Case Study: Debugging a Discovered Specification for `java.util.ArrayList` by Using Algebraic Interpretation

Technical Report CU-CS-970-04

Johannes Henkel and Amer Diwan
{henkel,diwan}@cs.colorado.edu

February 25, 2004

We use our specification discovery tool [1] to discover the specification for *java.util.ArrayList*. Given the discovered specification, our algebraic specification interpreter [2] provides a rapid prototype of the *java.util.ArrayList* class. We debug the prototype (and therefore the specification) by using it within a BibTeX parser; this involves completing the discovered specification by addressing the warning messages printed by the specification interpreter. We report on the incremental changes made to the discovered specification. We also discuss how our specification discovery tool can be improved to increase its effectiveness in providing a good starting point for a complete and correct specification.

1 Specification Discovery

We ran our specification discovery tool [1] on the `java.util.ArrayList` class from Sun's JDK 1.4.2_02. This took 22 minutes on a Pentium 2.4 Ghz, 2 GB of RAM, SuSE Linux 9.0. The discovery tool came up with 146 algebraic axioms.

1.1 Commenting out unsupported axioms

In a first step, we remove 76 axioms for two reasons: (i) Even though we do not support arrays in our interpreter or our specification discovery tool, the discovery tool still discovers 42 axioms containing arrays. (ii) 75 of the discovered axioms contain references to particular objects. For example:

forall x0:ArrayList ***(Axiom 5)***[1]
getClass(x0).retval==java.lang.Class@31789152

This axiom says that whenever we use the *getClass* operation on an arbitrary *ArrayList* object, we get the same *Class* object as a result. However, this object is specific to the run of the specification discovery tool and can thus not be used by the interpreter.

70 axioms which our interpreter understands remain.

The number of axioms axioms that we have to remove is higher than we expected. However, we find that there are some remedies that we consider to implement in the future. For example, supporting conditional axioms in the discovery tool would allow us to remove many redundancies. In particular, we find 30 axioms where it would be beneficial to allow the condition *equals(x1,x2).retval==false* (*x1* and *x2* are instances of *java.lang.Object*). For example:

indexOf(add(ArrayList().state,Object@18961126).state,Object@31038029).retval ***(Axiom 107)***
== −1

could be generalized into

```
forall x1:Object forall x2:Object
    if equals(x1,x2).retval==false then
        indexOf(add(newArrayList().state,x1).state,x2).retval
        == −1
```

thereby eliminating other axioms that say the same with different object constants.

1.2 Algebraic Interpretation

We use our algebraic interpreter to simulate the *ArrayList* class and the corresponding *Iterator* interface for a client application. For this case study, our client application is a BibTeX parser written in Java [2]. We chose this client application because it is not dependent on libraries other than the Java standard libraries, uses collection classes, and we were familiar with the code.

We run the interpreter with the following command line arguments:

instrumentation targets: This parameter specifies which classes in the client application (i.e. the simulation client in Figure 1) should be modified so that references to simulation subjects (in this case *java.util.ArrayList*) are replaced with references to simulation stubs (in this case *SIMSTUB_java.util.ArrayList*). We choose all client classes.

client entry point: This parameter specifies the class of the simulation client which contains the *public static void main(String[])* method that is the entry point for execution. In our case, the entry point is the class *bibtex.Main*.

[1] Axioms in this document are numbered according to the full specification given in the appendix

[2] Available at `http://www-plan.cs.colorado.edu/henkel/stuff/javabib/`.

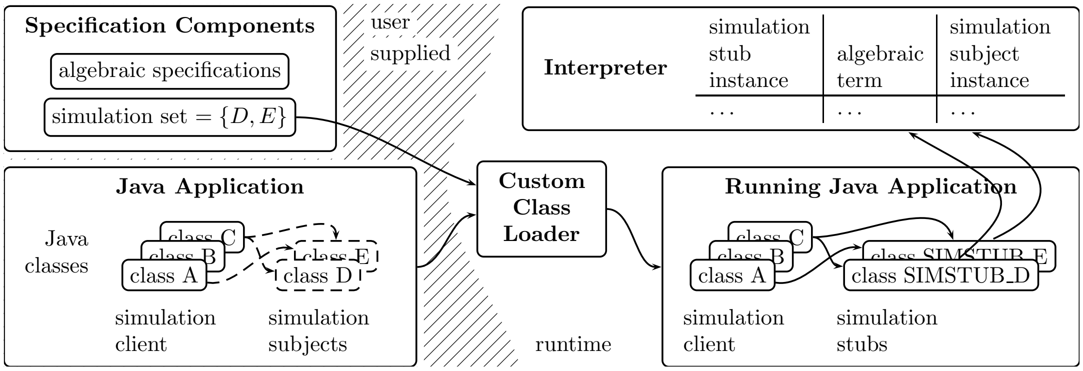


Figure 1: Architecture of our system

specification file: The specification in our algebraic specification language.

trace output file: The output file name for the rewriting trace.

Since we have specified simulation subjects (actual implementations of the specification), the application executes successfully and behaves as usual. However, we also find that there is a warning message in the output (shortened for clarity), which indicates that our specification is incomplete.

Warning 1 *bibtex.dom.BibtexFile.printBibtex(), line92:*
Algebraic Interpreter failed to compute a value.
term = iterator(add(...,bibtex.dom.BibtexEntry@7245716).state).retval

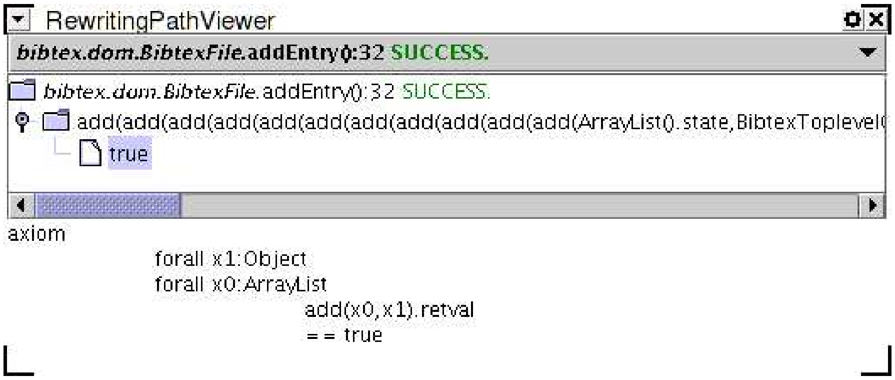
Before printing the warning message, the interpreter successfully computes the return values of the *boolean add(Object)* method when the array list is constructed by the client application. By studying the rewriting trace file, we find out that the interpreter used the rewriting rule

forall l:ArrayList forall o:Object ***(Axiom 81)***
add(l,o).retval==true

which was in the set of discovered axioms.

As an alternative for examining the rewriting trace, we developed a little user interface for the rewriting engine as shown in Fig. 2. Using the drop down menu at the top of the window, the user selects which rewriting computation to view. Below, a tree view of a is shown in which each term has been derived from its parent by using one rewriting step. When a user selects a term in the tree view, the axiom that has been used to generate that term from its parent is displayed in the text area at the bottom of the window.

The warning message (Warning 1) in the interpreter output means that the interpreter had no way to compute the return value of the iterator function. Since *Iterator* is not in the simulation set, the interpreter tried to compute a return value by rewriting. But the return value is really a new object with a new state. Thus, we edit the specification file



```
bibtex.dom.BibtexFile.addEntry():32 SUCCESS.
bibtex.dom.BibtexFile.addEntry():32 SUCCESS.
add(add(add(add(add(add(add(add(add(add(add(ArrayList().state,BibtexToplevel(
    true
axiom
            forall x1:Object
            forall x0:ArrayList
                        add(x0,x1).retval
                        == true
```

Figure 2: User interface for rewriting engine.

to add *Iterator* to the simulation set. This means that instead of trying to compute the value, the interpreter will create a new simulation stub instance for the iterator and use the term that was contained in Warning 1 as its state. Warning 1 disappears, but many new warnings appear, since the interpreter now also tries to interpret the *hasNext()* and *next()* operations that the client performs on the iterator. The first warning is as follows.

Warning 2 *bibtex.dom.BibtexFile.printBibtex(), line92:*
Algebraic Interpreter failed to compute a value.
term = hasNext(iterator(add(...,bibtex.dom.BibtexEntry@9715140).state).retval).retval

We add the following axiom, which takes care of it.

forall x1:ArrayList forall x2:Object ***(Axiom 147)***
hasNext(iterator(add(x1,x2).state).retval).retval
== true

This axiom was not discovered by our specification discovery tool since it contains a return value computation inside the term on the left side. In order to discover axioms such as this, we need to extend our term generation component in the discovery tool: At this point, the state terms that are generated by the term generator involve only one algebra. In Axiom 147, the state of the Iterator is expressed by the return value of the observer function *iterator()* of *ArrayList*. Fixing this should be relatively straight forward.

Warning 3 *bibtex.dom.BibtexFile.printBibtex(), line93:*
Algebraic Interpreter failed to compute a value.
term = next(hasNext(iterator(...).retval).state).retval

In other words, we need to compute the next element for the iterator which is really the first element of the linked list.

We add the axiom

forall l:ArrayList ***(Axiom 148)***
next(iterator(l).retval).retval == get(l,0).retval

There is already an axiom in the discovered specification that will help us to compute the first element of the linked list:

forall x0:Object ***(Axiom 124)***
get(add(newArrayList().state,x0).state,0).retval == x0

However, this axiom is not enough, since we need to do two more things: (i) we need to eliminate hasNext from the term, so that Axiom 148 can be used. (ii) we ned to move the get(l,0).retval inside the term, so that Axiom 124 can be used.

To achieve (i), we can use the following axiom:

forall it:Iterator hasNext(it).state==it ***(Axiom 149)***

Similarly to Axiom 147, which we discussed above, this axiom can be discovered if we fix our term generator.

To achieve (ii), we can use

forall l:ArrayList forall o1:Object forall o2:Object ***(Axiom 150)***
get(add(add(l,o1).state,o2).state,0).retval
==get(add(l,o1).state,0).retval

This is a nice solution because we can get by without using conditional axioms. This axiom could have been discovered if we had chosen larger term sizes and waited a little longer for the results of the specification discovery process.

The additions up to and including Axiom 150 eliminate Warning 3.

At this point, the first warning we encounter is:

Warning 4 *bibtex.dom.BibtexFile.printBibtex(), line92:*
Algebraic Interpreter failed to compute a value.
term = hasNext(next(iterator(...).retval).state).retval

We have specified what *hasNext* does already, but we need to create a situation in which Axiom 147 can be applied. In other words, we need to eliminate the *next* operation in the term. So the question is, what kind of state does *next* compute? *next* moves over the list step by step, so one intuition would be that the state (as far as the iterator is concerned) is equivalent to a state in which the first element in the list has been removed. So we should be able to get away with:

forall l:ArrayList ***(Axiom 151a)***
next(iterator(l).retval).state
==iterator(remove0(l,0).state).retval

Unfortunately, the left side of this axiom has a term size of 3 while the right size of this axiom has a term size of 4. This means that our rewriting strategy will not consider rewriting using this axiom from left to right. We could change the rewriting strategy, but it would have a global effect on our system (and it won't be an option to a user I guess). One way to still achieve the same effect that the axiom above is trying to achieve is to introduce a "hidden method", which does not correspond to a method in the datatype we want to model:

hidden method removeFirst is <java.util.ArrayList: void removeFirst()>

We can now rephrase the axiom as

forall l:ArrayList ***(Axiom 151)***
next(iterator(l).retval).state
==iterator(removeFirst(l).state).retval

Additionally, we need to make sure that removeFirst propagates in direction of the constructor of ArrayList and we need to specify the base case:

forall l:ArrayList ***(Axiom 152)***
forall o1:Object forall o2:Object
removeFirst(add(add(l,o1).state,o2).state).state
==add(removeFirst(add(l,o1).state).state,o2).state

forall o:Object ***(Axiom 153)***
removeFirst(add(ArrayList().state,o).state).state
==ArrayList().state

Notice that Axiom 153 is a trivial variation of the following base case axiom for get, which was discovered by the specification discovery tool:

forall x0:Object ***(Axiom 126)***
remove0(add(ArrayList().state,x0).state,0).state
== ArrayList().state

The fact that we have to introduce the hidden function *removeFirst* is unfortunate, especially since the specification discovery tool could have helped us otherwise - both with the base case in Axiom 126 and also (if we had used larger term sizes) with a variation of Axiom 152.

Maybe it was not such a good decision to rely on term sizes (or any similar function of a term) for deciding in which way axioms can be used as rewriting rules. If the user was allowed to simply specify the direction of a rewriting rule, he could have used the original version of the Axiom 151a. Given we implement the fixes for the term generator, Axiom 151a can be discovered, too. To make better use of the discovery tool, we are planning to reimplement our interpreter with a more flexible scheme, for example, axioms that use => instead of == are rewritten from left to right, no matter how term sizes change. Axioms with == use the term size metric (or other termination functions).

This is the only term for which interpretation still fails:

```
bibtex.dom.BibtexFile.printBibtex(), line92:
Algebraic Interpreter failed to compute a value.
term = hasNext(next(...iterator(add(...).state).retval).state).retval
```

Why does it fail? The number of add and next operations is equal. This means that we are asking for hasNext of an empty array list. This axiom takes care of it:

hasNext(iterator(ArrayList().state).retval).retval==false **(Axiom 154)**

Again, similarly to Axiom 147 and Axiom 149, this axiom could be discovered if we fix our term generator as described above.

No more warning messages.

2 Summary

We find that our algebraic specification discovery tool was useful in this particular case study. While we came across a number of limitations within our discovery tool, it seems that all of the limitations are relatively straight forward to address, and none are general limitations of our technique.

The following table shows how the improvements proposed in this text impact which axioms can be discovered. To discover an axiom, one has to implement all improvements marked with ×. "larger term sizes for generator" only involves changing the configuration.

Improvement	81	124	147	148	149	150	151	152	153	154
larger term sizes for generator						×		×		
term generation across algebra boundaries			×	×	×		×			×
different rewriting strategy							×	×	×	

For this case study, introducing conditional axioms will significantly reduce the number of axioms discovered.

We are currently conducting additional studies that will help us in determining which improvements are needed to make the discovery tool more effective.

References

[1] Johannes Henkel and Amer Diwan. Discovering algebraic specifications from Java classes. In Luca Cardelli, editor, *ECOOP 2003 - Object-Oriented Programming, 17th European Conference*, Darmstadt, July 2003. Springer.

[2] Johannes Henkel and Amer Diwan. A tool for writing and debugging algebraic specifications. In *International Conference on Software Engineering (ICSE)*, 2004. (accepted).

Appendix: Specification

```
specification ArrayListSpecification

class ArrayList is java.util.ArrayList
class Object is java.lang.Object
class Iterator is java.util.Iterator

method hashCode is <java.util.AbstractList: int hashCode()>
method addAll0 is <java.util.ArrayList: boolean addAll(int,java.util.Collection)>
method listIterator is <java.util.AbstractList: java.util.ListIterator listIterator()>
method indexOf is <java.util.ArrayList: int indexOf(java.lang.Object)>
method getClass is <java.lang.Object: java.lang.Class getClass()>
method get is <java.util.ArrayList: java.lang.Object get(int)>
method toString is <java.util.AbstractCollection: java.lang.String toString()>
method ArrayList is <java.util.ArrayList: void <init>()>
method toArray is <java.util.ArrayList: java.lang.Object[] toArray()>
method clone is <java.util.ArrayList: java.lang.Object clone()>
method add0 is <java.util.ArrayList: void add(int,java.lang.Object)>
method containsAll is <java.util.AbstractCollection: boolean containsAll(java.util.Collection)>
method retainAll is <java.util.AbstractCollection: boolean retainAll(java.util.Collection)>
method ensureCapacity is <java.util.ArrayList: void ensureCapacity(int)>
method listIterator0 is <java.util.AbstractList: java.util.ListIterator listIterator(int)>
method addAll is <java.util.ArrayList: boolean addAll(java.util.Collection)>
method clear is <java.util.ArrayList: void clear()>
method lastIndexOf is <java.util.ArrayList: int lastIndexOf(java.lang.Object)>
method size is <java.util.ArrayList: int size()>
method trimToSize is <java.util.ArrayList: void trimToSize()>
method add is <java.util.ArrayList: boolean add(java.lang.Object)>
method isEmpty is <java.util.ArrayList: boolean isEmpty()>
method contains is <java.util.ArrayList: boolean contains(java.lang.Object)>
method removeAll is <java.util.AbstractCollection: boolean removeAll(java.util.Collection)>
method subList is <java.util.AbstractList: java.util.List subList(int,int)>
method remove0 is <java.util.ArrayList: java.lang.Object remove(int)>
method remove is <java.util.AbstractCollection: boolean remove(java.lang.Object)>
method ArrayList0 is <java.util.ArrayList: void <init>(int)>
method toArray0 is <java.util.ArrayList: java.lang.Object[] toArray(java.lang.Object[])>
method iterator is <java.util.AbstractList: java.util.Iterator iterator()>
method intPlus is <edu.colorado.cs.heureka.terms.model.StaticOperators: int intPlus(int,int)>
method ArrayList1 is <java.util.ArrayList: void <init>(java.util.Collection)>
method equals is <java.util.AbstractList: boolean equals(java.lang.Object)>
method hasNext is <java.util.Iterator: boolean hasNext()>
method next is <java.util.Iterator: Object next()>

define ArrayList
define Iterator
```

```
// ===[1]======================
axiom
                hashCode(ArrayList().state).retval
                == 1

// ===[2]======================
axiom
                isEmpty(ArrayList().state).retval
                == true

// ===[3]======================
axiom
                size(ArrayList().state).retval
                == 0

// ===[4]======================
axiom
        forall x0:int
                ArrayList0(x0).state
                == ArrayList().state

// ===[5]======================
// Reason for commenting out: axiom contains object constant
// axiom
//  forall x0:ArrayList
//   getClass(x0).retval
//   == java.lang.Class@31789152

// ===[6]======================
axiom
        forall x0:ArrayList
                getClass(x0).state
                == x0

// ===[7]======================
axiom
        forall x0:ArrayList
                toString(x0).state
                == x0

// ===[8]======================
axiom
        forall x0:ArrayList
                hashCode(x0).state
                == x0

// ===[9]======================
axiom
        forall x0:ArrayList
```

```
                iterator(x0).state
                == x0

// ===[10]======================
axiom
        forall x0:ArrayList
                listIterator(x0).state
                == x0

// ===[11]======================
axiom
        forall x0:ArrayList
                isEmpty(x0).state
                == x0

// ===[12]======================
axiom
        forall x0:ArrayList
                size(x0).state
                == x0

// ===[13]======================
axiom
        forall x0:ArrayList
                clone(x0).state
                == x0

// ===[14]======================
axiom
        forall x0:ArrayList
                toArray(x0).state
                == x0

// ===[15]======================
axiom
        forall x0:ArrayList
                ArrayList1(x0).state
                == x0

// ===[16]======================
axiom
        forall x0:ArrayList
                clear(x0).state
                == ArrayList().state

// ===[17]======================
axiom
        forall x0:ArrayList
                trimToSize(x0).state
                == x0

// ===[18]======================
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:[LObject;
//   toArray0(ArrayList().state,x0).retval
//   == x0

// ===[19]======================
axiom
        forall x0:Object
                remove(ArrayList().state,x0).retval
                == false

// ===[20]======================
axiom
        forall x0:Object
                remove(ArrayList().state,x0).state
                == ArrayList().state

// ===[21]======================
axiom
        forall x0:Object
                contains(ArrayList().state,x0).retval
                == false

// ===[22]======================
axiom
        forall x0:Object
                indexOf(ArrayList().state,x0).retval
                == −1

// ===[23]======================
axiom
        forall x0:Object
                lastIndexOf(ArrayList().state,x0).retval
                == −1

// ===[24]======================
axiom
        forall x0:ArrayList
                containsAll(x0,x0).retval
                == true

// ===[25]======================
axiom
        forall x0:ArrayList
                removeAll(ArrayList().state,x0).retval
                == false

// ===[26]======================
axiom
        forall x0:ArrayList
                removeAll(ArrayList().state,x0).state
                == ArrayList().state

// ===[27]======================
axiom
        forall x0:ArrayList
                removeAll(x0,ArrayList().state).retval
                == false

// ===[28]======================
axiom
        forall x0:ArrayList
                removeAll(x0,ArrayList().state).state
                == x0

// ===[29]======================
axiom
        forall x0:ArrayList
                retainAll(ArrayList().state,x0).retval
                == false

// ===[30]======================
axiom
        forall x0:ArrayList
                retainAll(ArrayList().state,x0).state
                == ArrayList().state

// ===[31]======================
axiom
        forall x0:ArrayList
                retainAll(x0,x0).retval
                == false

// ===[32]======================
axiom
        forall x0:ArrayList
                retainAll(x0,x0).state
                == x0

// ===[33]======================
axiom
        forall x0:ArrayList
                listIterator0(x0,0).state
                == x0
```

```
// ===[34]======================
axiom
        forall x0:ArrayList
                addAll(ArrayList().state,x0).state
                == x0

// ===[35]======================
axiom
        forall x0:ArrayList
                addAll(x0,ArrayList().state).retval
                == false

// ===[36]======================
axiom
        forall x0:ArrayList
                addAll(x0,ArrayList().state).state
                == x0

// ===[37]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@10002314).retval
//   == [LObject;@10002314

// ===[38]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@10154474).retval
//   == [LObject;@10154474

// ===[39]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@11824466).retval
//   == [LObject;@11824466

// ===[40]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@14156733).retval
//   == [LObject;@14156733

// ===[41]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@15928050).retval
//   == [LObject;@15928050

// ===[42]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@16619703).retval
//   == [LObject;@16619703

// ===[43]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@1705819).retval
//   == [LObject;@1705819

// ===[44]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@17433056).retval
//   == [LObject;@17433056

// ===[45]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@21345323).retval
//   == [LObject;@21345323

// ===[46]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@21499351).retval
//   == [LObject;@21499351
// ===[47]======================
// Reason for commenting out: axiom contains object constantaxiom
// Reason for commenting out: arrays are not supported by interpreter
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@21637174).retval
//   == [LObject;@21637174

// ===[48]======================
// Reason for commenting out: axiom contains object constantaxiom
// Reason for commenting out: arrays are not supported by interpreter
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@21728762).retval
//   == [LObject;@21728762

// ===[49]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@22511636).retval
//   == [LObject;@22511636

// ===[50]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@22574727).retval
//   == [LObject;@22574727

// ===[51]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@23890203).retval
//   == [LObject;@23890203

// ===[52]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@24692682).retval
//   == [LObject;@24692682

// ===[53]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@24801042).retval
//   == [LObject;@24801042

// ===[54]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@25726653).retval
//   == [LObject;@25726653

// ===[55]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@26312551).retval
//   == [LObject;@26312551

// ===[56]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@26866310).retval
//   == [LObject;@26866310

// ===[57]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@27881100).retval
//   == [LObject;@27881100

// ===[58]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@27972023).retval
//   == [LObject;@27972023

// ===[59]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@28611518).retval
//    == [LObject;@28611518

// ===[60]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@29041878).retval
//    == [LObject;@29041878

// ===[61]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@2985799).retval
//    == [LObject;@2985799

// ===[62]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@30139050).retval
//    == [LObject;@30139050

// ===[63]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@30538114).retval
//    == [LObject;@30538114

// ===[64]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@3154634).retval
//    == [LObject;@3154634

// ===[65]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@32052912).retval
//    == [LObject;@32052912

// ===[66]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@3230294).retval
//    == [LObject;@3230294

// ===[67]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@32598094).retval
//    == [LObject;@32598094

// ===[68]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@3793194).retval
//    == [LObject;@3793194

// ===[69]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@4740071).retval
//    == [LObject;@4740071

// ===[70]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//    toArray0(x0,[LObject;@5368329).retval
//    == [LObject;@5368329

// ===[71]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@5845243).retval
//   == [LObject;@5845243

// ===[72]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@6126812).retval
//   == [LObject;@6126812

// ===[73]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@6605754).retval
//   == [LObject;@6605754

// ===[74]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@7535634).retval
//   == [LObject;@7535634

// ===[75]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@8996485).retval
//   == [LObject;@8996485

// ===[76]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//   toArray0(x0,[LObject;@9127705).retval
//   == [LObject;@9127705

// ===[77]======================
// Reason for commenting out: axiom contains object constant
// Reason for commenting out: arrays are not supported by interpreter
// axiom
//  forall x0:ArrayList
//  forall x1:[LObject;
//   toArray0(x0,x1).state
//  == x0

// ===[78]======================
axiom
        forall x0:ArrayList
        forall x1:int
                ensureCapacity(x0,x1).state
                == x0

// ===[79]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                equals(x0,x1).retval
                == false

// ===[80]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                equals(x0,x1).state
                == x0

// ===[81]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                add(x0,x1).retval
                == true

// ===[82]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                contains(x0,x1).state
                == x0

// ===[83]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                indexOf(x0,x1).state
                == x0

// ===[84]======================
axiom
        forall x0:ArrayList
        forall x1:Object
        lastIndexOf(x0,x1).state
        == x0

// ===[85]======================
axiom
    forall x0:ArrayList
    forall x1:ArrayList
        containsAll(x0,x1).state
        == x0

// ===[86]======================
// Reason for commenting out: axiom contains object constant
// axiom
//   hashCode(add(ArrayList().state,Object@18961126).state).retval
//   == 18961157

// ===[87]======================
// Reason for commenting out: axiom contains object constant
// axiom
//   hashCode(add(ArrayList().state,Object@31038029).state).retval
//   == 31038060

// ===[88]======================
// Reason for commenting out: axiom contains object constant
// axiom
//   hashCode(add(ArrayList().state,Object@5268497).state).retval
//   == 5268528

// ===[89]======================
axiom
    forall x0:Object
        size(add(ArrayList().state,x0).state).retval
        == 1

// ===[90]======================
axiom
    forall x0:ArrayList
        subList(x0,0,0).state
        == x0

// ===[91]======================
axiom
    forall x0:ArrayList
        addAll0(ArrayList().state,0,x0).state
        == x0

// ===[92]======================
axiom
    forall x0:ArrayList
```

```
                addAll0(x0,0,ArrayList().state).retval
                == false

// ===[93]======================
axiom
        forall x0:ArrayList
                addAll0(x0,0,ArrayList().state).state
                == x0

// ===[94]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                isEmpty(add(x0,x1).state).retval
                == false

// ===[95]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@18961126).state,Object@31038029).retval
//   == false

// ===[96]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@18961126).state,Object@5268497).retval
//   == false

// ===[97]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@31038029).state,Object@18961126).retval
//   == false

// ===[98]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@31038029).state,Object@5268497).retval
//   == false

// ===[99]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@5268497).state,Object@18961126).retval
//   == false

// ===[100]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@5268497).state,Object@31038029).retval
//   == false

// ===[101]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@18961126).state,Object@31038029).retval
//   == false

// ===[102]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@18961126).state,Object@5268497).retval
//   == false

// ===[103]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@31038029).state,Object@18961126).retval
//   == false

// ===[104]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@31038029).state,Object@5268497).retval
//   == false

// ===[105]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@5268497).state,Object@18961126).retval
//   == false

// ===[106]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   contains(add(ArrayList().state,Object@5268497).state,Object@31038029).retval
//   == false

// ===[107]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@18961126).state,Object@31038029).retval
//   == −1

// ===[108]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@18961126).state,Object@5268497).retval
//   == −1

// ===[109]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@31038029).state,Object@18961126).retval
//   == −1

// ===[110]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@31038029).state,Object@5268497).retval
//   == −1

// ===[111]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@5268497).state,Object@18961126).retval
//   == −1

// ===[112]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   indexOf(add(ArrayList().state,Object@5268497).state,Object@31038029).retval
//   == −1

// ===[113]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@18961126).state,Object@31038029).retval
//   == −1

// ===[114]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@18961126).state,Object@5268497).retval
//   == −1

// ===[115]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@31038029).state,Object@18961126).retval
//   == −1

// ===[116]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@31038029).state,Object@5268497).retval
//   == −1

// ===[117]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@5268497).state,Object@18961126).retval
//   == −1

// ===[118]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   lastIndexOf(add(ArrayList().state,Object@5268497).state,Object@31038029).retval
//   == −1

// ===[119]======================
axiom
        forall x0:Object
                containsAll(ArrayList().state,add(ArrayList().state,x0).state).retval
                == false

// ===[120]======================
axiom
        forall x0:Object
                remove(add(ArrayList().state,x0).state,x0).state
                == ArrayList().state
```

```
// ===[121]======================
axiom
        forall x0:Object
                retainAll(add(ArrayList().state,x0).state,ArrayList().state).retval
                == true

// ===[122]======================
axiom
        forall x0:Object
                indexOf(add(ArrayList().state,x0).state,x0).retval
                == 0

// ===[123]======================
axiom
        forall x0:Object
                lastIndexOf(add(ArrayList().state,x0).state,x0).retval
                == 0

// ===[124]======================
axiom
        forall x0:Object
                get(add(ArrayList().state,x0).state,0).retval
                == x0

// ===[125]======================
axiom
        forall x0:Object
                remove0(add(ArrayList().state,x0).state,0).retval
                == x0

// ===[126]======================
axiom
        forall x0:Object
                remove0(add(ArrayList().state,x0).state,0).state
                == ArrayList().state

// ===[127]======================
axiom
        forall x0:ArrayList
                containsAll(x0,addAll(x0,x0).state).retval
                == true

// ===[128]======================
axiom
        forall x0:ArrayList
                removeAll(addAll(x0,x0).state,x0).state
                == ArrayList().state

// ===[129]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                remove(add(x0,x1).state,x1).retval
                == true

// ===[130]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                addAll(x0,add(x0,x1).state).retval
                == true

// ===[131]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                contains(add(x0,x1).state,x1).retval
                == true

// ===[132]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@18961126).state,Object@31038029).state
//   == add(ArrayList().state,Object@18961126).state

// ===[133]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@18961126).state,Object@5268497).state
//   == add(ArrayList().state,Object@18961126).state

// ===[134]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@31038029).state,Object@18961126).state
//   == add(ArrayList().state,Object@31038029).state

// ===[135]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@31038029).state,Object@5268497).state
//   == add(ArrayList().state,Object@31038029).state

// ===[136]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@5268497).state,Object@18961126).state
//   == add(ArrayList().state,Object@5268497).state

// ===[137]======================
// Reason for commenting out: axiom contains object constant
// The condition equals(var1,var2).retval==false could have helped here
// axiom
//   remove(add(ArrayList().state,Object@5268497).state,Object@31038029).state
//   == add(ArrayList().state,Object@5268497).state

// ===[138]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                intPlus(size(add(x0,x1).state).retval,−1).retval
                == size(x0).retval

// ===[139]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                listIterator0(add(x0,x1).state,1).state
                == add(x0,x1).state

// ===[140]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                get(add(x0,x1).state,0).state
                == add(x0,x1).state

// ===[141]======================
axiom
        forall x0:ArrayList
        forall x1:Object
        forall x2:Object
                intPlus(size(add(add(x0,x1).state,x2).state).retval,−2).retval
                == size(x0).retval

// ===[142]======================
axiom
        forall x0:Object
                add0(ArrayList().state,0,x0).state
                == add(ArrayList().state,x0).state

// ===[143]======================
axiom
        forall x0:ArrayList
                addAll0(x0,0,x0).state
                == addAll(x0,x0).state

// ===[144]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                intPlus(size(x0).retval,1).retval
                == size(add(x0,x1).state).retval

// ===[145]======================
axiom
        forall x0:ArrayList
        forall x1:Object
                size(add(x0,x1).state).retval
                == intPlus(size(x0).retval,1).retval

// ===[146]======================
axiom
        forall x0:ArrayList
        forall x1:Object
        forall x2:Object
                size(add(x0,x1).state).retval
                == size(add(x0,x2).state).retval

// ===[147]======================
// Added by user
axiom
        forall x1:ArrayList
        forall x2:Object
                hasNext(iterator(add(x1,x2).state).retval).retval
                == true

// ===[148]======================
// Added by user
axiom
        forall l:ArrayList
                next(iterator(l).retval).retval
                ==get(l,0).retval

// ===[149]======================
// Added by user
axiom
        forall it:Iterator
                hasNext(it).state==it

// ===[150]======================
// Added by user
axiom
        forall l:ArrayList
        forall o1:Object
        forall o2:Object
                get(add(add(l,o1).state,o2).state,0).retval
                ==get(add(l,o1).state,0).retval

// ===[151]======================
// Added by user
axiom
        forall l:ArrayList
                next(iterator(l).retval).state
                ==iterator(removeFirst(l).state).retval

// ===[152]======================
// Added by user
axiom
        forall l:ArrayList
        forall o1:Object
        forall o2:Object
                removeFirst(add(add(l,o1).state,o2).state).state
                ==add(removeFirst(add(l,o1).state).state,o2).state


// ===[153]======================
// Added by user
axiom
        forall o:Object
                removeFirst(add(ArrayList().state,o).state).state
                ==ArrayList().state

// ===[154]======================
// Added by user
axiom
        hasNext(iterator(ArrayList().state).retval).retval==false
```