Supplemental Material for "ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming"

Yuan Yuan and Wolfgang Banzhaf

Abstract—This supplemental material first describes the correct patches generated by ARJA for 18 real bugs from Defects4J. We would like to explain why these patches are correct. Then, for each of the 10 repair algorithms in comparison, we list which bugs in Defects4J are repaired (in terms of satisfying the test suite).

Index Terms—Defects4J, Patch correctness, ARJA

1 Correct Patches

1.1 Correct Patch for Chart3

The patches created by a human developer and ARJA for Chart3 are shown in Fig. 1.

To understand the bug, we should first illustrate the following points:

1) The `clone` method (line 4–5 in Fig. 1(a)) will assign the values of `this.minY` and `this.maxY` to `copy.minY` and `copy.maxY` respectively.
2) When `copy` invokes the method `add(clone)` (line 18 in Fig. 1(a)), it indeed invokes `add(clone, true)`. This method invocation will update the values of `copy.minY` and `copy.maxY` according to their current values and the double value of `clone`.
3) The line 8 in Fig. 1(a) makes `copy.data` an empty list. So `copy.minY` and `copy.maxY` should be set to the initial value (i.e., `Double.NaN`). Otherwise, according to the second point, there is a mistake in the updating of `copy.minY` and `copy.maxY` in the method `add(clone)` because `copy.minY` and `copy.maxY` have wrong initial values (i.e., `this.minY` and `this.maxY` respectively, see the first point).
4) The method `findBoundsByIteration` is to update the values of `minY` and `maxY` of a `TimeSeries` object. But it conducts the updating of `minY` and `maxY` from scratch rather than according to their current values.The code of the method `findBoundsByIteration` is shown as follows. This method first assigns the initial values to `minY` and `maxY`, then scans each element in the list `data` to update `minY` and `maxY`.

```
1 private void findBoundsByIteration() {
2   this.minY = Double.NaN;
3   this.maxY = Double.NaN;
4   Iterator iterator =
        this.data.iterator();
5   while (iterator.hasNext()) {
6     TimeSeriesDataItem item =
          (TimeSeriesDataItem)
          iterator.next();
7     updateBoundsForAddedItem(item);
8    }
9 }
```

To fix this bug, the human-written patch just makes the initial values of `copy.minY` and `copy.maxY` correct by inserting two assignment statements as shown in Fig. 1(a).

The patch generated by ARJA modifies the method `add(TimeSeriesDataItem item, boolean notify)`. It inserts `findBoundsByIteration()` before line 9 in Fig. 1(b). The effect is that when `copy` invokes `add` (line 18 in Fig. 1(a)), we always first recompute `copy.minY` and `copy.maxY` according to the current `copy.data`. So the values of `copy.minY` and `copy.maxY` will be always correct once `copy.add(clone)` is executed (line 18 in Fig. 1(a)). If `copy.add(clone)` is never executed (i.e., `this.data.size()` is 0), then `this.minY` and `this.maxY` are both equal to `Double.NaN`. So the `copy.minY` and `copy.maxY` will already be assigned to the correct initial value (i.e., `Double.NaN`) by executing `TimeSeries copy = (TimeSeries)super.clone();`.

Moreover, in Fig. 1(b), the modification of the method `add(TimeSeriesDataItem item, boolean notify)` will not influence its original functionality, but will decrease the efficiency. Because the patched program by ARJA does not trust the correctness of the current values of `minY` and `maxY`, it always first recompute them before adding a new element.

However, in terms of making the buggy program functionally correct, there is no difference between the patch generated by ARJA and the human-written patch.

Y. Yuan and W. Banzhaf are with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: yyuan@msu.edu; banzhafw@msu.edu).
Manuscript received xxx; revised yyy.

```
// TimeSeries.java
public TimeSeries createCopy(int start,
    int end) {
...
  TimeSeries copy = (TimeSeries)
    super.clone();
+ copy.minY = Double.NaN;
+ copy.maxY = Double.NaN;
  copy.data = new java.util.ArrayList();
  if (this.data.size() > 0) {
    for (int index = start; index <= end;
        index++) {
      TimeSeriesDataItem item
                  = (TimeSeriesDataItem)
                      this.data.get(index);
      TimeSeriesDataItem clone =
          (TimeSeriesDataItem) item.clone();
      try {
          copy.add(clone);
      }
      catch (SeriesException e) {
          e.printStackTrace();
      }
    }
  }
  return copy;
}
```

(a) Human-written patch

```
// TimeSeries.java
 public void add(TimeSeriesDataItem item,
     boolean notify) {
   if (item == null) {
     throw new
         IllegalArgumentException("Null
       'item' argument.");
   }
+  findBoundsByIteration();
   item = (TimeSeriesDataItem)
     item.clone();
 ...
 }
```

(b) Correct patch generated by ARJA

Fig. 1. Patches created by a human developer and ARJA for the bug Chart3.

1.2 Correct Patch for Chart5

The patches created by a human developer and ARJA for Chart5 are shown in Fig. 2.

To understand this bug, we should first know the functionality of the method `addOrUpdate(Number x, Number y)`. According to the program specification and the human-written patch, The only difference between `addOrUpdate` and `add` lies in that `addOrUpdate` not only executes the functionality of `add` but also saves the original data that is updated by the functionality of `add` and then returns the saved data (i.e., `overwritten`). Note that `this.data` is updated only when `index >= 0` and `this.allowDuplicateXValues` is false. So the buggy program does the right thing when the updating occurs. However, when the updating does not happen, the buggy program does not work properly (i.e., in the `else` block in Fig. 2(b)). Recalling that except the updating procedure, there is no difference in functionality between the methods. `addOrUpdate` and `add`. The patch by ARJA just executes `add(x, y, true);` in the `else` block to make the patched program functionally correct. Moreover have confirmed that the statements (except `return overwritten;`) after line 20 in Fig. 2(b) have no effect when the updating does not happen.

```
1  // XYSeries.java
2  public XYDataItem addOrUpdate(Number x,
       Number y) {
3  ...
4  + if (this.allowDuplicateXValues) {
5  +  add(x, y);
6  +  return null;
7  + }
8    XYDataItem overwritten = null;
9    int index = indexOf(x);
10 + if (index >= 0) {
11 - if (index >= 0 &&
12 -    !this.allowDuplicateXValues) {
13       XYDataItem existing = (XYDataItem)
14           this.data.get(index);
15 ...
16   }
17   else {
18     if (this.autoSort) {
19       this.data.add(-index - 1, new
20           XYDataItem(x, y));
21     }
22     else {
23       this.data.add(new XYDataItem(x, y));
24     }
25 ...
26   }
27 ...
28   return overwritten;
29 }
```

(a) Human-written patch

```
1  // XYSeries.java
2  public XYDataItem addOrUpdate(Number x,
       Number y) {
3  ...
4    XYDataItem overwritten = null;
5    int index = indexOf(x);
6    if (index >= 0 &&
7      !this.allowDuplicateXValues) {
8       XYDataItem existing = (XYDataItem)
9           this.data.get(index);
10 ...
11   }
12   else {
13 -   if (this.autoSort) {
14 -     this.data.add(-index - 1, new
15 -         XYDataItem(x, y));
16 -   }
17 -   else {
18 -     this.data.add(new XYDataItem(x, y));
19 -   }
20 +   add(x, y, true);
21 ...
22   }
23 ...
24   return overwritten;
25 }
```

(b) Correct patch generated by ARJA

Fig. 2. Patches created by a human developer and ARJA for the bug Chart5.

1.3 Correct Patch for Chart12

The patches created by a human developer and ARJA for Chart12 are shown in Fig. 3.

Both human-written patch and the patch found by ARJA use a fix ingredient: `setDataset(dataset);`. The procedure of the method `setDataset` is as follows:

```
1  public void setDataset(CategoryDataset
       dataset) {
2    if (this.dataset != null) {
3      this.dataset.removeChangeListener(this);
4    }
5    this.dataset = dataset;
6    if (dataset != null) {
7      setDatasetGroup(dataset.getGroup());
8      dataset.addChangeListener(this);
9    }
10   datasetChanged(new
11     DatasetChangeEvent(this, dataset));
12 }
```

Note that when calling the method `setDataset` in the constructor `MultiplePiePlot(CategoryDataset dataset)`, `this.dataset` is always `null` initially. So in the patched program by human, `setDataset(dataset);` indeed executes lines 5–11 in the method `setDataset` and lines 2–4 have no effect.

If in the patched program by ARJA, the lines 2–4 in the method `setDataset` also have no effect, then the patch generated by ARJA will be semantically equivalent to the human-written patch and will be correct. First, in Fig. 3(b), if `dataset` is `null` , then `this.dataset` is null since `dataset` is assigned to `this.dataset` in line 4, and thus the lines 2–4 in method `setDataset` have no effect in the patched program by ARJA in this situation. Second, in Fig. 3(b), if `dataset` is not `null`, then `this.dataset` is not null. In this situation, when the patched program by ARJA calls `setDataset`, the line 3 in the method `setDataset` will be executed. However, `this.dataset` is now pointing to `dataset` and the registered listeners of `dataset` cannot contain the new object `this` that is to be constructed, so although the line 3 (removing the listener) in the method `setDataset` is executed in this situation, it indeed does nothing. Based on the above two situations, the lines 2–4 in method `setDataset` also have no effect in the patched program by ARJA, so the patch generated by ARJA is correct.

```
1  // MultiplePiePlot.java
2  public MultiplePiePlot(CategoryDataset
       dataset) {
3      super();
4  -   this.dataset = dataset;
5  +   setDataset(dataset);
6      PiePlot piePlot = new PiePlot(null);
7      this.pieChart = new
           JFreeChart(piePlot);
8      this.pieChart.removeLegend();
9      this.dataExtractOrder =
           TableOrder.BY_COLUMN;
10 ...
11 }
```

(a) Human-written patch

```
1   // MultiplePiePlot.java
2   public MultiplePiePlot(CategoryDataset
        dataset) {
3     super();
4     this.dataset = dataset;
5     PiePlot piePlot = new PiePlot(null);
6     this.pieChart = new JFreeChart(piePlot);
7     this.pieChart.removeLegend();
8   +   setDataset(dataset);
9     this.dataExtractOrder =
10      TableOrder.BY_COLUMN;
11  ...
12  }
```

(b) Correct patch generated by ARJA

Fig. 3. Patches created by a human developer and ARJA for the bug Chart12.

1.4 Correct Patch for Time15

The human-written patch and the correct patch generated by ARJA are shown in Fig. 4. Please refer to our paper for the explanation of the correctness.

```
// FieldUtils.java
public static long safeMultiply(long val1,
    int val2) {
  switch (val2) {
  case -1:
+  if (val1 == Long.MIN_VALUE) {
+    throw new
+    ArithmeticException("...overflows");
+  }
   return -val1;
  case 0: return 0L;
  case 1: return val1;
  }
  long total = val1 * val2;
  if (total / val2 != val1) {
    throw new
    ArithmeticException("...overflows");
  }
  return total;
}
```

(a) Human-written patch

```
// FieldUtils.java
public static long safeMultiply(long val1,
    int val2) {
  switch (val2) {
  case -1:
- return -val1;
+ break;
  case 0: return 0L;
  case 1: return val1;
  }
  long total = val1 * val2;
- if (total / val2 != val1) {
-   throw new
-     ArithmeticException("...overflows");
- }
+ if (total / val2 != val1 || val1 ==
+   Long.MIN_VALUE && val2 == -1 || val2 ==
+    Long.MIN_VALUE && val1 == -1) {
+   throw new
+    ArithmeticException("...overflows");
+ }
  return total;
}
```

(b) Correct patch generated by ARJA

Fig. 4. Patches created by a human developer and ARJA for the bug Time15.

1.5 Correct Patch for Lang20

The human-written patch and the correct patch generated by ARJA are shown in Fig. 5. Please refer to our paper for the explanation of the correctness.

```
// StringUtils.java
public static String join(Object[] array,
    char separator, int startIndex, int
    endIndex) {
  ...
- StringBuilder buf = new
-    StringBuilder((array[startIndex] ==
-      null ? 16 : array[startIndex]
-        .toString().length()) + 1);
+ StringBuilder buf = new
+    StringBuilder(noOfItems * 16);
  ...
}
public static String join(Object[] array,
    String separator, int startIndex, int
    endIndex) {
  ...
- StringBuilder buf = new
-    StringBuilder((array[startIndex] ==
-      null ? 16 : array[startIndex]
-        .toString().length()) +
-         separator.length());
+ StringBuilder buf = new
+    StringBuilder(noOfItems * 16);
  ...
}
```

(a) Human-written patch

```
// StringUtils.java
public static String join(Object[] array,
    char separator, int startIndex, int
    endIndex) {
  ...
- StringBuilder buf = new
-    StringBuilder((array[startIndex] ==
-      null ? 16 : array[startIndex]
-        .toString().length()) + 1);
+ StringBuilder buf = new
+    StringBuilder(256);
  ...
}
public static String join(Object[] array,
    String separator, int startIndex, int
    endIndex) {
  ...
- StringBuilder buf = new
-    StringBuilder((array[startIndex] ==
-      null ? 16 : array[startIndex]
-        .toString().length()) +
-         separator.length());
+ StringBuilder buf = new
+    StringBuilder(256);
  ...
}
```

(b) Correct patch generated by ARJA

Fig. 5. Patches created by a human developer and ARJA for the bug Lang20.

1.6 Correct Patch for Lang35

The human-written patch and the correct patch generated by ARJA are shown in Fig. 6. Please refer to our paper for the explanation of the correctness.

```
// ArrayUtils.java
public static <T> T[] add(T[] array, T
    element) {
...
-  type = Object.class;
+  throw new IllegalArgumentException
+    ("Arguments cannot both be null");
...
}
public static <T> T[] add(T[] array, int
    index, T element) {
...
-  return (T[]) new Object[] { null };
+  throw new IllegalArgumentException
+    ("Arguments cannot both be null");
...
}
```

(a) Human-written patch

```
// ArrayUtils.java
public static <T> T[] add(T[] array, T
    element) {
...
-  type = Object.class;
+  throw new IllegalArgumentException
+    ("The Integer did not match any ...");
...
}
public static <T> T[] add(T[] array, int
    index, T element) {
...
-  return (T[]) new Object[] { null };
+  throw new IllegalArgumentException
+    ("The Integer did not match any ...");
...
}
```

(b) Correct patch generated by ARJA

Fig. 6. Patches created by a human developer and ARJA for the bug Lang35.

1.7 Correct Patch for Lang43

The correct patch generated by ARJA is shown in Fig. 7. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

1.8 Correct Patch for Lang45

The human-written patch and the correct patch generated by ARJA are shown in Fig. 8.

To understand why the patch by ARJA is correct, we first want to illustrate the following points:

1) The bug occurs when `lower > str.length()`. In this situation, `lower` will be assigned to `upper` (lines 11–13 in Fig. 8(a) and lines 8–10 in Fig. 8(b)) and thus `upper` will be larger than `str.length()`, which triggers the bug.
2) `lower` is only used actually in the statement `int index = StringUtils.indexOf(str, "", lower);`. Moreover, we find that if the parameter `str` is fixed, the method `StringUtils.indexOf` will always return the same results when `lower >= str.length()`
3) According to the program specification, the meaningful input of `upper` is -1 (if no limit is desired) or a non-negative integer.

Considering the first and second points, we can know that there is no problem with the value of `lower` before executing line 15 in Fig. 8(b), and we can fix the bug correctly just by reseting `upper` to the correct value (i.e., `str.length()`) when `lower > str.length()`. The patch by ARJA does this by inserting the `if` statement as shown in Fig. 8(b). The condition `upper == -1` would not happen in the inserted statement considering the third point mentioned above. So the patch by ARJA just do the right thing to make the patched program functionally correct.

```
// ExtendedMessageFormat.java
 private StringBuffer
     appendQuotedString(...) {
...
   if (escapingOn && c[start] == QUOTE) {
+    next(pos);
     return appendTo == null ? null :
         appendTo.append(QUOTE);
   }
}
```

Fig. 7. Correct patch found by ARJA for the bug Lang43.

```
// WordUtils.java
public static String abbreviate(String
    str, int lower, int upper, String
    appendToEnd) {
...
+ if (lower > str.length()) {
+   lower = str.length();
+ }
  if (upper == -1 ||
    upper > str.length()) {
    upper = str.length();
  }
  if (upper < lower) {
    upper = lower;
  }
  StringBuffer result = new StringBuffer();
  int index = StringUtils.indexOf(str, " ",
    lower);
...
}
```

(a) Human-written patch

```
// WordUtils.java
public static String abbreviate(String
    str, int lower, int upper, String
    appendToEnd) {
...
  if (upper == -1 ||
    upper > str.length()) {
    upper = str.length();
  }
  if (upper < lower) {
    upper = lower;
  }
+ if (upper == -1 ||
+   upper > str.length()) {
+    upper = str.length();
+ }
  StringBuffer result = new StringBuffer();
  int index = StringUtils.indexOf(str, " ",
    lower);
...
}
```

(b) Correct patch generated by ARJA

Fig. 8. Patches created by a human developer and ARJA for the bug Lang45.

1.9 Correct Patch for Math5

The correct patch generated by ARJA is shown in Fig. 9. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

1.10 Correct Patch for Math22

The correct patch generated by ARJA is shown in Fig. 10. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

```
// Complex.java
public Complex reciprocal() {
...
  if (real == 0.0 && imaginary == 0.0) {
-   return NaN;
+   return INF;
  }
...
}
```

Fig. 9. Correct patch found by ARJA for the bug Math5.

```
// FDistribution.java
public boolean
   isSupportLowerBoundInclusive() {
-   return true;
+   return false;
}
// UniformRealDistribution.java
public boolean
   isSupportUpperBoundInclusive() {
-   return false;
+   return true;
}
```

Fig. 10. Correct patch found by ARJA for the bug Math22.

1.11 Correct Patch for Math39

The patches created by a human developer and ARJA for Math39 are shown in Fig. 11.

First the `if` statements inserted by a developer (lines 5–13 in Fig. 11(a)) and by ARJA (lines 6–10 in Fig. 11(b)) are semantically equivalent. Second, we find that `stages` is always larger than 1 in the program, so the inserted `if` statement by ARJA can always be executed. Third, we find that line 9 in Fig. 11(b) can only be executed at most once.

Considering the three factors, for the `if` statement inserted by ARJA, there is indeed no semantic difference between the two edits: 1) inserting it before line 5 in Fig. 11(b) and 2) inserting it before line 11 in Fig. 11(b). Thus, the patch generated by ARJA is semantically equivalent to the human-written patch and is thus correct.

1.12 Correct Patch for Math50

The correct patch generated by ARJA is shown in Fig. 12. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

1.13 Correct Patch for Math53

The correct patch generated by ARJA is shown in Fig. 13. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

1.14 Correct Patch for Math58

The human-written patch and the correct patch generated by ARJA are shown in Fig. 14. Please refer to our paper for the explanation of the correctness.

1.15 Correct Patch for Math70

The correct patch generated by ARJA is shown in Fig. 15. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

```
1  // EmbeddedRungeKuttaIntegrator.java
2  public void integrate(final
       ExpandableStatefulODE equations, final
       double t)
3       throws MathIllegalStateException,
           MathIllegalArgumentException {
4  ...
5    stepSize = hNew;
6  + if (forward) {
7  +   if (stepStart + stepSize >= t) {
8  +     stepSize = t - stepStart;
9  +   }
10 + } else {
11 +   if (stepStart + stepSize <= t) {
12 +     stepSize = t - stepStart;
13 +   }
14 + }
15   for (int k = 1; k < stages; ++k) {
16     for (int j = 0; j < y0.length; ++j) {
17       double sum = a[k-1][0] * yDotK[0][j];
18       for (int l = 1; l < k; ++l) {
19         sum += a[k-1][l] * yDotK[l][j];
20       }
21       yTmp[j] = y[j] + stepSize * sum;
22     }
23   }
24 ...
25 }
```

(a) Human-written patch

```
1  // EmbeddedRungeKuttaIntegrator.java
2  public void integrate(final
       ExpandableStatefulODE equations, final
       double t)
3       throws MathIllegalStateException,
           MathIllegalArgumentException {
4  ...
5    stepSize = hNew;
6    for (int k = 1; k < stages; ++k) {
7  +   if ((forward && (stepStart + stepSize >
8  +   t)) || ((!forward) && (stepStart +
9  +     stepSize < t))) {
10 +     stepSize = t - stepStart;
11 +   }
12     for (int j = 0; j < y0.length; ++j) {
13       double sum = a[k-1][0] * yDotK[0][j];
14       for (int l = 1; l < k; ++l) {
15         sum += a[k-1][l] * yDotK[l][j];
16       }
17       yTmp[j] = y[j] + stepSize * sum;
18     }
19   }
20 ...
21 }
```

(b) Correct patch generated by ARJA

Fig. 11. Patches created by a human developer and ARJA for the bug Math39.

```
// BaseSecantSolver.java
protected final double doSolve() {
...
   case REGULA_FALSI:
-    if (x == x1) {
-      x0 = 0.5 * (x0 + x1 -
-    FastMath.max(rtol * FastMath.abs(x1),
-       atol));
-      f0 = computeObjectiveValue(x0);
-    }
     break;
...
}
```

Fig. 12. Correct patch found by ARJA for the bug Math50.

```
// Complex.java
public Complex add(Complex rhs) throws
    NullArgumentException {
  MathUtils.checkNotNull(rhs);
+ if (isNaN || rhs.isNaN) {
+   return NaN;
+ }
  return createComplex(real +
    hs.getReal(),
    imaginary + rhs.getImaginary());
}
```

Fig. 13. Correct patch found by ARJA for the bug Math53.

```
// GaussianFitter.java
public double[] fit() {
  final double[] guess = (new
    ParameterGuesser(getObservations()))
    .guess();
- return fit(new Gaussian.Parametric(),
-   guess);
+ return fit(guess);
}
```

(a) Human-written patch

```
// GaussianFitter.java
public double[] fit() {
  final double[] guess = (new
    ParameterGuesser(getObservations()))
      .guess();
- return fit(new Gaussian.Parametric(),
-     guess);
+ return fit((new
+   ParameterGuesser(getObservations()))
+     .guess());
}
```

(b) Correct patch generated by ARJA

Fig. 14. Patches created by a human developer and ARJA for the bug Math58.

```
// BisectionSolver.java
public double solve(final
    UnivariateRealFunction f, double min,
    double max, double initial)
       throws
           MaxIterationsExceededException,
           FunctionEvaluationException {
-   return solve(min, max);
+   return solve(f, min, max);
}
```

Fig. 15. Correct patch found by ARJA for the bug Math70.

1.16 Correct Patch for Math73

The human-written patch and the correct patch generated by ARJA are shown in Fig. 16.

The meaning of the human-written patch is very clear. The patch found by ARJA uses an ingredient statement: `verifyBracketing(min,max,f);`. Note that `f.value(max)` is equal to `yMax` and `f.value(min)` is equal to `yMin`. According to our analysis, this statement verifies the following two conditions:

1) `min < max` is true.
2) `((yMax > 0 && yMin < 0)|| (yMax < 0 && yMin > 0))` is true.

If any of the two conditions is not satisfied, the `createIllegalArgumentException` will be thrown. The first condition has been verified by `verifySequence(min, initial, max);` (line 4 in Fig. 16(b)). So when the program reaches line 10 in Fig. 16(b), the first condition must have already been satisfied.

So if we can confirm that the second condition is semantically equivalent to the condition:
`yMin * yMax <= 0` is true,
then the patch generated by ARJA is semantically equivalent to the human-written patch. We find that if `yMin` or `yMax` is equal to 0, the method `solve` will be returned before line 4 in Fig. 16(a) (line 6 in Fig. 16(b)). So when the patched program executes line 10 in Fig. 16(b), neither `yMax` nor `yMin` can be equal to 0. Therefore, the second condition is just semantically equivalent to the condition:
`yMin * yMax <= 0` is true,
and the patch by ARJA is correct.

```
1  // BrentSolver.java
2  public double solve(final
       UnivariateRealFunction f, final double
       min, final double max, final double
       initial) throws
       MaxIterationsExceededException,
       FunctionEvaluationException {
3  ...
4    if (yInitial * yMax < 0) {
5      return solve(f, initial, yInitial, max,
6       yMax, initial, yInitial);
7    }
8  + if (yMin * yMax > 0) {
9  +   throw MathRuntimeException.
10 +    createIllegalArgumentException(
11 +              NON_BRACKETING_MESSAGE, min,
12 +       max, yMin, yMax);
13 + }
14   return solve(f, min, yMin, max, yMax,
15     initial, yInitial);
16 }
```

(a) Human-written patch

```
1  // BrentSolver.java
2  public double solve(final
       UnivariateRealFunction f, final double
       min, final double max, final double
       initial) throws
       MaxIterationsExceededException,
       FunctionEvaluationException {
3  ...
4    verifySequence(min, initial, max);
5  ...
6    if (yInitial * yMax < 0) {
7      return solve(f, initial, yInitial, max,
8       yMax, initial, yInitial);
9    }
10 + verifyBracketing(min,max,f);
11   return solve(f, min, yMin, max, yMax,
12     initial, yInitial);
13 }
```

(b) Correct patch generated by ARJA

Fig. 16. Patches created by a human developer and ARJA for the bug Math73.

1.17 Correct Patch for Math86

The human-written patch and the correct patch generated by ARJA are shown in Fig. 17. Please refer to our paper for the explanation of the correctness.

1.18 Correct Patch for Math98

The correct patch generated by ARJA is shown in Fig. 18. The patch generated by ARJA is exactly the same as the human-written patch and is thus correct.

2 Test-Suite Adequate Bug Repair

Table 1 summarizes the results of the ten repair approaches on 224 bugs considered in Defects4J. For each approach, we list every bug for which at least one test-suite adequate patch is found.

References

[1] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset," *Empirical Software Engineering*, pp. 1–29, 2016.

TABLE 1
Results for 224 bugs considered in Defects4J with ten repair approaches. For each approach, we list the bugs for which the test-suite adequate patches are found

Project	ARJA	ARJA$_v$	ARJA$_m$	ARJA$_b$	GenProg
JFreeChart	C1, C3, C5, C7, C12, C13, C15, C19, C25	C1, C3, C5, C7, C12, C13, C15, C25	C1, C3, C5, C7, C12, C13, C15, C18, C19, C25	C1, C3, C7, C12, C13, C15, C19, C25	C1, C3, C7, C12, C13, C18, C25
	$\sum = 9$	$\sum = 8$	$\sum = 10$	$\sum = 8$	$\sum = 7$
JodaTime	T4, T11, T14, T15	T1, T4, T11	T4, T11, T15, T24	T1, T4, T11, T14, T17	T4, T11, T24
	$\sum = 4$	$\sum = 3$	$\sum = 4$	$\sum = 5$	$\sum = 3$
Commons Lang	L7, L16, L20, L22, L35, L39, L41, L43, L45, L46, L50, L51, L55, L59, L60, L61, L63	L7, L13, L14, L22, L35, L39, L43, L45, L51, L55, L59, L60, L63	L7, L20, L22, L27, L39, L43, L45, L50, L51, L55, L58, L59, L60, L61, L63	L7, L13, L14, L21, L22, L35, L39, L45, L46, L51, L55, L59, L60, L61, L63	L7, L22, L35, L39, L43, L51, L55, L59, L63
	$\sum = 17$	$\sum = 13$	$\sum = 15$	$\sum = 15$	$\sum = 9$
Commons Math	M2, M5, M6, M8, M20, M22, M28, M31, M39, M49, M50, M53, M56, M58, M60, M68, M70, M71, M73, M74, M80, M81, M82, M84, M85, M86, M95, M98, M103	M2, M6, M8, M20, M28, M31, M39, M49, M50, M53, M56, M70, M71, M73, M79, M80, M81, M82, M85, M95	M2, M5, M6, M8, M20, M22, M28, M31, M39, M40, M44, M49, M50, M53, M58, M70, M71, M73, M74, M79, M80, M81, M82, M84, M85, M86, M95, M98, M103	M2, M5, M6, M7, M8, M20, M28, M49, M50, M53, M58, M60, M71, M73, M80, M81, M82, M84, M85, M95, M103	M2, M6, M8, M20, M28, M31, M39, M40, M49, M50, M71, M73, M80, M81, M82, M85, M95
	$\sum = 29$	$\sum = 20$	$\sum = 29$	$\sum = 21$	$\sum = 17$
Total	59 (26.3%)	44 (19.6%)	58 (25.9%)	49 (21.9%)	36 (16.1%)

Project	RSRepair	Kali	jGenProg[1]	jKali[1]	Nopol[1]
JFreeChart	C1, C5, C7, C12, C13, C15, C25	C1, C5, C12, C13, C15, C25, C26	C1, C3, C5, C7, C13, C15, C25	C1, C5, C13, C15, C25, C26	C3, C5, C13, C21, C25, C26
	$\sum = 7$	$\sum = 7$	$\sum = 7$	$\sum = 6$	$\sum = 6$
JodaTime	T4, T11, T24	T4, T11, T24	T4, T11	T4, T11	T11
	$\sum = 3$	$\sum = 3$	$\sum = 2$	$\sum = 2$	$\sum = 1$
Commons Lang	L7, L22, L27, L39, L43, L51, L59, L60, L63	L7, L22, L27, L39, L44, L51, L55, L58, L63			L39, L44, L46, L51, L53, L55, L58
	$\sum = 9$	$\sum = 9$	$\sum = 0$	$\sum = 0$	$\sum = 7$
Commons Math	M2, M5, M6, M8, M18, M20, M28, M31, M39, M49, M50, M53, M58, M68, M70, M71, M73, M74, M80, M81, M82, M84, M85, M95, M103	M2, M8, M20, M28, M31, M32, M49, M50, M80, M81, M82, M84, M85, M95	M2, M5, M8, M28, M40, M49, M50, M53, M70, M71, M73, M78, M80, M81, M82, M84, M85, M95	M2, M8, M28, M32, M40, M49, M50, M78, M80, M81, M82, M84, M85, M95	M32, M33, M40, M42, M49, M50, M57, M58, M69, M71, M73, M78, M80, M81, M82, M85, M87, M88, M97, M104, M105
	$\sum = 25$	$\sum = 14$	$\sum = 18$	$\sum = 14$	$\sum = 21$
Total	44 (19.6%)	33 (14.7%)	27 (12.1%)	22 (9.8%)	35 (15.6%)

[1] The results are organized according to those reported in [1].

```
// CholeskyDecompositionImpl.java
public CholeskyDecompositionImpl(...) {
  for (int i = 0; i < order; ++i) {
    final double[] lI = lTData[i];
-   if (lTData[i][i] <
-    absolutePositivityThreshold) {
-     throw new
-       NotPositiveDefiniteMatrixException();
-   }
    ...
  }
  for (int i = 0; i < order; ++i) {
    final double[] ltI = lTData[i];
+   if (ltI[i] <
+     absolutePositivityThreshold) {
+     throw new
+       NotPositiveDefiniteMatrixException();
+   }
    ...
  }
 ...
}
```

(a) Human-written patch

```
// CholeskyDecompositionImpl.java
public CholeskyDecompositionImpl(...) {
  for (int i = 0; i < order; ++i) {
    final double[] lI = lTData[i];
    if (lTData[i][i] <
      absolutePositivityThreshold) {
      throw new
        NotPositiveDefiniteMatrixException();
    }
    ...
  }
  for (int i = 0; i < order; ++i) {
    final double[] ltI = lTData[i];
+   if (lTData[i][i] <
+     absolutePositivityThreshold) {
+     throw new
+       NotPositiveDefiniteMatrixException();
+   }
    ...
  }
 ...
}
```

(b) Correct patch generated by ARJA

Fig. 17. Patches created by a human developer and ARJA for the bug Math86.

```
// BigMatrixImpl.java
public BigDecimal[] operate(BigDecimal[]
    v) throws IllegalArgumentException {
...
-  final BigDecimal[] out = new
- BigDecimal[v.length];
+  final BigDecimal[] out = new
+ BigDecimal[nRows];
...
}
// RealMatrixImpl.java
public double[] operate(double[] v) throws
    IllegalArgumentException {
...
-  final double[] out = new
-    double[v.length];
+  final double[] out = new double[nRows];
...
}
```

Fig. 18. Correct patch found by ARJA for the bug Math98.