Specifying Java Iterators using JML and Esc/Java2

David R. Cok
Eastman Kodak Company
10 November 2006
at
SAVCBS 2006
The Problem

- Iterators walk through a sequence of values
- May have many independent iterators for a given iterable object
- Iterators may modify the parent object
- No requirements on the sequence of values returned by an iterator
The Problem: interactions

- If a set of Java iterators have a common parent object, there is an interaction among them:
  - an iterator may remove an object only once
  - if an iterator removes an object from the parent, all other iterators subsequently may have undefined behavior
  - if the parent object is modified, all iterators may have undefined behavior
  - Note: subclasses may define the behavior
In this paper

- Discuss only the second issue – interactions
- Specify the interfaces, not specific instantiations
- Use JML and Esc/Java2
- Goal: determine where the specification language falls short
package java.lang;

public interface Iterable<E> {
    public Iterator<E> iterator();
}

The only functionality is to produce an Iterator. An Iterator need not have a parent Iterable.
package java.lang;

public interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void remove();
}

The Iterator<E> interface
package java.lang;

public interface Iterator<E> {
  public E next();

  /*@
   public behavior
   requires removeOK;
   assignable removeOK;
   ensures removeOK;
   */

  /*@
   also public exceptional_behavior
   requires removeOK;
   assignable removeOK;
   ensures !removeOK;
   signals_only IllegalStateException;
   */

  public void remove();
}

//@ public instance model boolean removeOK;
//@ initially !removeOK;

//@ ....
assignable removeOK;
ensures removeOK;

/*@ ....
assignable removeOK;
ensures removeOK;
*/
A state machine!

- This actually encodes a little state machine:

- The state machine, or some equivalent, is a better specification – more obviously correct to a human.
Other examples

- A class that requires the calling of an initialization method before any other methods
- Iterator: if hasNext() returns false, then next() is illegal

- See Cheon & Perumendla, 2005, 2006 for some initial work on this issue
What sort of syntax?

- Regular expressions?
  
  \(( (next)^+ (remove) )*\) is OK
  
  \(?* (remove)(remove)\) is ERROR
  
  \((remove)\) is ERROR

- What about nested pairs (e.g. open/close)? method arguments? return values?
A research question

- What is the best way to specify method call sequences?
- What amount of syntax is helpful?
- How much is too much?
- Need lots of case studies and analysis of real code
Modification interactions

- Iterators store a proxy for the state of the Iterable
- If that state changes, except by the iterator itself, the iterator becomes invalid

- (See the paper for details)
- (Actual Java implementation is similar)
package java.lang;

public interface Iterator<E> {
  //@ public instance model Iterable iterable;
  //@ public instance model int iteratorTime;
  ...

  //@ public normal_behavior
  ensures iteratorTime>=iterable.lastModified;

  public pure model boolean isValid();
  /*@*/

  public void remove();
  }
package java.lang;

public interface Iterable<E> {
    /*@ public instance model int lastModified;
        public instance model int maxIterator;
        constraint lastModified >=
            \old(lastModified);
    ...
    /*@ ensures \result.iterable = this;
        ensures \result.isValid();
        ensures maxIterator >=
            \result.iteratorTime; */
    public Iterator<E> iterator();

    // Any subclass method that modifies
    // the Iterable must include specs
    // that invalidate the associated
    // Iterators, like this:
    /*@
        ensures lastModified >
            maxIterator;
    */
    public void clear();
A few notes

- No object alters fields within a different object
- Iterators must be able to see the fields of the parent Iterable
- Requirement on the specifications of all methods that mutate the Iterable
Another research topic

- How to apply specs to groups of methods?
- Would like the default to be such that forgetting to add a specification causes warnings
- History constraints impose a requirement on all methods; is there a way to impose a requirement on some methods—and how does one say which ones?:
  - by labeling with a Java annotation?
  - list method names?
  - defining a property?

/*@ constraint for (@Modifying) lastModified > maxIterator; */
/*@constraint except (@NonModifying) lastModified>maxIterator;*/
Ghost fields vs. model fields

- Ghost fields are additional spec-only fields
- Model fields are abstractions of the state
- Both work fine for static checking
- Both need implementations for runtime checking, inconvenient especially for classes without source
  - Ghost fields: altered through “set” statements (but one does not always have access to the implementation)
  - Model fields: need an implementation in terms of Java or ghost fields (which can be duplicative)
Testing using Esc/Java2

- Wrote a number of Java classes that utilized these specified interfaces
- Esc/Java2 successfully warned about invalid uses and was quiet about valid uses
Additional issue

- The validation of the *interface* specifications is through writing test cases and running a code verifier/bug finder.
- No tools to check that the specification is well-covered by the test cases (jmlc does capture some metrics)
- For *classes* there is the implementation to check, but coverage is still unchecked
Conclusion

- JML and Esc/Java2 “worked” for this part of the iterator problem
- Two research questions:
  - Facilities are needed for specifying sequences of method calls
  - How to write specs that apply to many methods