Iterator Specification with Typestates

Kevin Bierhoff – Carnegie Mellon University

SAVCBS 10 November 2006

Iterator Specification with Typestates

- Specify Iterator protocol as abstract state machine using **typestates**
- Ensure Iterator consistency with **access permissions**
- Talk focuses on **read-only Iterators**
  - Modifying Iterators in the paper

Java Iterators Enumerate and Modify Collections

- interface Iterator
  - boolean hasNext()
  - Object next()
  - void remove()
- interface Collection
  - Iterator iterator()
  - … methods for adding, removing, etc.

Is there another element?

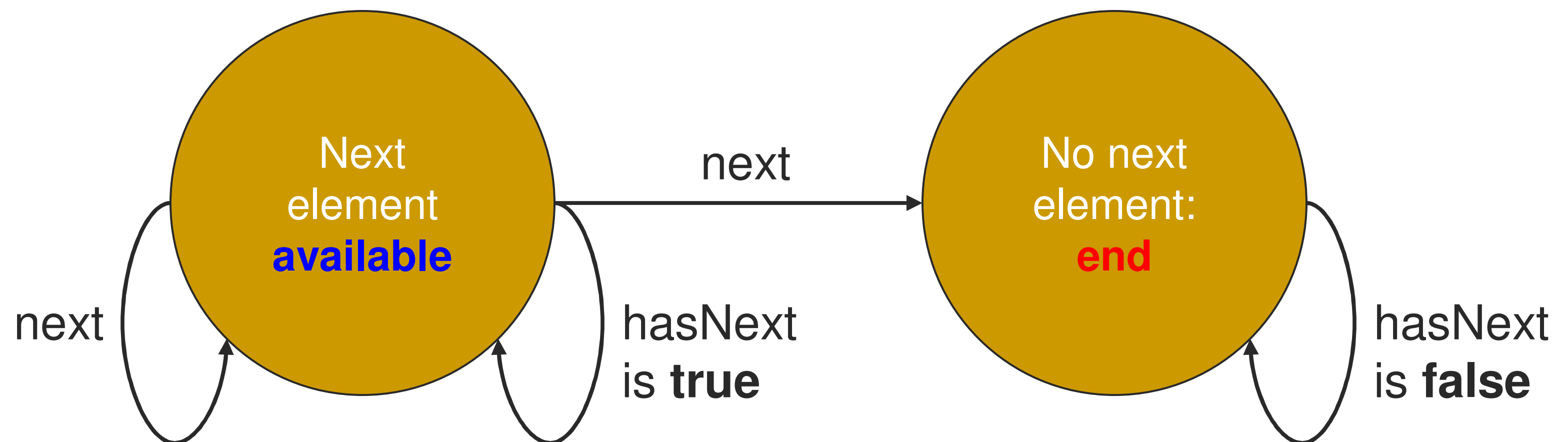
Give me the next element

Remove the element you just gave me

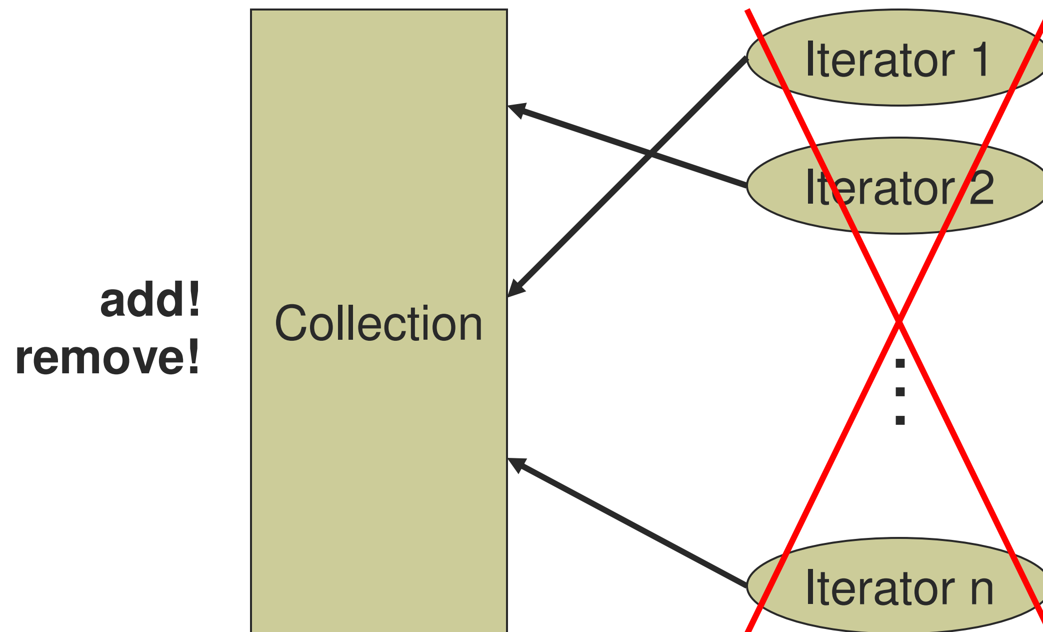
Create iterator over collection

Focus on read-only iterators in this talk

Iterator State Machine



Iterator Invalidation Through Concurrent Modification

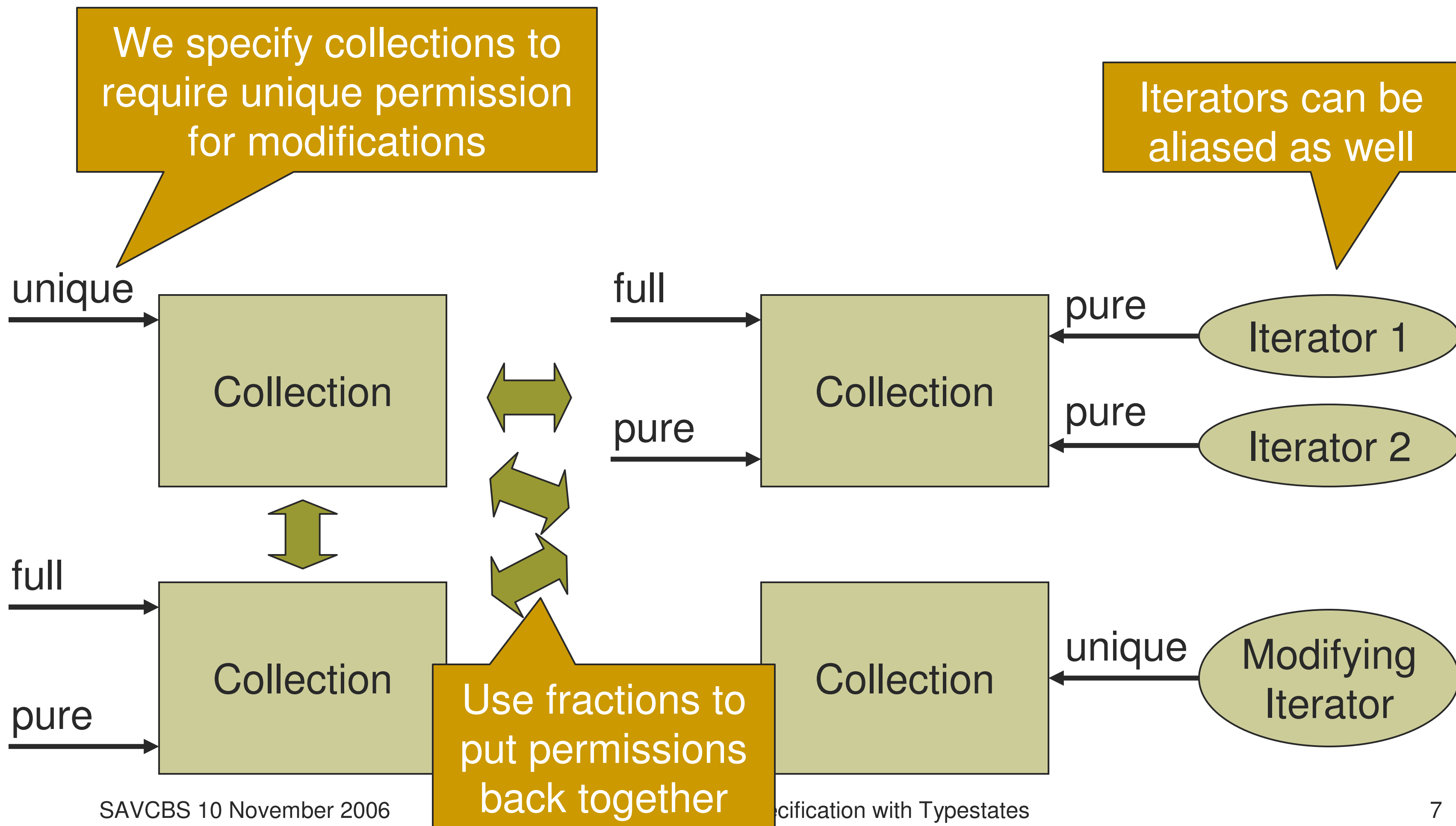


Need to ensure that Iterators are not used after concurrent modification

Access Permissions For Typestate Tracking

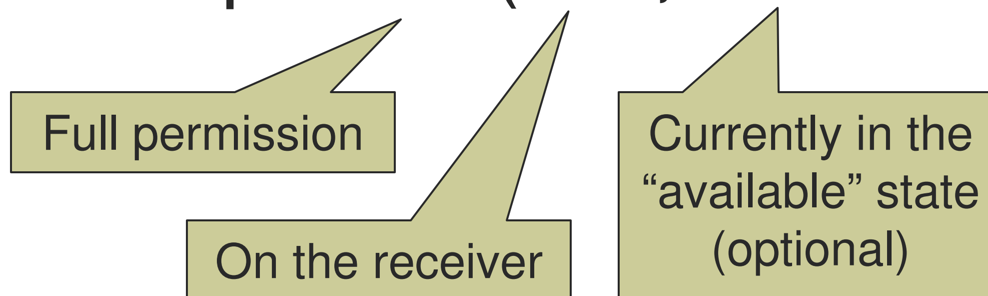
- Associate **access permissions** with object references
  - Limit what reference can do
  - Keep permissions consistent
- Enforce one of two situations
  - One **unique** permission
  - One **full** and many **pure** permissions
    - Full permission can modify object
    - Pure permissions can read from object

Permissions Can Model Collection Aliasing Through Iterators



Access Permissions For References

- Permissions to objects
  - What kind of permission?
  - For what reference?
  - What do we know about the state?
- Example: full(*this*, **available**)



Linear Logic for Method Specifications

- Permissions as resources
- Method behavior
  - $A \multimap B$
  - Transitions from A to B
- Conjunction
  - $A \otimes B$
  - A and B at the same time
- Disjunction (external choice)
  - $A \oplus B$
  - Either A or B non-deterministically—be ready for either

Read-only Iterator Specification

- interface Iterator
  - boolean hasNext() :
    pure(*this*) $\multimap$
    (*result* = true $\otimes$ pure(*this*, **available**)) $\oplus$
    (*result* = false $\otimes$ pure(*this*, **end**))
  - Object next() :
    full(*this*, **available**) $\multimap$ full(*this*)
- interface Collection
  - Iterator iterator() :
    pure(*this*) $\multimap$ unique(*result*)

Dynamic state test: result value paired with state information

Don’t know state after call

Iterator captures permission to collection

Enforces the characteristic hasNext() / next() call pairing

Releasing Collection Permission Upon Iterator Destruction

- interface Iterator<*c*: Collection, *g*: Fraction function>
  - hasNext, next as before
  - void finalize() :
    unique(*this*) ⊸ pure(*c, g*)
- interface Collection
  - Iterator iterator() :
    ∀ g : Fraction function.
    (pure(*this, g*) ⊸
    ∃ *result* : Iterator<*this, g*>. unique(*result*)

Iterator parameterized by iterated collection and permission fraction

Release "captured" collection permission upon destruction

Iterator captures permission to collection

Summary

- Specified read-only Iterators with **typestates**
  - Modifying Iterators in the paper
- Ensure Iterator consistency with **access permissions**
  - Can modify only unique collections
  - Cannot turn read-only into modifying Iterator after creation (details in paper)