Interruptible Iterators

Jed Liu   Aaron Kimball   Andrew C. Myers

Department of Computer Science
Cornell University

33rd ACM Symposium on Principles of Programming Languages
13 January, 2006

Cornell University
Computer Science

Iteration Abstractions

Important to support iteration abstractions well

- Clients get on-demand access to elements of a lazily-evaluated sequence
- Many mainstream languages support IAs
  - e.g., C++, Python, Ruby
  - Evolving to support better: C# 2.0, Java 1.5
  - Libraries too: Java Collections, Microsoft .NET
- Iterators are hard to implement
  - Especially if they support imperative update

Interruptible iterators make IAs easier to implement

- Implemented as part of JMatch

Iterators: Easy to Use, Hard to Write

Easy to use: Java iterator interface

```
interface Iterator {
  boolean hasNext(); // Is there a next element?
  Object next(); // Return the next element
  void remove(); // Remove last element returned
}
```

Can be hard to implement

- Iteration must continue where it last left off
- Iterator can become awkward state machine

Binary Tree Iterator Example in Java

```java
class TreeIterator implements Iterator {
  Iterator subIterator;
  boolean hasNext;
  Object current;

  // 1 = Iterating through left child
  // 2 = Just yielded current node value
  // 3 = Iterating through right child
  int state;

  TreeIterator() {
    subIterator = Tree.this.left.iterator();
    state = 1;
    current = preload();
  }

  public boolean hasNext() {
    return hasNext;
  }

  public Object next() {
    if (!hasNext)
      throw new NoSuchElementException();

    Object result = current;
    current = preload();
    return result;
  }
```

```java
  private Object preload() {
    loop: while (true) {
      switch (state) {
      case 1:
      case 3:
        hasNext = true;

        if (subIterator.hasNext()) {
          return subIterator.next();
        }

        if (state == 1) {
          state = 2;
          return Tree.this.value;
        }

        hasNext = false;
        return null;

      case 2:
        subIterator =
          Tree.this.right.iterator();
        state = 3;
        continue loop;
      }
    }
  }
}
```

Binary Tree Iterator Example in Java

```java
class TreeIterator implements Iterator {
  Iterator subIterator;
  boolean hasNext;
  Object current;

  // 1 = Iterating through left child
  // 2 = Just yielded current node value
  // 3 = Iterating through right child
  int state;

  TreeIterator() {
    subIterator = Tree.this.l
    state = 1;
    current = preload();
  }

  public boolean hasNext() {
    return hasNext;
  }

  public Object next() {
    if (!hasNext)
      throw new NoSuchElementException();

    Object result = current;
    current = preload();
    return result;
  }
```

```java
  private Object preload() {
    loop: while (true) {
      switch (state) {
      case 1:
      case 3:
        hasNext = true;

        if (subIterator.hasNext()) {
          return subIterator.next();
        }

                     1) {

                     .this.value;

                     se;
        return null;

      case 2:
        subIterator =
          Tree.this.right.iterator();
        state = 3;
        continue loop;
      }
    }
  }
}
```

Even worse when you add support for updates

Coroutine Iterators

- Increasingly popular: C# 2.0, Python, Ruby
- Iterator as a coroutine:
  - Separate stack
  - Iterator suspends execution by *yielding* values
  - Client obtains more values by resuming iterator

Example: JMatch binary tree iterator

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

Coroutine Iterators

- Increasingly popular: C# 2.0, Python, Ruby
- Iterator as a coroutine:
  - Separate stack
  - Iterator suspends execution by *yielding* values
  - Client obtains more values by resuming iterator

Example: JMatch bina

elements is an iterator

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

Coroutine Iterators

- Increasingly popular: C# 2.0, Python, Ruby
- Iterator as a coroutine:
  - Separate stack
  - Iterator suspends execution by *yielding* values
  - Client obtains more values by resuming iterator

Example: JMatch binary tree iterator

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

Coroutine Iterators

- Increasingly popular: C# 2.0, Python, Ruby
- Iterator as a coroutine:
  - Separate stack
  - Iterator suspends execution by *yielding* values
  - Client obtains more values by resuming iterator

Example: JMatch binary tree iterator

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

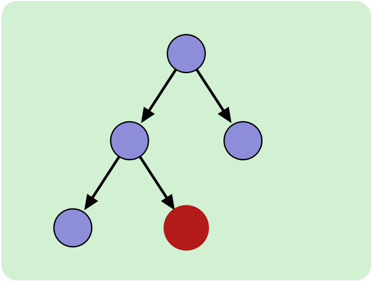
Coroutine Iterators

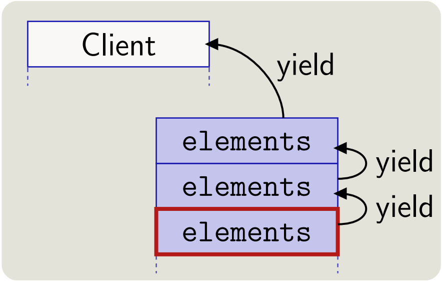
- Increasingly popular: C# 2.0, Python, Ruby
- Iterator as a coroutine:
  - Separate stack
  - Iterator suspends execution by *yielding* values
  - Client obtains more values by resuming iterator

Example: JMatch binary tree iterator

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

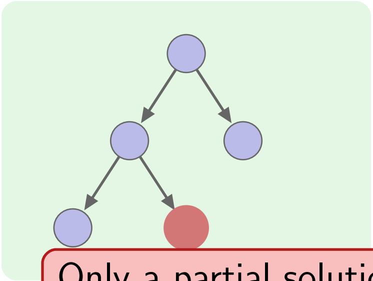
Coroutine Iterators

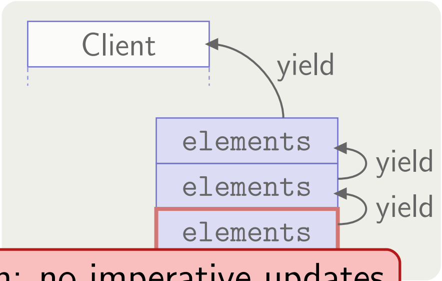




```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

Coroutine Iterators





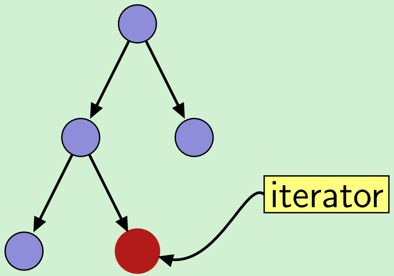
Only a partial solution: no imperative updates

```
class Node {
 int val;  Node left, right;
 int elements() iterates(result) {
   foreach (int elt = left.elements()) yield elt;
   yield val;
   foreach (int elt = right.elements()) yield elt;
 }
}
```

Imperative Updates

- Unsafe to change underlying data structure directly during iteration
- All updates must go through the iterator
  - Java: `remove()`
- Previous coroutine iterators don't have update

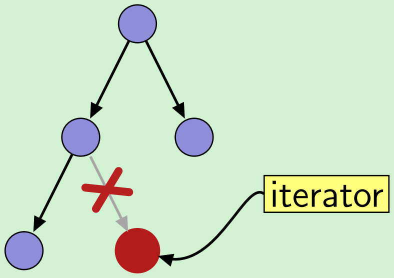
Example: Tree iterator



Imperative Updates

- Unsafe to change underlying data structure directly during iteration
- All updates must go through the iterator
  - Java: `remove()`
- Previous coroutine iterators don't have update

Example: Tree iterator



Iterator no longer points to part of the tree

Interruptible Iterators

JMatch extends coroutine iterators to handle updates via *interrupts*:

1. Client raises interrupt
2. Iterator handles interrupt
3. Control returns to client after `raise`

Example

```
Collection c = ...;
foreach (Object o = c.elements()) {
  if (o == null) raise new Remove();
  System.out.println(o);
}
```

Interruptible Iterators

JMatch extends coroutine iterators to handle updates via *interrupts*:

1. Client raises interrupt
2. Iterator handles interrupt
3. Control returns to client after `raise`

Example

Generates a `Remove` interrupt

```
Collection c = ...;
foreach (Object o = c.elements()) {
  if (o == null) raise new Remove();
  System.out.println(o);
}
```

Interruptible Iterators

JMatch extends coroutine iterators to handle updates via *interrupts*:

1. Client raises interrupt
2. Iterator handles interrupt
3. Control returns to client after `raise`

Example

Receives and handles the interrupt

```
Collection c = ...;
foreach (Object o = c.elements()) {
  if (o == null) raise new Remove();
  System.out.println(o);
}
```

Interruptible Iterators

JMatch extends coroutine iterators to handle updates via *interrupts*:

1. Client raises interrupt
2. Iterator handles interrupt
3. Control returns to client after `raise`

Example

Execution continues immediately after `raise` statement

```
Collection c
foreach (Object   = c.elements()) {
  if (o == null) raise new Remove();
  System.out.println(o);
}
```

Interruptible Iterators

JMatch extends coroutine iterators to handle updates via *interrupts*:

1. Client raises interrupt
2. Iterator handles interrupt
3. Control returns to client after `raise`

Example

```
Collection c = ...;
foreach (Object o = c.elements()) {
  if (o == null) raise new Remove();
  System.out.println(o);
}
```

Declaring Interrupt Handlers

- JMatch iterators declare handled interrupts
- Compiler checks all interrupts are handled

Example

```
interface Collection {
  ...
  Object elements() traps Remove
                    iterates(result);
}
```

Declaring Interrupt Handlers

- JMatch iterators declare handled interrupts
- Compiler checks all interrupts are handled

`elements` is an iterator that handles `Remove` interrupts

Example

```
interface Collection {
  ...
  Object elements() traps Remove
                    iterates(result);
}
```

Writing an Interruptible Iterator

Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {

  yield head;




  foreach (Object elt = tail.elements())
    yield elt;
}
```

Writing an Interruptible Iterator

Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {

  yield head;



  foreach (Object elt = tail.elements())
    yield elt;
}
```

Handling `SetValue` overwrites previous element returned
(à la Java: `ListIterator.set()`)

Writing an Interruptible Iterator

Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {

  yield head;




  foreach (Object elt = tail.elements())
    yield elt;
}
```

Interrupt appears to be raised by `yield`; propagates outward like an exception

Writing an Interruptible Iterator

Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Writing an Interruptible Iterator

Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Interrupt appears here too

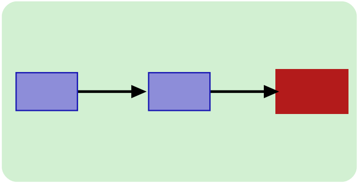
Writing an Interruptible Iterator

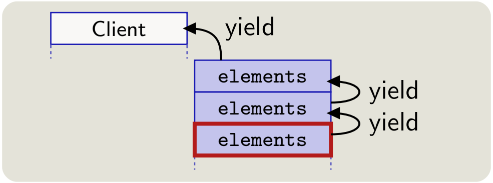
Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Receives interrupt

Writing an Interruptible Iterator

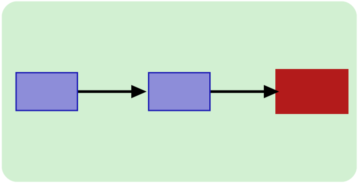


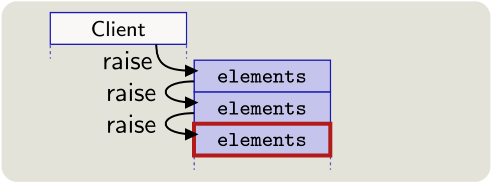


Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Writing an Interruptible Iterator

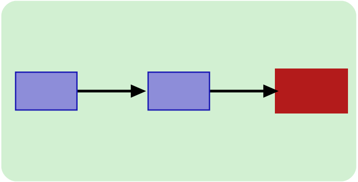


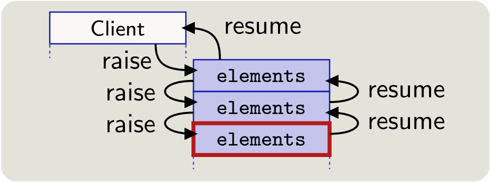


Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Writing an Interruptible Iterator

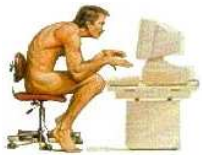




Example: Linked list iterator

```
// Object head;  List tail;
Object elements() traps SetValue iterates(result) {
  try {
    yield head;
  } trap (SetValue s) {
    head = s.value;  // resume in caller
  }
  foreach (Object elt = tail.elements())
    yield elt;
}
```

Declarative Iterators



State Machine Iterators

Coroutine Iterators

Declarative Iterators

+ Interrupts

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Example uses

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Logical formula interpreted in multiple ways

Example uses

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Mode declarations

Example uses

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Iterates all key-value pairs

Example uses

```
foreach (map.contains(Object key, Object value)) ...
```

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Iterates all keys that map to a given value

Example uses

```
foreach (map.contains(Object key, Object value)) ...
foreach (map.contains(Object key, "foo")) ...
```

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Returns value mapped by a given key

Example uses

```
foreach (map.contains(Object key, Object value)) ...
foreach (map.contains(Object key, "foo")) ...
let map.contains("foo", Object value);
```

Declarative Iterators

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)

  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

Implicit mode

Example uses

```
foreach (map.contains(Object key, Object value)) ...
foreach (map.contains(Object key, "foo")) ...
let map.contains("foo", Object value);
if (map.contains("foo", "bar")) ...
```

Declarative Iterators + Imperative Update

Example: JMatch hash map [PADL 2003]

```
boolean contains(Object key, Object value)
  traps Remove
  iterates(key,value) iterates(key) returns(value)
(
  int n = hash(key) &&
  Bucket b = table[n] &&
  b.contains(key, value)
)
```

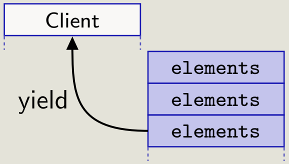
Updates can be supported via interrupts

Example use

```
foreach (map.contains(Object key, Object value))
  if (value == null) raise new Remove();
```

Implementation

- JMatch implemented using Polyglot extensible compiler framework [CC 2003]
  - Java back-end available for download
  - Designed C++ back-end for performance evaluation
    - Better memory management for coroutine stack
- *Tail-yield optimisation:* send values back to client in constant time



Evaluation

Expressiveness (LOC)

	Java	JMatch	Savings
ArrayList	204	112	45%
LinkedList	249	155	38%
HashMap	434	158	64%
TreeMap	805	472	41%
Total	1692	897	47%

Performance vs. C++ STL

Average 3% difference iterating 250k elements: LinkedList, HashMap, TreeMap vs. STL equivalent

- More results in paper, including vs. Java

Related Work

- Coroutine iterators
  - CLU, ICON, Python, Ruby, C#
  - Sather: Limited support for imperative updates through “hot” arguments
- Coroutines
  - Simula, Modula-2, BETA
- Resumption-style exceptions
  - Cedar
- First-class continuations
  - SML/NJ, Scheme, Ruby

Summary

- Interrupts make it easier to write iteration abstractions with imperative update
  - Supports coroutine and declarative iterators
- Implemented for Java in JMatch
- LOC savings without performance penalty

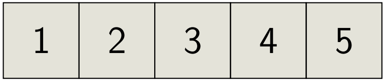
Also in the paper...

- Non-compositionality of Java iterators
- Interaction of interrupts & exceptions
- Static checking of interrupts
  - Checks all raised interrupts have unique handler
- Support for first-class iterator objects
  - Implement Java iterator interface

http://www.cs.cornell.edu/projects/jmatch/

Java Iterators are Non-Compositional

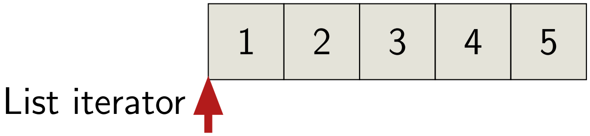
Client



```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

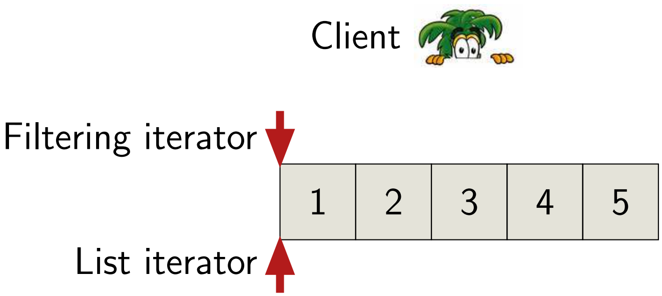
Java Iterators are Non-Compositional

Client



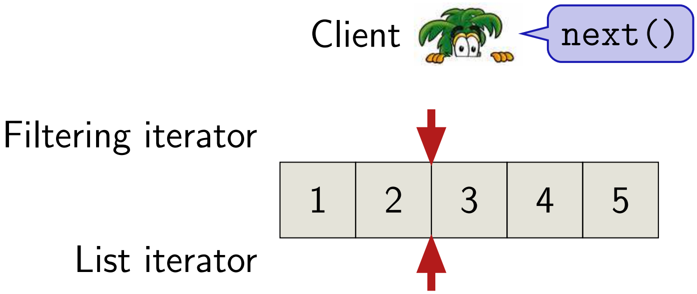
```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Java Iterators are Non-Compositional



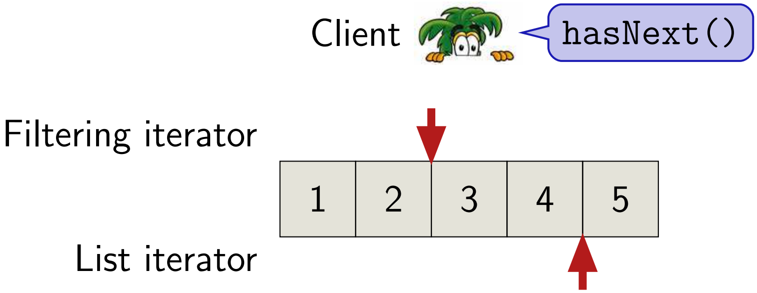
```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Java Iterators are Non-Compositional



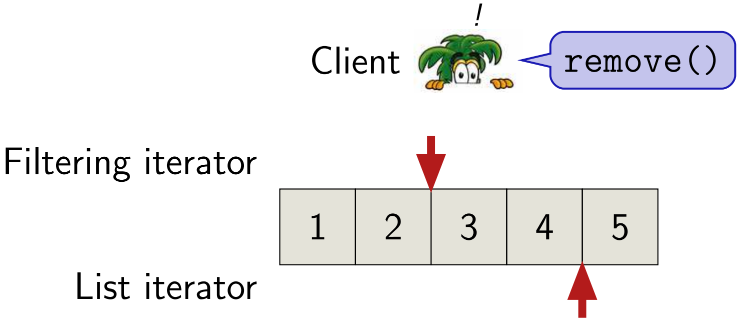
```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Java Iterators are Non-Compositional



```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Java Iterators are Non-Compositional



```
interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Performance Results

	ArrayList	LinkedList	HashMap	TreeMap
JMatch	135.0	56.1	3.7	3.1
C++ STL	215.0	57.7	3.1	3.9
Java	6.3	10.3	4.2	3.5

Millions of elements iterated per second, iterating over collections of 250k elements. Average of 8 measurements, $\sigma \leq 5\%$.