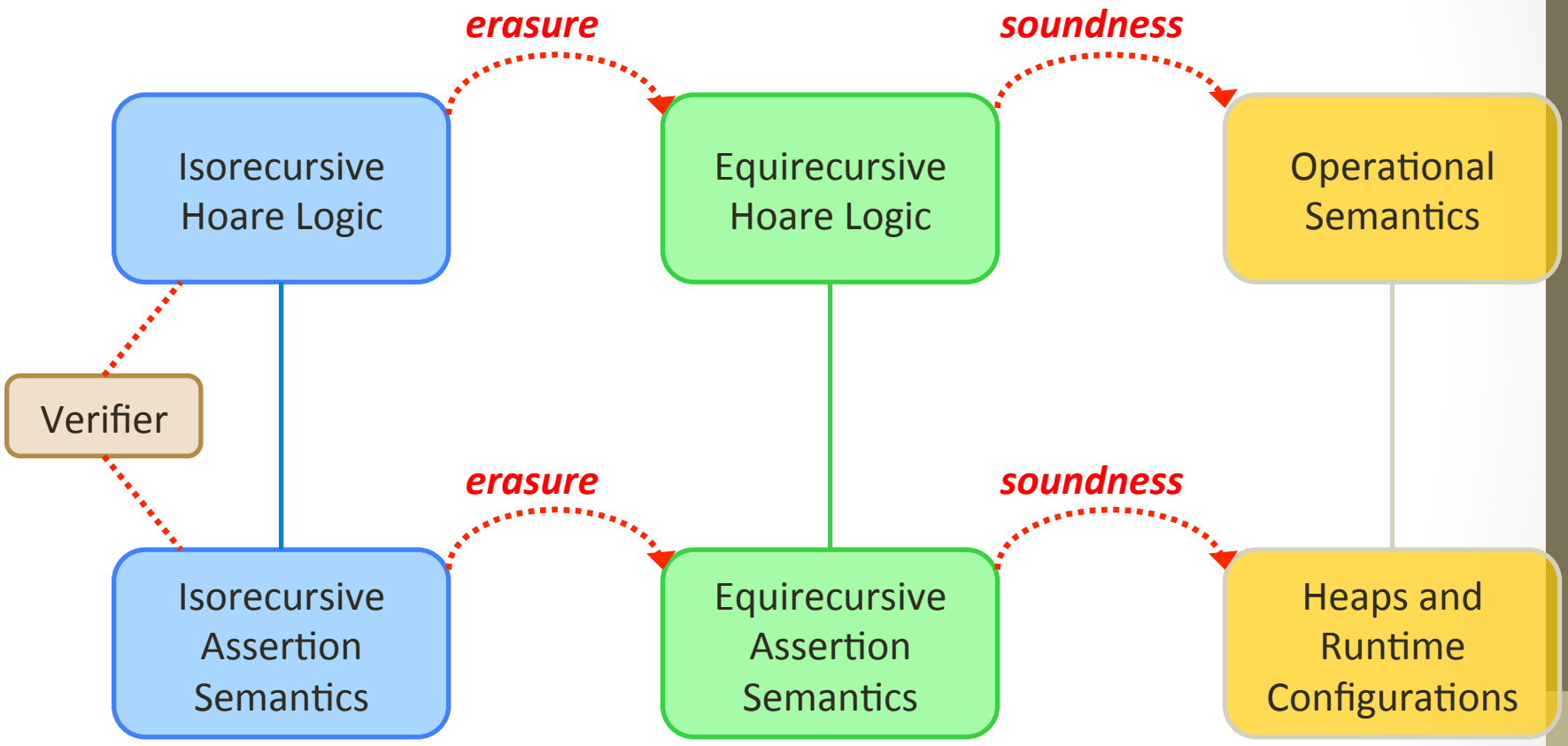
Recursive definitions: two views

- For recursive definitions, such as the predicate **`list`**, we consider two different interpretations:
- In the *equirecursive interpretation*, a predicate is identified with its body
  - holding an instance of a predicate is "the same" as holding the contents "all the way down"
- In the *isorecursive interpretation*, a predicate and its body are differentiated from one another
  - However, the two can be explicitly exchanged
  - For predicates, this is typically achieved with ghost *fold* and *unfold* steps in a program

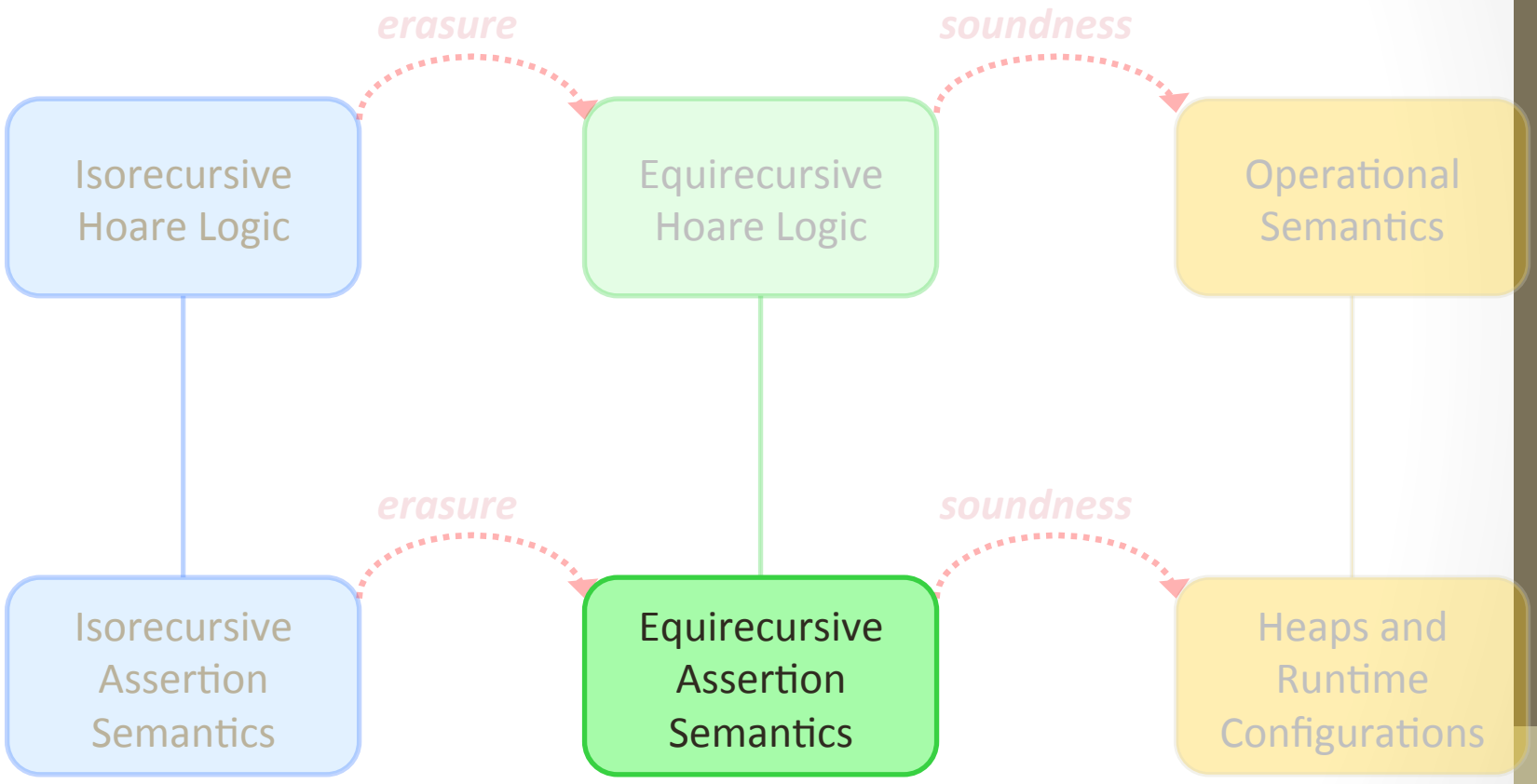
Isorecursive vs Equirecursive

- The *equirecursive* approach is mathematically simple, and easy to relate to a runtime semantics
  - typically used for formalisations of verifiers etc.
- ...not suitable for implementing a static verifier
  - general handling of recursion not automatable
- The *isorecursive* approach is used in many tools
  - predicates folded/unfolded to certain depths
  - function evaluation limited to certain depths
- This paper addresses this mismatch formally
  - handling isorecursion can be subtle; a prototype encoding of functions in Chalice tool was unsound

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
Verifier
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Equirecursive Assertions

- e ::= null | x | e.f | e.g(e) | e=e | (e ? e : e)
- a ::= e | acc(e.f) | e ⇒ a | a*a | e.P
- We seek to define a judgement $H, \Pi, \sigma \vDash_E a$ specifying the equirecursive semantics of assertion a
  - H is a heap (map from field locations to values)
  - Π is a permission mask (set of field locations)
  - σ is an environment (maps variables to values)
- [Parkinson/Summers'12] gives simple cases:
  - $H, \Pi, \sigma \vDash_E a_1 * a_2 \Leftrightarrow \exists\ \Pi_1, \Pi_2 :$
    $H, \Pi_1, \sigma \vDash_E a_1\ \&\ H, \Pi_2, \sigma \vDash_E a_2$
  - Permissions are split across *

Equirecursive Assertions (2)

- What about heap-dependent expressions? x.f > 5
  - In general, reading from the heap without permission might not be meaningful (races)
- We can't locally see whether permission is held
  - even in an assertion such as acc(x.f) * x.f > 5
- **1st Attempt:**
  - expressions undefined, unless overall expression holds sufficient permissions
  - Distinguish global from local permissions

Equirecursive Assertions (3)

- What about heap-dependent expressions? x.f > 5
  - In general, reading from the heap without permission might not be meaningful (races)
- We can't *locally* see whether permission is held
  - even in an assertion such as acc(x.f) * x.f > 5
- **Solution**:
    - expressions get "optimistic" semantics
    - A separate judgement H, Π, σ $\vDash_{frmE}$ a checks that an assertion is *framed* in a state (i.e. only reads locations to which permission is held)
    - Only *self-framing* assertions used in contracts
    - overall assertion will thus be *false* in "bad" cases

Non-terminating predicates?

- Equirecursive semantics for predicates, given by:

$$H, \Pi, \sigma \vDash_E x.P \quad \Leftrightarrow \quad H, \Pi, \sigma \vDash_E \mathit{Body}(x.P)$$

- But what about

```
predicate cyclic ≡ this.cyclic
```

?

- Or, even (previous example)

```
predicate list ≡
    acc(this.val) * acc(this.next)
    *(this.next ≠ null ⇒ this.next.list)
```

when we have a cycle: `this.next == this` ?

- **Three Possible Approaches:**
  1. Predicate undefined
  2. Predicate holds (greatest fixed point)
  3. Predicate does not hold (least fixed point)

Non-terminating predicates? (2)

- Equirecursive semantics for predicates, given by:

$$H, \Pi, \sigma \vDash_E x.P \quad \Leftrightarrow \quad H, \Pi, \sigma \vDash_E \mathit{Body}(x.P)$$

- But what about

```
predicate cyclic ≡ this.cyclic
```

?

- Or, even (previous example)

```
predicate list ≡
    acc(this.val) * acc(this.next)
    *(this.next ≠ null ⇒ this.next.list)
```

when we have a cycle: `this.next == this` ?

- **Our Solution:** Predicate does not hold.
  - We interpret predicate definitions as their least fixed-points (an infinite instance means **`false`**)
  - For verifiers this is not a restriction (describes unreachable code)
  - This also helps with functions, as shown next...

Non-terminating functions?

- Unrestricted function definitions are dangerous

```
function bad(): int { return 1 + bad() }
```

  - encoding this to prover as a function is unsound
  - typical approach: insist on function termination – this guarantees existence of mathematical function
- Also, what about

```
function length() : int
  { next==null ? 1 : 1 + next.length() }
```

  when `this.next == this` ?

Non-terminating functions? (2)

- Unrestricted function definitions are dangerous

```
function bad(): int { return 1 + bad() }
```

- Also, what about

```
function length() : int
  { next==null ? 1 : 1 + next.length() }
```

when `this.next == this`?

- **Three Possible Approaches**
  1. Undefined
  2. A fixed value from the underlying type, eg `1` for `int`.
  3. An `error` value for each underlying type.

Non-terminating functions? (3)

- Unrestricted function definitions are dangerous

```
function bad(): int { return 1 + bad() }
```

- Also, what about

```
function length() : int
  { next==null ? 1 : 1 + next.length() }
```

when `this.next == this`?

- **Our Solution**

3. An `error` value for each underlying type.

- Note difference in treatment of non-terminating expressions and predicates

Error values for total semantics

- We define naïve expression semantics – regardless of access permissions - by a judgement $e \Downarrow_{H,\sigma} v$.
- $e \Downarrow_{H,\sigma} v$ is partial.
- We add (to each type) a special value *error*
- We define our actual semantics $\|e\|_{H,\sigma}$ by:
  - $\|e\|_{H,\sigma} = v$ *if* $e \Downarrow_{H,\sigma} v$
  - $\|e\|_{H,\sigma} = error$ *if* $\neg\exists\, v\ (\, e \Downarrow_{H,\sigma} v\,)$
- $\|e\|_{H,\sigma}$ is total.
- As an *assertion* (not sub-expression) we treat *error* as we do falsity

Equirecursive Semantics - concretely

Values of Equi-expressions $e \Downarrow_{H,\sigma} v$

- $x \Downarrow_{H,\sigma} \sigma(x)$
- $\text{true} \Downarrow_{H,\sigma}$ …
- $e.f \Downarrow_{H,\sigma} v$ if …
- $e.g(e') \Downarrow_{H,\sigma} v$ if …
- $e = e' \Downarrow_{H,\sigma} \text{true}$ if …
- $e = e' \Downarrow_{H,\sigma} \text{false}$ if …

$\text{null} \Downarrow_{H,\sigma}$ …

$\text{false} \Downarrow_{H,\sigma}$ …

Remember:

We define naïve expression semantics – regardless of access permissions - by a judgement $e \Downarrow_{H,\sigma} v$ (not a total function of e)

Values of Equi-expressions $e \Downarrow_{H,\sigma} v$ (2)

- $x \Downarrow_{H,\sigma} \sigma(x)$
- $\text{true} \Downarrow_{H,\sigma} \text{true}$
- $e.f \Downarrow_{H,\sigma} v$ if $e \Downarrow_{H,\sigma} \iota$ & $H(\iota,f)=v$
- $e.g(e') \Downarrow_{H,\sigma} v$ if $e \Downarrow_{H,\sigma} \iota$ & $e' \Downarrow_{H,\sigma} v'$ & $\text{Body}(g) \Downarrow_{H,\sigma'} v$ where $\sigma'(\text{this})=\iota$, $\sigma'(x)=v'$
- $e = e' \Downarrow_{H,\sigma} \text{true}$ if $e \Downarrow_{H,\sigma} v$ & $e' \Downarrow_{H,\sigma} v$ for some v
  $e = e' \Downarrow_{H,\sigma} \text{false}$ if $e \Downarrow_{H,\sigma} v$ & $e' \Downarrow_{H,\sigma} v'$ for some $v \neq v'$

- $\text{null} \Downarrow_{H,\sigma} \text{null}$
- $\text{false} \Downarrow_{H,\sigma} \text{false}$

Values of Equi-expressions $e \Downarrow_{H,\sigma} v$ (2)

- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if ???
- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if ???

- $\|e\|_{H,\sigma} =$ … if ???

- **Remember:** We add an error value to each type, and make $\|e\|_{H,\sigma}$ a total function.

Values of Equi-expressions $e \Downarrow_{H,\sigma} v$ (3)

- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if $e1 \Downarrow_{H,\sigma} true$ and $e2 \Downarrow_{H,\sigma} v$
- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if $e1 \Downarrow_{H,\sigma} false$ and $e2 \Downarrow_{H,\sigma} v$

- $\|e\|_{H,\sigma} = v$ if $e \Downarrow_{H,\sigma} v$
- $\|e\|_{H,\sigma} = error$ if $\neg\exists\, v\ (e \Downarrow_{H,\sigma} v)$

Therefore, if z points to a cycle in H, σ, then

$\|z.length()\|_{H,\sigma} = ???$

$\|z.length()=3\|_{H,\sigma} = ???$

$\|z.length()=3?false:true\|_{H,\sigma} = ???$

Values of Equi-expressions $e \Downarrow_{H,\sigma} v$ (4)

- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if $e1 \Downarrow_{H,\sigma} true$ & $e2 \Downarrow_{H,\sigma} v$
- $(e1?e2:e3) \Downarrow_{H,\sigma}$ if $e1 \Downarrow_{H,\sigma} false$ & $e2 \Downarrow_{H,\sigma} v$
- $\|e\|_{H,\sigma} = v$ if $e \Downarrow_{H,\sigma} v$
- $\|e\|_{H,\sigma} = error$ if $\neg \exists\, v\ (e \Downarrow_{H,\sigma} v)$

Note that value of an expression always defined.

Therefore, if z points to a cycle in H, σ, then

- $\|z.length()\|_{H,\sigma} =$ error
- $\|z.length()=3\|_{H,\sigma} =$ error
- $\|z.length()=3?false:true\|_{H,\sigma} =$ error

Semantics of Equi-assertions, $H,\Pi,\sigma \vDash_E a$

- $H, \Pi, \sigma \vDash_E e \quad \Leftrightarrow \quad \ldots$
- $H, \Pi, \sigma \vDash_E acc(e.f) \quad \Leftrightarrow \quad \ldots.$
- $H, \Pi, \sigma \vDash_E e \rightarrow a \quad \Leftrightarrow \quad \ldots.$
- $H, \Pi, \sigma \vDash_E a1 * a2 \quad \Leftrightarrow \quad \ldots.$
- $H, \Pi, \sigma \vDash_E e.P \quad \Leftrightarrow \quad \ldots.$

Semantics of Equi-assertions, $H,\Pi,\sigma \vDash_E a$ (2)

- $H, \Pi, \sigma \vDash_E e \quad \Leftrightarrow \quad e \Downarrow_{H,\sigma} true$
- $H, \Pi, \sigma \vDash_E acc(e.f) \quad \Leftrightarrow \quad (\|e\|_{H,\sigma,}, f) \in \Pi$
- $H, \Pi, \sigma \vDash_E e \rightarrow a \quad \Leftrightarrow \quad e \Downarrow_{H,\sigma\ \sigma} true \Rightarrow H, \Pi, \sigma \vDash_E a$
- $H, \Pi, \sigma \vDash_E a1*a2 \quad \Leftrightarrow \quad H, \Pi1, \sigma \vDash_E a1 \ \& \ H, \Pi2, \sigma \vDash_E a2 \ \& \ \Pi = \Pi1*\Pi2$
- $H, \Pi, \sigma \vDash_E e.P \quad \Leftrightarrow \quad H, \Pi, \sigma \vDash_E Body(P)[e/this]$

Therefore, if z points to a cycle, then

- $H, \Pi, \sigma \ ?\vDash_E ?$ z.length()=3
- $H, \Pi, \sigma \ ?\vDash_E ?$ (z.length()=3? true : false)
- $H, \Pi, \sigma \ ?\vDash_E ?$ (z.length()=3 -> false)

Semantics of Equi-assertions, $H,\Pi,\sigma \vDash_E a$ (3)

- $H, \Pi, \sigma \vDash_E e \quad \Leftrightarrow \quad e \Downarrow_{H,\sigma} true$
- $H, \Pi, \sigma \vDash_E acc(e.f,q) \quad \Leftrightarrow \quad (\|e\|_{H,\sigma},f) \in \Pi$
- $H, \Pi, \sigma \vDash_E e \text{ -> } a \quad \Leftrightarrow \quad e \Downarrow_{H,\sigma} true \Rightarrow H, \Pi, \sigma \vDash_E a$
- $H, \Pi, \sigma \vDash_E a1*a2 \quad \Leftrightarrow \quad H, \Pi1, \sigma \vDash_E a1 \ \& \ H, \Pi2, \sigma \vDash_E a2 \ \& \ \Pi = \Pi1*\Pi2$
- $H, \Pi, \sigma \vDash_E e.P \quad \Leftrightarrow \quad H, \Pi^{(2)}, \sigma \vDash_E Body(P)[e/this]$

Therefore, if z points to a cycle, then

- $H, \Pi, \sigma \nvDash_E z.length()=3$
- $H, \Pi, \sigma \vDash_E (z.length()=3? true : false)$
- $H, \Pi, \sigma \vDash_E (z.length()=3 \text{ -> } false)$

This may feel disturbing … until we consider framing.

Framing: H, Π, σ $\models_{frmE}$ e

- H, Π, σ $\models_{frmE}$ x
- H, Π, σ $\models_{frmE}$ true
- H, Π, σ $\models_{frmE}$ e.f if …
- H, Π, σ $\models_{frmE}$ e.g(e') if …
- H, Π, σ $\models_{frmE}$ e = e' if …
- H, Π, σ $\models_{frmE}$ e1?e2:e3 if …
- H, Π, σ $\models_{frmE}$ null
- H, Π, σ $\models_{frmE}$ false

H, Π, σ $\models_{frmE}$ e holds if Π contains permissions for all heap accesses required for the evaluation of e (ie for e $\Downarrow_{H,\sigma}$)

Framing: $H, \Pi, \sigma \vDash_{frmE} e$ (2)

- $H, \Pi, \sigma \vDash_{frmE} x$
- $H, \Pi, \sigma \vDash_{frmE} true$
- $H, \Pi, \sigma \vDash_{frmE} e.f$
- $H, \Pi, \sigma \vDash_{frmE} e.g(e')$
- $H, \Pi, \sigma \vDash_{frmE} e = e'$
- $H, \Pi, \sigma \vDash_{frmE} e1?e2:e3$

- $H, \Pi, \sigma \vDash_{frmE} null$
- $H, \Pi, \sigma \vDash_{frmE} false$

if $H, \Pi, \sigma \vDash_{frmE} e$ & $(\|e\|_{H,\sigma,}, f) \in \Pi$

if $H, \Pi, \sigma \vDash_{frmE} e$ & $H, \Pi, \sigma \vDash_{frmE} e'$ & $H, \Pi, \sigma' \vDash_{frmE} Body(g)$ & $\sigma'(this) = \|e\|_{H,\sigma}$ & $\sigma'(x) = \|e'\|_{H,\sigma}$

if $H, \Pi, \sigma \vDash_{frmE} e$ & $H, \Pi, \sigma \vDash_{frmE} e'$

if $H, \Pi, \sigma \vDash_{frmE} e1$ & ($H, \Pi, \sigma \vDash_{E} e1 \Rightarrow H, \Pi, \sigma \vDash_{frmE} e2$) & ($H, \Pi, \sigma \nvDash_{E} e1 \Rightarrow H, \Pi, \sigma \vDash_{frmE} e3$)

$H, \Pi, \sigma \vDash_{frmE} e$ holds if Π contains permissions for all heap accesses required for the evaluation of e (ie for $e \Downarrow_{H,\sigma}$)

Framing: $H, \Pi, \sigma \vDash_{frmE} a$

- $H, \Pi, \sigma \vDash_{frmE} a$ if Π contains permissions for all heap accesses required to determine validity of a (ie for $H, \Pi, \sigma \vDash_{E} a$)
- $H, \Pi, \sigma \vDash_{frmE} acc(e.f, q)$ if ….
- $H, \Pi, \sigma \vDash_{frmE} e \rightarrow a$ if …
- $H, \Pi, \sigma \vDash_{frmE} a1 * a2$ if …
- $H, \Pi, \sigma \vDash_{frmE} e.P$ if …
- a **frames** e, ie $a \vDash_{frmE} e$, if a has permissions for all heap accesses required to evaluate e
- $a \vDash_{frmE} e$ if …
- a is **self-framing**, ie $\vDash_{frmE} a$, if it requires permissions for all heap accesses required to determine whether it holds.
- $\vDash_{frmE} a$ if …

Framing: $H, \Pi, \sigma \vDash_{frmE} a$ (2)

- $H, \Pi, \sigma \vDash_{frmE} a$ if Π contains permissions for all heap accesses required to determine validity of a (ie for $H, \Pi, \sigma \vDash_{E} a$)
- $H, \Pi, \sigma \vDash_{frmE} acc(e.f)$ if $H, \Pi, \sigma \vDash_{frmE} e$
- $H, \Pi, \sigma \vDash_{frmE} e \rightarrow a$ if $H, \Pi, \sigma \vDash_{frmE} e$ & $(H, \Pi, \sigma \vDash_{E} e \Rightarrow H, \Pi, \sigma \vDash_{E} a)$
- $H, \Pi, \sigma \vDash_{frmE} a1 * a2$ if $H, \Pi1, \sigma \vDash_{frmE} a1$ & $H, \Pi2, \sigma \vDash_{frmE} a2$ & $\Pi = \Pi1 * \Pi2$
- $H, \Pi, \sigma \vDash_{frmE} e.P$ if $H, \Pi, \sigma \vDash_{frmE} e$ & $H, \Pi, \sigma \vDash_{frmE} Body(P)[e/this]$
- a **frames** e, ie $a \vDash_{frmE} e$, if a has permissions for all heap accesses required to evaluate e
- $a \vDash_{frmE} e$ if $H, \Pi, \sigma \vDash a \Rightarrow H, \Pi, \sigma \vDash_{frmE} e$
- a is **self-framing**, ie $\vDash_{frmE} a$, if it requires permissions for all heap accesses required to determine whether it holds.
- $\vDash_{frmE} a$ if $H, \Pi, \sigma \vDash a \Rightarrow H, \Pi, \sigma \vDash_{frmE} a$

Framing: H, Π, σ $\vDash_{frmE}$ a (3)

- H, Π, σ $\vDash_{frmE}$ acc(e.f, q) if H, Π, σ $\vDash_{frmE}$ e
- H, Π, σ $\vDash_{frmE}$ e -> a if H, Π, σ $\vDash_{frmE}$ e & (H, Π, σ $\vDash_{E}$ e ⇒ H, Π, σ $\vDash_{E}$ a)
- H, Π, σ $\vDash_{frmE}$ a1 * a2 if H, Π1, σ $\vDash_{frmE}$ a1 & H, Π2, σ $\vDash_{frmE}$ a2 & Π= Π1 *Π2
- H, Π, σ $\vDash_{frmE}$ e.P if H, Π, σ $\vDash_{frmE}$ e & H, Π, σ $\vDash_{frmE}$ Body(P)[e/this]
- a is self-framing, $\vDash_{frmE}$ a if H, Π, σ ⊨ a ⇒ H, Π, σ $\vDash_{frmE}$ a

Thus, are the following assertions self-framing?

- x.next == null …
- acc(x.next) …
- x.next=x.next …
- x.next≠x.next …
- x.list …
- x.list * x.lengh=3 …
- x.list -> x.lengh=3 …

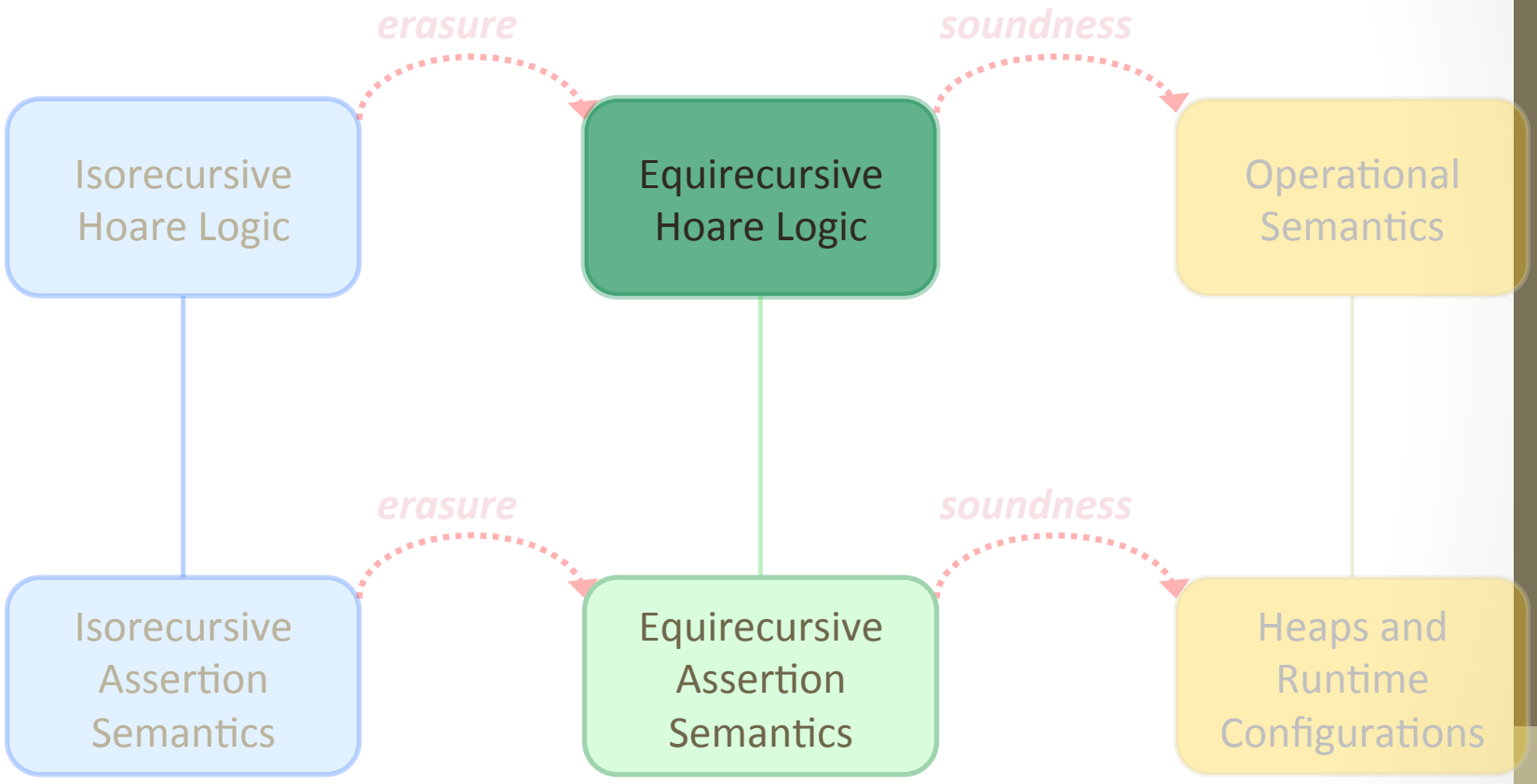
Framing: H, Π, σ $\vDash_{frmE}$ a (4)

- H, Π, σ $\vDash_{frmE}$ acc(e.f) if H, Π, σ $\vDash_{frmE}$ e
- H, Π, σ $\vDash_{frmE}$ e -> a if H, Π, σ $\vDash_{frmE}$ e & (H, Π, σ $\vDash_{E}$ e ⇒ H, Π, σ $\vDash_{E}$ a)
- H, Π, σ $\vDash_{frmE}$ a1 * a2 if H, Π1, σ $\vDash_{frmE}$ a1 & H, Π2, σ $\vDash_{frmE}$ a2 & Π= Π1 *Π2
- H, Π, σ $\vDash_{frmE}$ e.P if H, Π, σ $\vDash_{frmE}$ e & H, Π, σ $\vDash_{frmE}$ Body(P)[e/this]
- a $\vDash_{frmE}$ e if H, Π, σ ⊨ a ⇒ H, Π, σ $\vDash_{frmE}$ e
- a is self-framing, $\vDash_{frmE}$ a if H, Π, σ ⊨ a ⇒ H, Π, σ $\vDash_{frmE}$ a

Thus, are the following assertions self-framing?

- x.next == null not self-framing
- acc(x.next) self-framing
- x.next=x.next not self-framing
- x.next≠x.next not self-framing
- x.list not self-framing
- x.list * x.lengh=3 self-framing
- x.list -> x.lengh=3 badly formed

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Equi-recursive Hoare Logic

- Hoare triples: $\vdash_E \{a_1\}\ s\ \{a_2\}$
- Implicitly require pre/post-conditions to be self-framing (in the equi-recursive sense, ie $\vdash_{frmE} a$)

- Some rules are standard, e.g. “skip” statement:

$$\frac{}{\vdash_E \{\ldots\}\ \texttt{skip}\ \{\ldots.\}}\ (skipE)$$

- Usual rule of consequence, using equi-entailment:

$$\frac{\ldots. \qquad \ldots.}{\vdash_E \{a_1\}\ s\ \{a_2\}}\ (consE)$$

Equi-recursive Hoare Logic (2)

- Hoare triples: $\vdash_E \{a_1\}\ s\ \{a_2\}$
- Implicitly require pre/post-conditions to be self-framing (in the equi-recursive sense, ie $\vdash_{frmE} a$)

- Some rules are standard, e.g. “skip” statement:

$$\frac{}{\vdash_E \{a\}\ \texttt{skip}\ \{a\}}\ (skipE)$$

- Usual rule of consequence, using equi-entailment:

$$\frac{a_1 \vDash_E a_3 \quad \vdash_E \{a_3\}\ s\ \{a_4\} \quad a_4 \vDash_E a_2}{\vdash_E \{a_1\}\ s\ \{a_2\}}\ (consE)$$

Equi-recursive Hoare Logic (3)

- The assignment rule

$$\frac{\ldots.}{\vdash_E \{\ldots.\}\ \mathtt{x:=e}\ \{a\}}\ (\mathit{varAssE})$$

- Field Assignment:

$$\frac{\ldots.}{\vdash_E \{\ldots\}\ \mathtt{x.f:=y}\ \{\ldots.\}}\ (\mathit{fldAssE})$$

- Sequence:

$$\frac{\ldots. \qquad \ldots}{\vdash_E \{a\}\ \mathtt{s}\ \{a''\}}\ (\mathit{seqE})$$

- Frame:

$$\frac{\ldots \qquad \ldots}{\vdash_E \{a*a1\}\ \mathtt{s}\ \{a*a2\}}\ (\mathit{frameE})$$

Equi-recursive Hoare Logic (4)

- The assignment rule

$$\frac{a[e/x] \vDash_{frmE} e}{\vdash_E \{a[e/x]\}\ \mathbf{x := e}\ \{a\}}\ (varAssE)$$

- Field Assignment:

$$\frac{}{\vdash_E \{x \neq null * acc(x,f)\}\ \mathbf{x.f := y}\ \{acc(x,f) * x.f = y\}}\ (fldAssE)$$

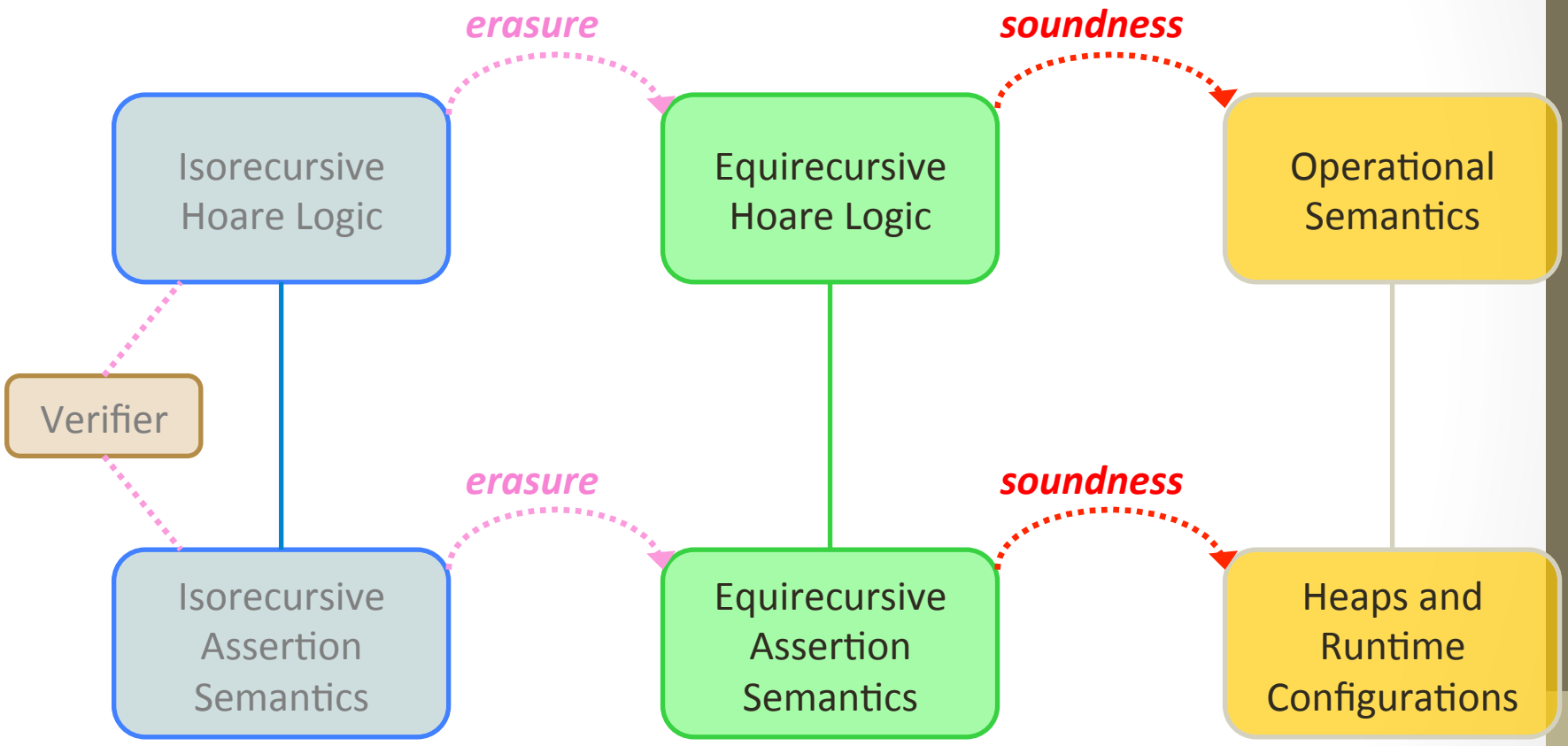
- Sequence:

$$\frac{\vdash_E \{a\}\ \mathbf{s}\ \{a'\} \qquad \vdash_E \{a'\}\ \mathbf{s'}\ \{a''\}}{\vdash_E \{a\}\ \mathbf{s}\ \{a''\}}\ (seqE)$$

- Frame:

$$\frac{\vdash_E \{a1\}\ \mathbf{s}\ \{a2\} \quad mods(s) \cap FV(a) = \varnothing \qquad \vDash_{frmE} a}{\vdash_E \{a * a1\}\ \mathbf{s}\ \{a * a2\}}\ (frameE)$$

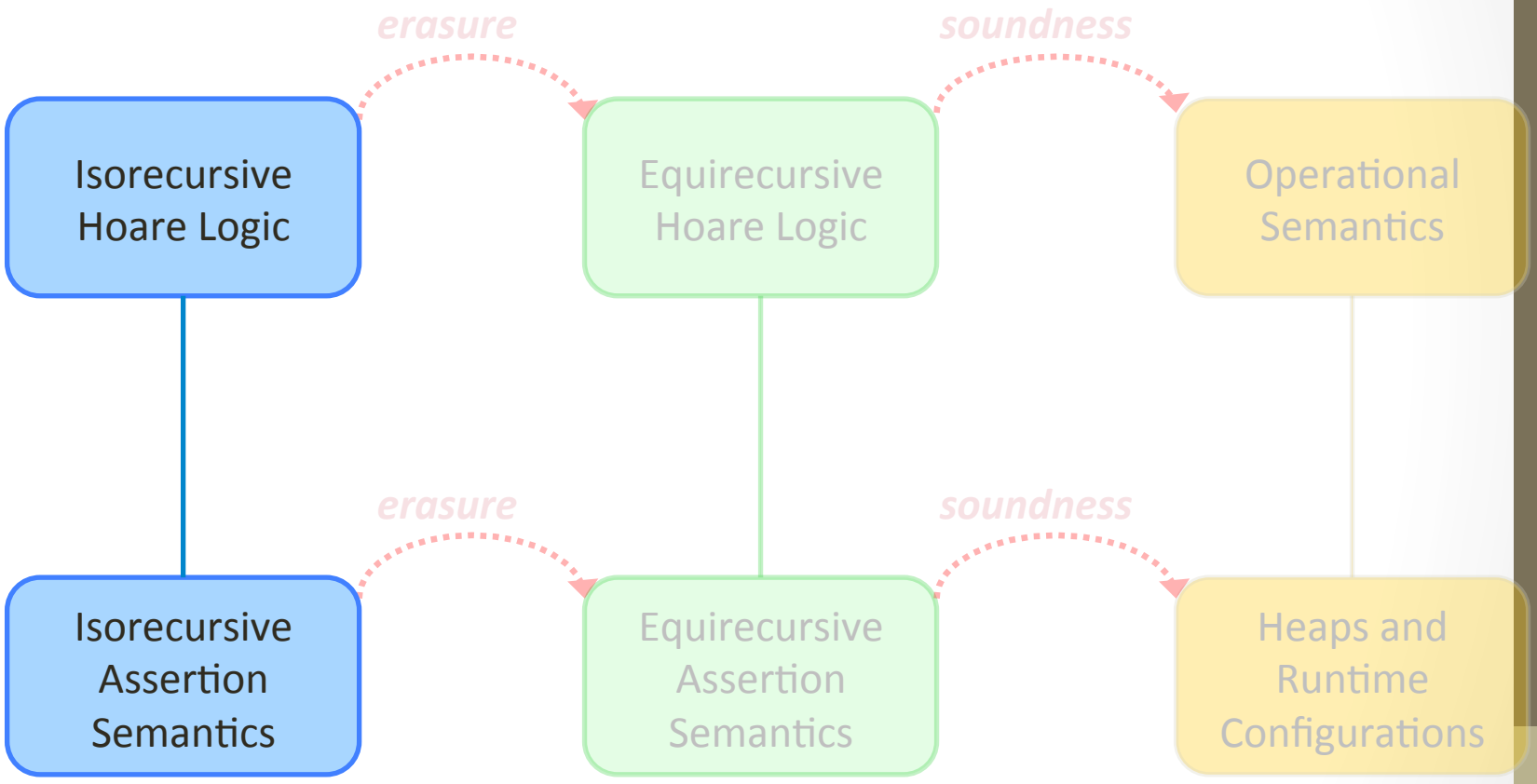
Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
Verifier
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Soundness - summary

- We extend assertions to talk about threads
- We define operational semantics for concurrency, and fork-join – some novel approaches
- We prove soundness of equi-Hoare logic wrt operational semantics

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Isorecursive Semantics

- Isorecursive assertions differ in two ways:
  - Instead of predicate instances x.P, we have only *permissions* to predicates, written acc(x.P)
  - We add assertions of the form *unfolding* x.P *in* B
- Functions and Predicates have Pre-conditions
- Having permission to a predicate does *not* permit directly using its contents
  - these can be accessed by fold/unfold statements
  - or, in assertions, using *unfolding* (temporary unfold)
- Notion of *(self-)framing* differs correspondingly
  - *acc*(x.list)* x.next == null ✘
  - *acc*(x.list)*(*unfolding* x.list *in* x.next == null) ✓

Well-formed definitions

- Function definitions must satisfy conditions:
  - $f_{pre} \vDash_{frml} f_{body}$ (body framed if precondition true)
  - $H, \Pi, \sigma \vDash_{I} f_{pre} \Rightarrow \| f_{body} \|_{H,\sigma} \neq \mathit{error}$ (termination)
- Iso-recursive version of "length" function:

```
function length() : int
    requires this.list {
        unfolding this.list in
         next == null ? 1 : 1 + next.length()
 }
```

- The function length is well-defined accord to the criteria from above.
- Note that checking above criteria does *not* imply the need to actually evaluate $\| f_{body} \|_{H,\sigma}$
  - any termination proof is sufficient (unspecified)

Isorecursive Hoare Logic

- Derivation rules for Hoare triples: $\vdash_I \{A_1\}\ s\ \{A_2\}$
  - we implicitly require the pre/post-conditions to be self-framing (in the iso-recursive sense)
- Some rules are standard, e.g. "skip" statement:

$$\frac{}{\vdash_I \{\ldots\}\ \texttt{skip}\ \{\ldots\}}\ (\textit{skipI})$$

- Usual rule of consequence, using iso-entailment:

$$\frac{\ldots\ldots}{\vdash_I \{A_1\}\ s\ \{A_2\}}\ (\textit{consI})$$

Isorecursive Hoare Logic (2)

- Derivation rules for Hoare triples: $\vdash_I \{A_1\}\ s\ \{A_2\}$
  - we implicitly require the pre/post-conditions to be self-framing (in the iso-recursive sense)
- Some rules are standard, e.g. "skip" statement:

$$\frac{}{\vdash_I \{A\}\ \texttt{skip}\ \{A\}}\ (\textit{skipI})$$

- Usual rule of consequence, using iso-entailment:

$$\frac{A_1 \vDash_I A_3 \quad \vdash_I \{A_3\}\ s\ \{A_4\} \quad A_4 \vDash_I A_2}{\vdash_I \{A_1\}\ s\ \{A_2\}}\ (\textit{consI})$$

Isorecursive Hoare Logic (3)

- Some rules need slightly careful treatment:
- We must check that the program only reads locations to which permissions are held:

$$\frac{\cdots\cdots}{\vdash_I \{A[E/x]\}\ x := E\ \{A\}}\ (assI)$$

- *Frame rule*, based on that of separation logic, but only for self-framing additional assertions:

$$\frac{\cdots \quad \cdots \qquad mods(s) \cap FV(A_3)=\varnothing}{\vdash_I \{A_1 * A_3\}\ s\ \{A_2 * A_3\}}\ (frameI)$$

Isorecursive Hoare Logic (3)

- Some rules need slightly careful treatment:
- We must check that the program only reads locations to which permissions are held:

$$\frac{A[E/x] \vDash_{frmI} E}{\vdash_I \{A[E/x]\}\ x := E\ \{A\}}\ (assI)$$

- *Frame rule*, based on that of separation logic, but only for self-framing additional assertions:

$$\frac{\vDash_{frmI} A_3 \qquad \vdash_I \{A_1\}\ s\ \{A_2\} \qquad mods(s) \cap FV(A_3) = \varnothing}{\vdash_I \{A_1 * A_3\}\ s\ \{A_2 * A_3\}}\ (frameI)$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule

state

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule

state

$\vdash_I \{Body(\texttt{x.P})\}\ \texttt{fold x.P}\ \{\quad acc(\text{x.P})\quad \}$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule

state

$\vdash_I \{Body(\mathtt{x.P})\}\ \mathtt{fold\ x.P}\ \{\quad acc(\mathrm{x.P})\quad \}$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule

state

$$\frac{A' = Body(\mathtt{x.P})}{\vdash_I \{\ A'\ \}\ \mathtt{fold\ x.P}\ \{\ acc(x.P)\ \}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A'\ \}\ \mathbf{fold}\ \mathbf{x.P}\ \{\ \mathit{acc}(x.P)\ \}}$$

Isorecursive "fold" statement

- For "fold", we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- *For example, if we knew* **`x.val=3`** *before fold,*

how do we preserve this information after folding **`x.list`** *?*

$$\frac{A' = Body(\mathbf{x.P})}{\vdash_I \{\ A'\ \}\ \mathbf{fold\ x.P}\ \{\ acc(x.P)\ \}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A'\ \}\ \mathbf{fold}\ \mathbf{x.P}\ \{\ \mathit{acc}(x.P)\ \}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A'\ \}\ \mathbf{fold\ x.P}\ \{\ \mathit{acc}(x.P)\ \}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A' * B\ \}\ \mathbf{fold}\ \mathbf{x.P}\ \{\ \mathit{acc}(x.P)\ \}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A' * B\ \}\ \mathbf{fold}\ \mathbf{x.P}\ \{\ \mathit{acc}(x.P) * (\mathit{unfolding}\ x.P\ \mathit{in}\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A' * B\ \}\ \mathbf{fold\ x.P}\ \{\ \mathit{acc}(x.P) * (\mathit{unfolding}\ x.P\ \mathit{in}\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here

what if B *depends on other heap locations, too?*

$$\frac{A' = Body(\mathbf{x.P})}{\vdash_I \{\ A' * B\ \}\ \mathbf{fold\ x.P}\ \{\ acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here
- We allow an additional assertion A, on both sides:

$$\frac{A' = \mathit{Body}(\mathbf{x.P})}{\vdash_I \{\ A' * B\ \}\ \mathbf{fold\ x.P}\ \{\ \mathit{acc}(x.P) * (\mathit{unfolding}\ x.P\ \mathit{in}\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here
- We allow an additional assertion A, on both sides:

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{\ acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here
- We allow an additional assertion A, on both sides:

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

Isorecursive “fold” statement

- For “fold”, we considered first the simplest rule
- But, this does not allow us to preserve extra information about the state framed by the body
- We allow its preservation with an unfolding assertion
- But, not all such B might be framed, here
- We allow an additional assertion A, on both sides:

$$\frac{\vDash_{frmI} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

Isorecursive "fold" statement (2)

- So, from the fold Hoare rule,

$$\frac{\vDash_{frmI} A \qquad A' = Body(\texttt{x.P})}{\vdash_I \{A * A' * B\}\ \texttt{fold x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- we obtain

$$\frac{\vDash_{frmI} A \qquad A' = Body(\texttt{x.List})}{\vdash_I \{A * A' * \texttt{x.length=3}\}\ \texttt{fold x.List}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ \texttt{x.length=3})\}}$$

Isorecursive “unfold” statement

- The fold statement is

$$\frac{\vDash_{frmI} A \qquad A' = Body(\mathbf{x.P})}{\vdash_I \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- The unfold statement is

$$\frac{}{\vdash_I \{\ \ \ldots\ \ \}\ \mathbf{unfold\ x.P}\ \{\ \ \ldots\ \ \}}$$

Isorecursive "unfold" statement (2)

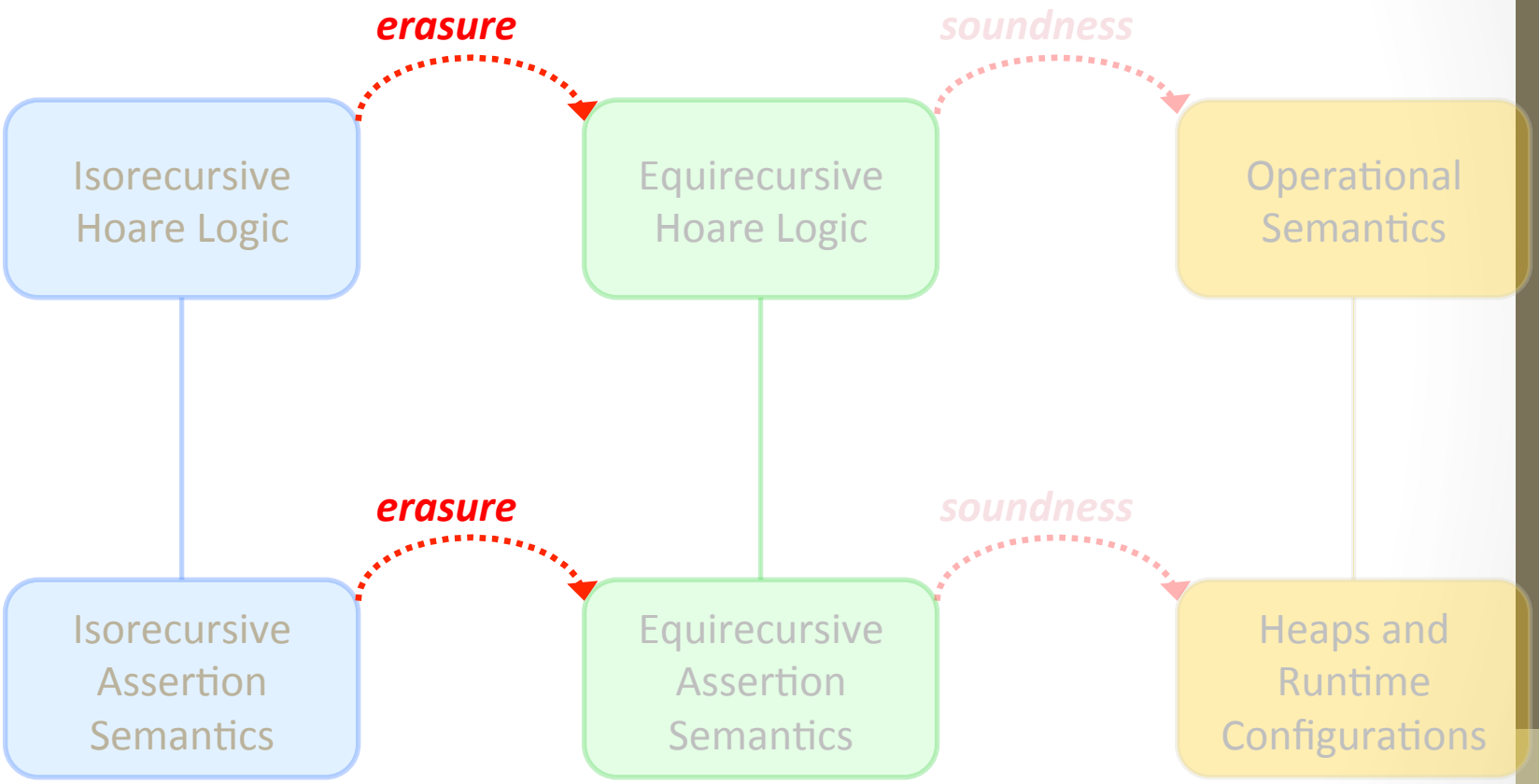
- The fold statement is

$$\frac{\vDash_{frmI} A \qquad A' = Body(\texttt{x.P})}{\vdash_I \{A * A' * B\}\ \texttt{fold x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- The unfold statement is

$$\frac{\vDash_{frmI} A \qquad A' = Body(\texttt{x.List})}{\vdash_I \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}\ \texttt{unfold x.P}\ \{A * A' * B\}}$$

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Erasure: Iso --> Equi (assertions)

- We define an *erasure* function 《A》 from equi-recursive assertions A to iso-recursive (a)
- Idea: replace unfolding expressions with bodies, replace permissions to predicates with instances
- e.g.

《 *acc*(x.list)*
(***unfolding*** x.list ***in*** x.next == null) 》

= x.list * x.next == null

Erasure: Iso -> Equi expressions

- 《x》 = …
- 《null》 = ….
- 《true》 = ….
- 《false》 = ….
- 《e.f》 = …
- 《e.g(e')》 = ….
- 《e = e' 》 = ….
- 《e1? e2: e3》 = ….
- 《**unfolding** e.P **in** e'》 = ….
- 《**acc** (e.f,q)》 = ….
- 《**acc** (e.P)》 = …
- 《a*a'》 = ….
- 《e -> a》 = ….

Erasure: Iso -> Equi expressions (2)

- 《x》 = x
- 《null》 = null
- 《true》 = true
- 《false》 = false
- 《e.f》 = 《e》.f
- 《e.g(e')》 = 《e》.g(《e'》)
- 《e = e' 》 = 《e》 = 《e'》
- 《e1? e2: e3》 = 《e1》 ? 《e2》 **:** 《e3》
- 《**unfolding** e.P **in** e'》 = 《e'》
- 《**acc** (e.f,q)》 = **acc(** 《e》.f, q)
- 《**acc** (e.P)》 = 《e》.P
- 《a*a'》 = 《a》 * 《a'》
- 《e -> a》 = 《e》 **->** 《a》

Erasure: Iso -> Equi assertions

- We also define an *erasure* function 《A》 from equi-recursive assertions A to iso-recursive assertions (a)
- Idea: replace unfolding expressions with bodies, replace permissions to predicates with instances
- e.g.

《 *acc*(x.list)*
(***unfolding*** x.list ***in*** x.next == null) 》

= x.list * x.next == null

- Properties of *erasure*
  - Expression/assertion semantics are preserved
  - Framedness/self-framing are preserved
  - Entailment is preserved (iso to equi notions)

Erasure: Iso --> Equi (statements)

- We extend *erasure* to *statements* 《S》
- Idea: replace fold and unfold with "skip",
  apply erasure to all expressions in the program
- e.g.

```
《unfold x.list; x.val := 4; fold x.list》
    =
 skip; x.val := 4; skip
```

- Derivable Hoare triples preserved by erasure:

  $\vdash_I \{A\}\ S\ \{A'\} \quad \Rightarrow \quad \vdash_E \{\ 《A》\ \}\ 《S》\ \{\ 《A'》\ \}$

- Thus obtain soundness of iso-Hoare Logic! ☺

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for “fold”:

$$\frac{\vDash_{frmI} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for “fold”:

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- In the equi-world:

$\{A * A' * B\}$ **`fold x.P`** $\{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for “fold”:

$$\frac{\vDash_{frml} A \qquad A' = Body(\texttt{x.P})}{\vdash_{I} \{A * A' * B\}\ \texttt{fold x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program:

$\{A * A' * B\}\ \langle\!\langle \texttt{fold x.P} \rangle\!\rangle\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program:

{ A* A'* B } **skip** { A* *acc*(x.P)*(*unfolding* x.P *in* B) }

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program, and the assertions

$$\{\quad A * A' * B \quad\}\ \mathtt{skip}\ \{\quad A * acc(x.P) * (unfolding\ x.P\ in\ B) \quad\}$$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\texttt{x.P})}{\vdash_{l} \{A * A' * B\}\ \texttt{fold x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program, and the assertions

$\{\ \langle\!\langle A * A' * B \rangle\!\rangle\ \}\ \texttt{skip}\ \{\ \langle\!\langle A * acc(x.P) * (unfolding\ x.P\ in\ B) \rangle\!\rangle\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ \langle\!\langle A * A' * B \rangle\!\rangle\ \}\ \mathtt{skip}\ \{\ \langle\!\langle A * acc(x.P) * (unfolding\ x.P\ in\ B) \rangle\!\rangle\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frmI} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ 《A * A' * B》\ \}\ \mathtt{skip}\ \{\ 《A * acc(x.P)》 * 《B》\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program and the assertions

$\{\ \langle\!\langle A * A' * B \rangle\!\rangle\ \}\ \mathbf{skip}\ \{\ \langle\!\langle A * acc(x.P) \rangle\!\rangle\ *\ \langle\!\langle B \rangle\!\rangle\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frmI} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ \langle\!\langle A * A' * B \rangle\!\rangle\ \}\ \mathbf{skip}\ \{\ \langle\!\langle A \rangle\!\rangle\ *\ x.P\ *\ \langle\!\langle B \rangle\!\rangle\ \}$

Isorecursive “fold” statement

- Recall isorecursive Hoare Logic rule for “fold”:

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ \langle\!\langle A * A' * B \rangle\!\rangle\ \}\ \mathtt{skip}\ \{\ \langle\!\langle A \rangle\!\rangle\ *\ x.P\ *\ \langle\!\langle B \rangle\!\rangle\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ 《A》 * 《A'》 * 《B》\ \}\ \mathbf{skip}\ \{\ 《A》 * x.P * 《B》\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathtt{x.P})}{\vdash_{l} \{A * A' * B\}\ \mathtt{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ 《A》 * 《A'》 * 《B》\ \}\ \mathtt{skip}\ \{\ 《A》 * x.P * 《B》\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\mathbf{x.P})}{\vdash_{I} \{A * A' * B\}\ \mathbf{fold\ x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program.. and the assertions..

$\{\ 《A》 * 《Body(\mathbf{x.P})》 * 《B》\ \}\ \mathbf{skip}\ \{\ 《A * x.P * 《B》\ \}$

Sanity: erasure of iso-fold Hoare Triple

- Recall isorecursive Hoare Logic rule for "fold":

$$\frac{\vDash_{frml} A \qquad A' = Body(\texttt{x.P})}{\vdash_{l} \{A * A' * B\}\ \texttt{fold x.P}\ \{A * acc(x.P) * (unfolding\ x.P\ in\ B)\}}$$

- Apply *erasure* to the program and the assertions

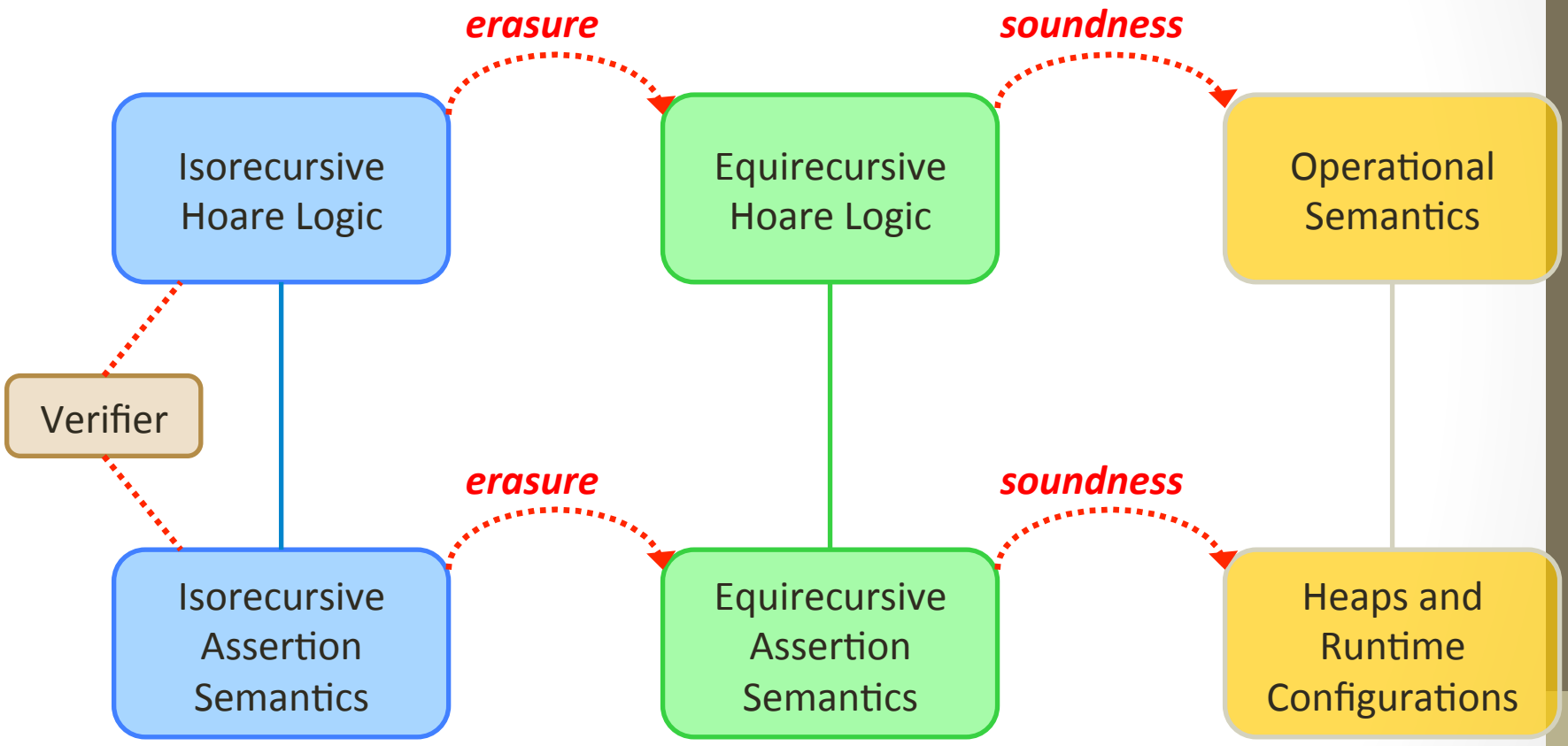
$\{ \langle\!\langle A \rangle\!\rangle * \langle\!\langle Body(\texttt{x.P}) \rangle\!\rangle * \langle\!\langle B \rangle\!\rangle \}\ \texttt{skip}\ \{ \langle\!\langle A \rangle\!\rangle * x.P * \langle\!\langle B \rangle\!\rangle \}$

  - The only difference between pre/postconditions is $\langle\!\langle Body(\texttt{x.P}) \rangle\!\rangle$ on one side, and x.P on the other
- In the equi-recursive semantics, the predicate and its body are equivalent; thus, we can derive this triple

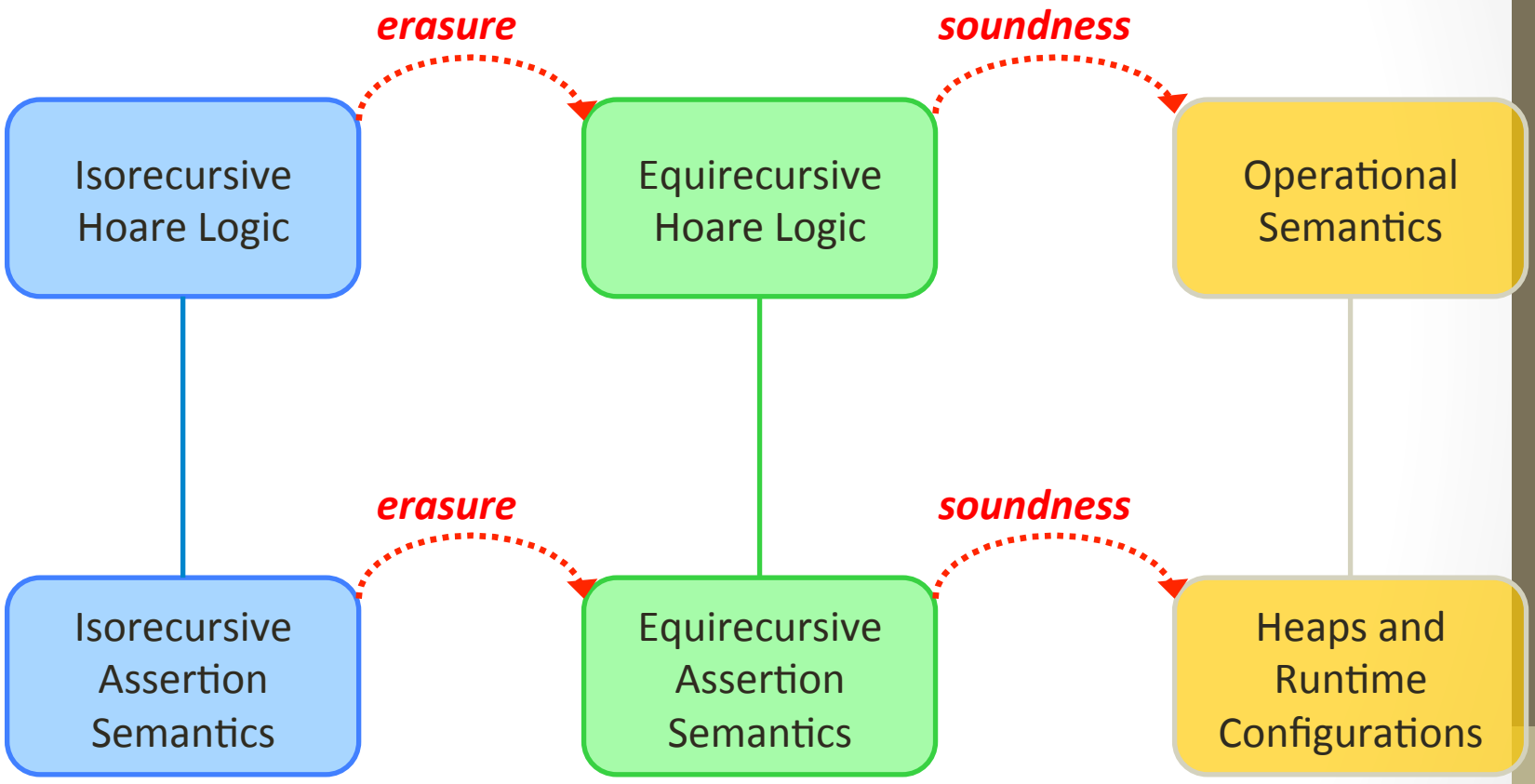
Conclusions and Future Work

- We presented equirecursive and isorecursive..
  - expression/assertion semantics
  - Axiomatic semantics (Hoare Logics)
  - ... and shown their relationships
- ECOOP'2013 paper also discusses many design decisions and delicate points, along with proofs.
  Requires more kinds of permissions.
- We would like to formally connect our axiomatic semantics with the (Boogie) encoding used in Chalice tool
- We would like to extend to the full logic
  (eg a=>a rather than e=>a)
- We would like to adopt a more semantic approach to framedness.

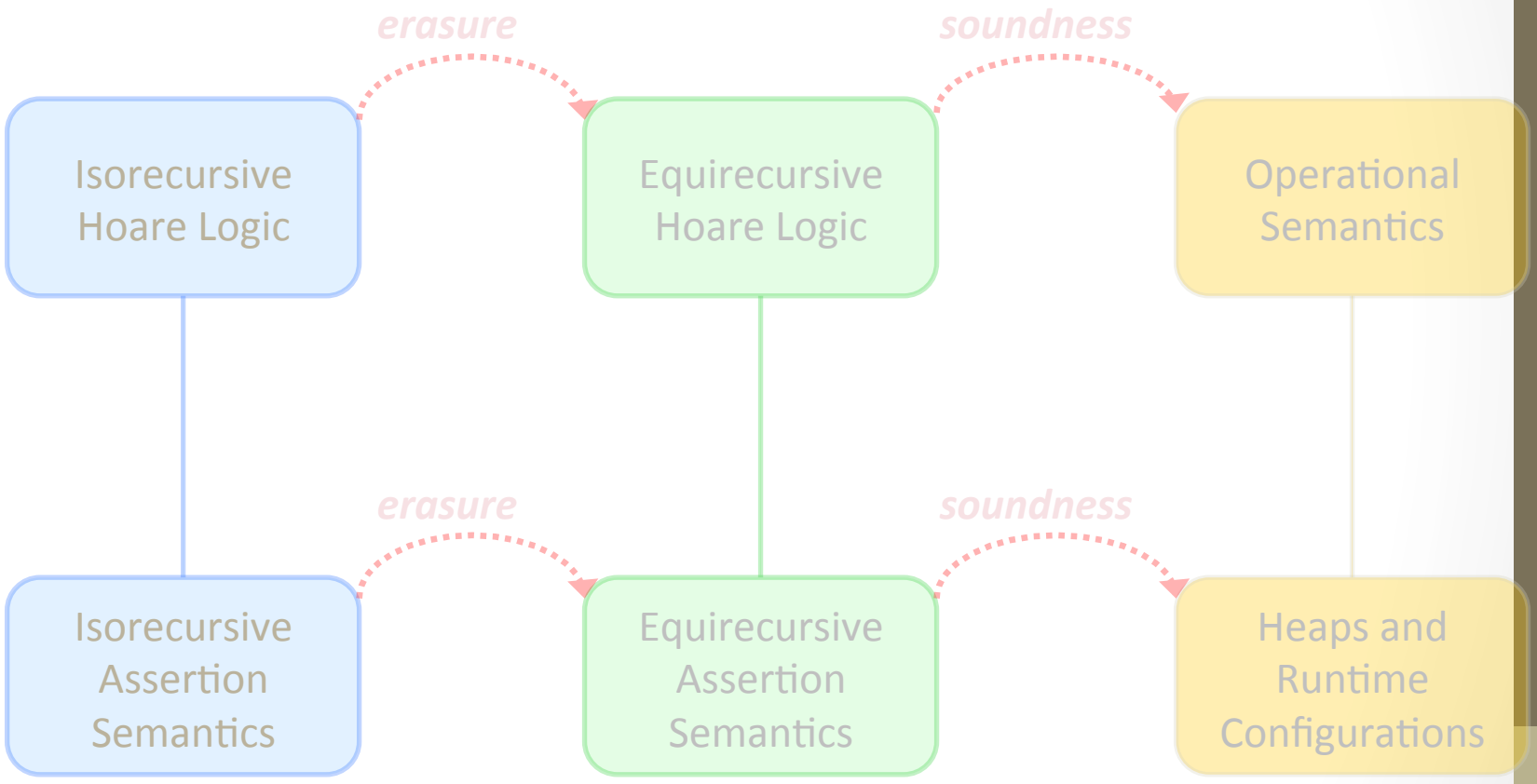
The End.. any questions?



Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations

Overview


erasure
soundness
Isorecursive Hoare Logic
Equirecursive Hoare Logic
Operational Semantics
erasure
soundness
Isorecursive Assertion Semantics
Equirecursive Assertion Semantics
Heaps and Runtime Configurations