USER'S MANUAL

Deryaft version 1.0

Software Verification, Validation and Testing
Research Group at University of Texas at Austin

December, 2007

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	5/30/07	User's Manual Template and Checklist
Rev. 1	4/10/07	Conversion to WORD 2000 format

USER'S MANUAL

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

1.0 GENERAL INFORMATION

This section provides an introduction to the Deryaft tool.

1.1 System Overview

Deryaft is a tool that discovers representation invariants of structurally complex data manipulated by java programs. By structurally complex data we mean that the inputs are structural (e.g., represented with linked data structures) and must satisfy complex constraints that relate parts of the structure (e.g., acyclicity for linked data structures).

Given a small set of concrete structures, Deryaft analyzes their key characteristics to formulate local and global properties that the structures exhibit. For effective formulation of structural invariants, Deryaft focuses on graph properties and views the program heap as an edge-labeled graph.

Deryaft tool is based on a mathematically rigorous novel algorithm proposed in Deryaft Framework [1] for strongly type checked languages. The tool is currently available as a command line utility with limited number of options that will grow as the tool evolves.

At the moment tool accepts a file of serialized objects as inputs and processes them using Java Reflection APIs. The result of the analysis performed by the tool is a predicate Java function for the set of objects.

Currently, constructing examples for generation is as tedious as writing the predicate method itself but in the next major revision the processes will be improved with graphical user interface support.

1.2 Points of Contact

If you fail to find relevant information in this manual please visit http://deryaft.sourceforge.net there you will find references to related conference publications. We strongly recommend reading *Design of Deryaft* and *Implementation of Deryaft* reports. If you still have any further questions please feel free to contact the Deryaft development team. The contact information is also present at the website.

2.0 GETTING STARTED

2.0 GETTING STARTED

1. download binary distribution, from http://deryaft.sourceforge.net
2. extract the files to a directory (i.e., Deryaft-0.9)
   - `unzip -d Deryaft-0.9 Deryaft-0.9.zip`
3. cd Deryaft-0.9
4. try out some of our examples to make sure that everything is working:
   - `java -jar Deryaft.jar -g`
     (This command generates various examples data structures and serializes them in a file.)

   - `java -jar Deryaft.jar -d filename`
     (This command reads the file as input and performs detection on it.)

5. Checking Deryaft on your personal data is also very easy. You need to take the following steps
   - Write the Class definition, the class should implement serializable interface.
   - Construct representative examples of the structures, this means to make objects of the class and point data as you like.
   - Write the objects to **desiredFileName**
   - `java -jar Deryaft.jar -d` **`desiredFileName`**

3.0 Providing Deryaft Input Examples

3.0 PROVIDING DERYAFT INPUT EXAMPLES

The most demanding task for any user is to provide examples to Deryaft to run its analysis on. In principle, an example for Deryaft is a valid data configuration on the heap. In reality, it is a bunch of serialized objects in a file which Deryaft tool can recreate the same data configuration for its analysis.

3.1 Writing the Example Class

The user only need to supply the class definition for generation the representation invariants, hence *test-before-writing* code paradigm is not violated. The only difference in a class written for Deryaft and a normal class is that this class must implement the interface serializable and serialVersionUID

Here is an example test class:

```
import java.io.Serializable;


public class myTree implements Serializable{
	public static final long serialVersionUID = 9999;
	int NumofNodes;
	TreeNode root = new TreeNode();
	public myTree(){
		root = new TreeNode();
		root.nodeVal = 0;
		root.parent = null;
		NumofNodes = 0;
	}
};
```

Another important consideration is to also provide class definition of classes used by the example class. Hence we need to supply the class TreeNode in this case, which should also implement serializable.

```
import java.io.Serializable;

public class TreeNode implements Serializable{
	public static final long serialVersionUID = 1111;
	public TreeNode parent;
	public TreeNode left;
	public TreeNode right;
	public int nodeVal;
};
```

The final consideration for such a class is that no package is provided or `package deryaft` is used. This is to allow reflection API to access example data properly without generating access violation exception.

3.2 Generating the Example Objects and Serializing them

Once the example class has been written we need to actually specify its heap configurations of interest. The following code shows one heap configuration created and then stored for the class defined above.

```
myTree testTree = new myTree();
testTree.NumofNodes = 4;
testTree.root.nodeVal = 2;

testTree.root.left = new TreeNode();
testTree.root.left.parent = testTree.root;
testTree.root.left.nodeVal = 1;
testTree.root.left.left = null;
testTree.root.left.right = null;

testTree.root.right = new TreeNode();
testTree.root.right.parent = testTree.root;
testTree.root.right.nodeVal = 3;
testTree.root.right.left = null;
testTree.root.right.right = new TreeNode();
testTree.root.right.right.parent = testTree.root.right;
testTree.root.right.right.nodeVal = 4;
testTree.root.right.right.left = null;
testTree.root.right.right.right = null;

try{
      ObjectOutputStream objstream = new ObjectOutputStream(new
FileOutputStream("testTree.obj"));
      objstream.writeObject(testTree);
      objstream.close();
}catch(IOException e){
      e.printStackTrace();
}
```

3.3 Analyzing Examples with Deryaft

If you are using Deryaft tool executable JAR then you only need to make the following call:

- java -jar Deryaft.jar -d testTree.obj

If you are using the Deryaft API then using following code you can analyze the examples.

```
Vector<Object> vecObjs = readObjectFile("testTree.obj");

  for(int i=0;i<vecObjs.size();i++){

      deryaft.ExtractedProperties.extractInformation(vecObjs.get(i));
  }
```

We have provided many examples for your convenience which can be accessed by calling:

java -jar Deryaft.jar -g

Using the output object files as example inputs you can generate representative repOks.

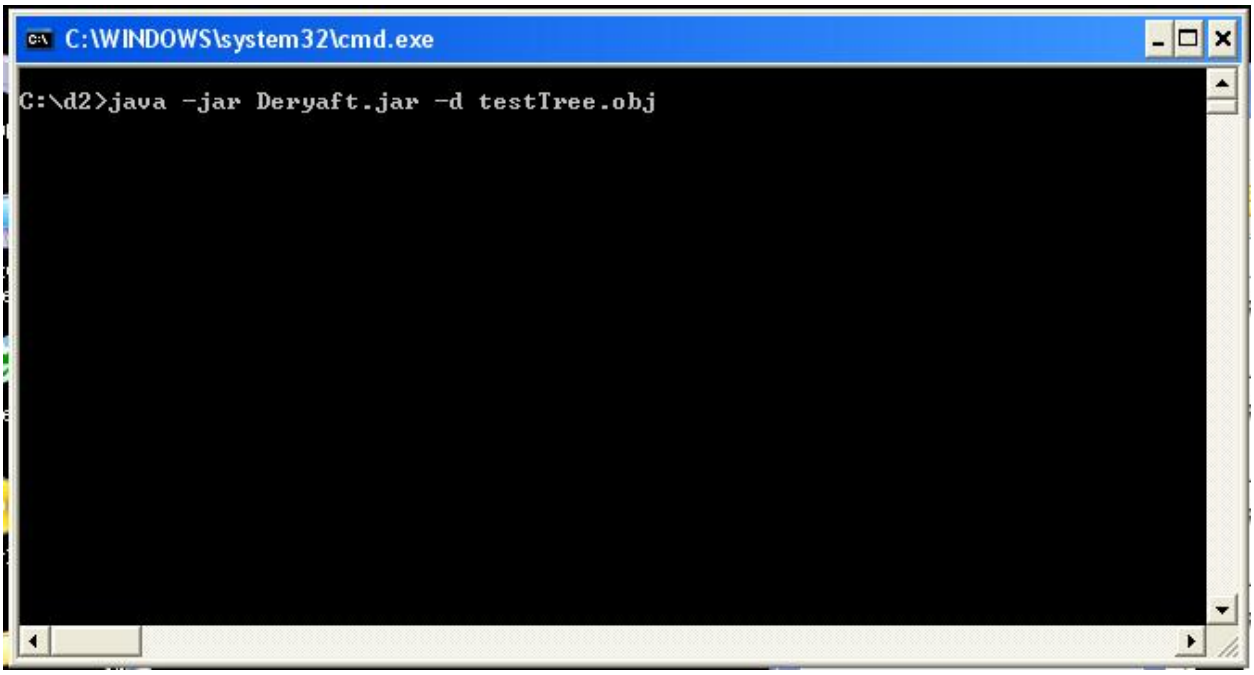
4.0 Understanding Deryaft Output

4.0 UNDERSTANDING DERYAFT OUTPUT

As described earlier Deryaft at the moment only provides a Java executable JAR for analyzing the code. Once an object has been serialized using the steps described in previous section we can run Deryaft.

4.1 Running Deryaft

If you do not already have the Deryaft executable jar available then we suggest that you download Deryaft bundle from http://deryaft.sourceforge.net and unzip it. Assuming it was unzipped in my folder called d2, running Deryaft is quite simple.



4.2 Deryaft Messages

When executing Deryaft prints its status to keep you informed as it analyzes the example objects. A sample execution run is given below, this execution depends on the known properties and known rules provided to Deryaft at the time of execution.

```
C:\d2>java -jar Deryaft.jar -d testTree.obj
_____________________________________________________

Attempting to read rules from the file: KnownRules
---------Printing Rules---------

---- unary ---
[Positive Rules]
acyclicity
maxOrder
minOrder
[Negative Rules]

---- linear ---
[Positive Rules]
```

```
arrayMaxBreadth
arrayCyclicity
arrayCompleteness
arrayMaxHeap
arrayBSTLeft
[Negative Rules]

---- intentionalName ---
[Positive Rules]
rootNull
isFirstLast
intentionalCyclicty
[Negative Rules]

---- acyclicity ---
[Positive Rules]
[Negative Rules]
maxOrder
minOrder
binary
doubleList

---- binary ---
[Positive Rules]
leftBinaryOrder
rightBinaryOrder
maxHeapProperty
minHeapProperty
maxBFSOrder
minBFSOrder
completeness
maxBalance
maxHeight
[Negative Rules]
unary
maxOrder
minOrder

_____________________________________________________

Attempting to read models from the file: KnownModels
---------Printing Rules---------

---- Tree ---
acyclicity (1.0 )
binary (1.0 )

---- Linked List ---
unary (1.0 )
acyclicity (1.0 )

---- Max Linked List ---
unary (1.0 )
acyclicity (1.0 )
maxOrder (1.0 )

---- Min Linked List ---
unary (1.0 )
minOrder (1.0 )

---- Binary Search Tree ---
binary (1.0 )
acyclicity (1.0 )
Order (1.0 )

----  MinHeap ---
binary (1.0 )
```

```
acyclicity (1.0 )
Completeness (1.0 )
MaxProperty (1.0 )

----  MaxHeap ---
binary (1.0 )
acyclicity (1.0 )
Completeness (1.0 )
MinProperty (1.0 )

---- AVL Tree ---
binary (1.0 )
acyclicity (1.0 )
Balance (1.0 )


_____________________________________________________

Attempting to read file: testTree.obj
Serializing into the a Vector
Self reference found  so assuming no special header probably or linear data structure
_____________________________________________
Going to test if the field [left] is core or not in the class [Node]
Number of unique nodes found = 5
Number of reachable unique nodes found = 2
Field [left] in class [Node] is core
_____________________________________________
Going to test if the field [right] is core or not in the class [Node]
Number of unique nodes found = 5
Number of reachable unique nodes found = 3
Field [right] in class [Node] is core
_____________________________________________
Going to test if the field [parent] is core or not in the class [Node]
Number of unique nodes found = 5
Number of reachable unique nodes found = 5
Field [parent] in class [Node] is NOT core
**********  PRINITNG REPORT **********************
_____________________________________________________
Core fields detected are
_____________________________________________________
(0) [left] belonging to class [Node]
(1) [right] belonging to class [Node]
_____________________________________________________
_____________________________________________________
Non Core fields detected are
_____________________________________________________
(0) [parent] belonging to class [Node]
_____________________________________________________
_____________________________________________________
value fields detected are
_____________________________________________________
(0) [value] belonging to class [int]
_____________________________________________________
Number of reachable references = 0
_____________________________________________________
Evaluating acyclicity Property
acyclicity found false
Attempting to add a new property
Going to remove from queue: acyclicity
Going to remove from queue: doubleList
Evaluating Binary Property
Attempting to add a new property
Going to remove from queue: binary
Evaluating leftBinaryOrder Property
leftBinaryOrder found true for value field [value]
Attempting to add a new property
Going to remove from queue: leftBinaryOrder
Evaluating rightBinaryOrder Property
```

```
rightBinaryOrder found false for value field [value]
Attempting to add a new property
Going to remove from queue: rightBinaryOrder
Evaluating maxHeapProperty Property
innerValue      = 8
innerChildValue = 7
Inner type = int
innerValue      = 7
innerChildValue = 4
Inner type = int
innerValue      = 7
innerChildValue = 10
Inner type = int
maxHeapProperty found false for value field [value]
Attempting to add a new property
Going to remove from queue: maxHeapProperty
Evaluating minHeapProperty Property
innerValue      = 8
innerChildValue = 7
Inner type = int
minHeapProperty found false for value field [value]
Attempting to add a new property
Going to remove from queue: minHeapProperty
Evaluating maxBFSOrder Property
8
7
maxBFSOrder found false for value field [value]
Attempting to add a new property
Going to remove from queue: maxBFSOrder
Evaluating minBFSOrder Property
8
7
9
minBFSOrder found false for value field [value]
Attempting to add a new property
Going to remove from queue: minBFSOrder
Evaluating completeness Property
completeness found true
Attempting to add a new property
Going to remove from queue: completeness
Evaluating maxBalance Property
maxBalance Property: 1
Attempting to add a new property
Going to remove from queue: maxBalance
Evaluating maxHeight Property
maxHeight Property: 2
Attempting to add a new property
Going to remove from queue: maxHeight
**************************************
PropertyName = acyclicity
numSamplesEncountered = 1
positiveSamples = 1
negativeSamples = 0
positiveConfidence = 1.0
negativeConfidence = 0.0

_____________________________________________
PropertyName = binary
numSamplesEncountered = 1
positiveSamples = 1
negativeSamples = 0
positiveConfidence = 1.0
negativeConfidence = 0.0

_____________________________________________
PropertyName = leftBinaryOrder
numSamplesEncountered = 1
positiveSamples = 1
negativeSamples = 0
```

```
positiveConfidence = 1.0
negativeConfidence = 0.0

_____________________________________________
PropertyName = rightBinaryOrder
numSamplesEncountered = 1
positiveSamples = 0
negativeSamples = 1
positiveConfidence = 0.0
negativeConfidence = 1.0

_____________________________________________
PropertyName = maxHeapProperty
numSamplesEncountered = 1
positiveSamples = 0
negativeSamples = 1
positiveConfidence = 0.0
negativeConfidence = 1.0

_____________________________________________
PropertyName = minHeapProperty
numSamplesEncountered = 1
positiveSamples = 0
negativeSamples = 1
positiveConfidence = 0.0
negativeConfidence = 1.0

_____________________________________________
PropertyName = maxBFSOrder
numSamplesEncountered = 1
positiveSamples = 0
negativeSamples = 1
positiveConfidence = 0.0
negativeConfidence = 1.0

_____________________________________________
PropertyName = minBFSOrder
numSamplesEncountered = 1
positiveSamples = 0
negativeSamples = 1
positiveConfidence = 0.0
negativeConfidence = 1.0

_____________________________________________
PropertyName = completeness
numSamplesEncountered = 1
positiveSamples = 1
negativeSamples = 0
positiveConfidence = 1.0
negativeConfidence = 0.0

_____________________________________________
PropertyName = maxBalance
numSamples = 1
maxIntegerProperty = 1
minIntegerProperty = 1
averageIntegerProperty = 1.0
allEqualIntegerProperty = true

_____________________________________________
PropertyName = maxHeight
numSamples = 1
maxIntegerProperty = 2
minIntegerProperty = 2
averageIntegerProperty = 2.0
allEqualIntegerProperty = true

_____________________________________________
Generating code for acylicity
Generating code for leftBinaryOrder
Generating code for completeness
Generating code for maxBalance

C:\d2>
```

4.3 Deryaft Output: repOK

After analyzing the objects, Deryaft outputs the hypothesis that hold in the form of a repOk. The output is a Java file with the same name as the input file with java extension. So for the serialized binary search tree input object file called “BinarySearchTree” , the output will be “BinarySearchTree.java”.

The contents of BinarySearchTree.java are given below. The predicate function repOK is a valid input for other test case generation tool such as Korat.

```
import java.util.Hashtable;
import java.util.Vector;
import java.util.ArrayList;

public class BinarySearchTree
{

public static boolean repOK(Node head)
{
 if(detectCyclesRecursiveFunction(head, new Hashtable())==false) return false;
 if(leftBinaryOrder(head)==false) return false;
 if(isComplete(head)==false) return false;
 if(evaluateMaxBalance(head,1)==false) return false;
 return true;
}


public static boolean detectCyclesRecursiveFunction(Node _Node, Hashtable hashTable)
{
  hashTable.put( _Node.hashCode(),  _Node);
    if( _Node.left != null)
    {
      if(hashTable.containsKey( _Node.left.hashCode()))         return true;
      else
      if(detectCyclesRecursiveFunction( _Node.left, hashTable)) return true;
    }
    if( _Node.right != null)
    {
      if(hashTable.containsKey( _Node.right.hashCode()))         return true;
      else
      if(detectCyclesRecursiveFunction( _Node.right, hashTable)) return true;
    }
  return false;
}


public static boolean detectLeftOrder(Node _Node)
{
 if(  _Node.left != null)
    {
     if( _Node.value < _Node.left.value)     return false;
     if(detectMaxHeap(  _Node.left) == false) return false;
    }
 if(  _Node.right != null)
    {
     if( _Node.value > _Node.right.value)     return false;
     if(detectLeftOrder(  _Node.right) == false) return false;
    }
 return true;
}


public static boolean isComplete(Node _Node)
{
```

```
 Vector<TreeElement> queue = new Vector<TreeElement>();
 queue.add(new TreeElement( _Node,0));
 TreeElement root = null;
 while(queue.size() != 0)
 {
  root = queue.remove(0);
  if(root.obj != null)
   {
    queue.add(new TreeElement(((Node)(root.obj)).left,root.level + 1));
    queue.add(new TreeElement(((Node)(root.obj)).right,root.level + 1));
   }
   else
    {
     for(int i=0;i<queue.size();i++)
     if(((Node)(queue.get(i).obj)) != null) return false;
    }
 }
return true;
}


public static boolean evaluateMaxBalance(Node _Node,int maxBalance)
{
 BalanceInformation balanceInformation = findMaxOffsetBalance(  _Node);
 if(balanceInformation.maxOffset <= maxBalance) return true;
else return false;
}

public static BalanceInformation findMaxOffsetBalance(Node _Node)
{
  Vector<BalanceInformation> balances = new Vector<BalanceInformation>();
  if( _Node == null) return new BalanceInformation(0,0);
    if( _Node.left != null)balances.add(findMaxOffsetBalance( _Node.left ));
    if( _Node.right != null)balances.add(findMaxOffsetBalance( _Node.right ));
  return computeNewBalanceInfo(balances);
}

}
```

5.0 Deryaft Rules and Models

5.0 DERYAFT RULES AND MODELS

Deryaft analysis is limited to the properties and models supplied to Deryaft by the user. It is impossible to hypothesize all possible properties and models over a high level program written in Java.

5.1 Known Rules

Deryaft Rules are listed as XML file and inform Deryaft which other properties are positive and which are negative example constraint for a property under consideration. The example file provided with the Deryaft bundle is listed below for reference. The file is always called “KnownRules”.

```
<KnownRules>
	<Rule unary>
		<Positive>
		<posistiveRule>acyclicity</posistiveRule>
		<posistiveRule>maxOrder</posistiveRule>
		<posistiveRule>minOrder</posistiveRule>
		</Positive>
	</Rule>
	<Rule linear>
		<Positive>
		<posistiveRule>arrayMaxBreadth</posistiveRule>
		<posistiveRule>arrayCyclicity</posistiveRule>
		<posistiveRule>arrayCompleteness</posistiveRule>
		<posistiveRule>arrayMaxHeap</posistiveRule>
		<posistiveRule>arrayBSTLeft</posistiveRule>
		</Positive>
	</Rule>

	<Rule intentionalName>
		<Positive>
		<posistiveRule>rootNull</posistiveRule>
		<posistiveRule>isFirstLast</posistiveRule>
		<posistiveRule>intentionalCyclicty</posistiveRule>
		</Positive>
	</Rule>
	<Rule acyclicity>
		<Negative>
		<negativeRule>maxOrder</negativeRule>
		<negativeRule>minOrder</negativeRule>
		<negativeRule>binary</negativeRule>
		<negativeRule>doubleList</negativeRule>
		</Negative>
	</Rule>
	<Rule binary>
		<Negative>
		<negativeRule>unary</negativeRule>
		<negativeRule>maxOrder</negativeRule>
		<negativeRule>minOrder</negativeRule>
		</Negative>

		<Positive>
		<posistiveRule>leftBinaryOrder</posistiveRule>
		<posistiveRule>rightBinaryOrder</posistiveRule>
		<posistiveRule>maxHeapProperty</posistiveRule>
		<posistiveRule>minHeapProperty</posistiveRule>
		<posistiveRule>maxBFSOrder</posistiveRule>
		<posistiveRule>minBFSOrder</posistiveRule>
		<posistiveRule>completeness</posistiveRule>
		<posistiveRule>maxBalance</posistiveRule>
		<posistiveRule>maxHeight</posistiveRule>

		</Positive>

	</Rule>


</KnownRules>
```

5.2 Known Models

A unique data structure is represented as a known model and this file is used to infer true model from the examples.

```
<KnownModels>
<Model Tree>
<property acyclicity> 1 </property>
<property binary> 1 </property>
</Model>

<Model Linked List>
<property unary> 1 </property>
<property acyclicity> 1 </property>
</Model>

<Model Max Linked List>
<property unary> 1 </property>
<property acyclicity> 1 </property>
<property maxOrder> 1 </property>
</Model>

<Model Min Linked List>
<property unary> 1 </property>
<property minOrder> 1 </property>
</Model>



<Model Binary Search Tree>
<property binary> 1 </property>
<property acyclicity> 1 </property>
<property Order> 1 </property>
</Model>

<Model  MinHeap>
<property binary> 1 </property>
<property acyclicity> 1 </property>
<property Completeness> 1 </property>
<property MaxProperty> 1 </property>
</Model>

<Model  MaxHeap>
<property binary> 1 </property>
<property acyclicity> 1 </property>
<property Completeness> 1 </property>
<property MinProperty> 1 </property>
</Model>

<Model AVL Tree>
<property binary> 1 </property>
<property acyclicity> 1 </property>
<property Balance> 1 </property>
</Model>

</KnownModels>
```

6.0 Deryaft API

6.0 DERYAFT API

For the users that want to contribute to Deryaft development we have made the source of Deryaft available via SVN at Deryaft project home on sourceforge.net. Below we describe the major classes of Deryaft and their functionality.

6.1 Deryaft Major Classes

The principle classes of Deryaft source are:

- Interface Classes
  - Driver
  - DetectionEngine
- Analysis
  - ExtractProperties
  - ExtractedProperties
  - KnownRules
  - KnownModels
  - ModelProperty
  - DSProperty
- Utility
  - XML parsing classes
  - DataStructure Generation classes

6.2 Interface Classes

These classes provide entry and exit points to Deryaft analysis classes. The DetectionEngine class is the one used for running the actual application while Driver class provides an example of using the Deryaft API.

6.3 Analysis Classes

The core field analysis which enables the rest of Deryaft analysis is performed in ExtractedProperties class. Normally a user will only need to call ExtractedProperties.extractInformation(myObject). The ExtractedProperties after the core field analysis calls most of the functionality from ExtractProperties. Once the properties have been extracted KnownRules and KnownModels is used to infer valid invariants.