

A dataset of program sosies

1. INTRODUCTION

In this document we present a dataset of sosies that have been generated for large open source Java programs. The objective is to illustrate the diversity of situations in which we can generate good or bad sosies, as defined in our previous work ¹.

This dataset will grow in the future and will serve as the basis to establish a taxonomy of different code regions where we can synthesize valuable program sosies.

Redundant code. There are zones in the computation that are redundant (e.g., a method call performed twice), which can be safely removed and still produce a useful variant.

Extra functionality. Application developers implement functionalities that can handle many different situations, yet, some of these functions might never be used or the situations that the program can handle might never occur when the application is in production ². These are areas that can be safely removed or replaced while still producing useful variants. Listing 7 is an example of sosie that exploits such extra functionality.

Platform specificities Listing 12

Plastic computation / specification Listing 1

Checks Listing 4

2. GOOD SOSIES

Listing 1 is a good example of a sosie in a function which specification is flexible, i.e., it exhibits specification diversity. [elaborate on the expected behavior of hashCode](#)

¹<https://hal.archives-ouvertes.fr/file/index/docid/938855/filename/sosies.pdf>

²<http://people.csail.mit.edu/rinard/paper/oops1a07.comfortZone.pdf>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

Listing 1: hashCode in Rhino and a sosie

```
1 //original
2 public int hashCode(){}
3 if (hashCode == -1) {
4     int h1 = className.hashCode();
5     int h2 = name.hashCode();
6     int h3 = type.hashCode();
7     hashCode = h1 ^ h2 ^ h3;
8 }
9 return hashCode;
10 }
11 //sosie
12 public int hashCode(){}
13 if (hashCode == -1) {
14     int h1 = className.length();
15     int h2 = name.hashCode();
16     int h3 = type.hashCode();
17     hashCode = h1 ^ h2 ^ h3;
18 }
19 return hashCode;
20 }
```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 2 shows a sosie of the `toJson()` method from the Google Gson library. The last statement of the original method is replaced by another one: instead of setting the serialization format of the `writer` it set the indent format. Each variant creates a JJson with slightly different formats, and none of these formatting decisions are part of the specified domain (and actually, specifying the exact formatting of the JJson String could be considered as over-specification).

Here, sosification exploits a specific kind of plasticity that we call “code rigidities”. We have found many regions in programs where statements assign specific values to variables, while any value in a given range would be as good. Fixing one value is what we call a rigidity, and changing this value is an interesting way to create sosies that modify the program state but still deliver a correct service.

Listing 2: toJson in GSON and a sosie

```

2 // Original program
void toJson(Object src, Type typeOfSrc, JsonWriter
  writer){
4   writer.setSerializeNulls(oldSerializeNulls); } }

6 //sosie
void toJson(Object src, Type typeOfSrc, JsonWriter
  writer){
8   writer.setIndent("  ")
} }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

In listing 3, the original program calls `openOutputStream`, which checks different things about the file name, while the sosie directly calls the constructor of `FileOutputStream`. In all nominal cases, these two programs behave exactly in the same way, and the sosie executes less code. In exceptional cases, i.e. when `writeStringToFile()` is called with an invalid file name, the original program handles it, while the variant throws a `FileNotFoundException`. The original program and the sosie exhibit failure diversity on exceptional cases.

Considering the parts of the program that handle checks and exceptional cases as plasticity in the specification and the implementation is conceptually very close to the ideas of exploration of the correctness envelop as expressed by Rinard and colleagues³.

Listing 3: writeStringToFile in commons.io

```

1 //original program
void writeStringToFile(File file, String data,
  Charset encoding, boolean append) throws
  IOException {
3   OutputStream out = null;
  out = openOutputStream(file, append);
5   IOUtils.write(data, out, encoding);
  out.close(); }

7 // sosie
void writeStringToFile(File file, String data,
  Charset encoding, boolean append) throws
  IOException {
9   OutputStream out = null;
  out = new FileOutputStream(file, append); //this
11  statement replaces the original
  IOUtils.write(data, out, encoding);
13  out.close(); }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 4 is an example of sosie that removes checks on inputs.

Listing 4: setAttributes in Rhino and a sosie

```

1 //original
public void setAttributes(String name, int attributes
  ){
3   ScriptableObject.checkValidAttributes(attributes);
  ...
5 }
//sosie
7 public void setAttributes(String name, int attributes
  ){
9   while (attributes < attributes) {
    ++attributes;
    attributes <= 1;
11  }
  ...
13 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 5 is an example of sosie that hits a rigidity: assigning 0 or 4 to `itsMaxStack` has no incidence on the behavior of Rhino.

Listing 5: stopMethod in Rhino and a sosie

```

//original
2 public void stopMethod(short maxLocals) {
  ...
4   itsMaxStack = 0;
  itsStackTop = 0;
6   itsLabelTableTop = 0;
  itsFixupTableTop = 0;
8   itsVarDescriptors = null;
  itsSuperBlockStarts = null;
10  itsSuperBlockStartsTop = 0;
  itsJumpFroms = null;
12 }
//sosie
14 public void stopMethod(short maxLocals) {
  ...
16  itsMaxStack = 4;
  itsStackTop = 0;
18  itsLabelTableTop = 0;
  itsFixupTableTop = 0;
20  itsVarDescriptors = null;
  itsSuperBlockStarts = null;
22  itsSuperBlockStartsTop = 0;
  itsJumpFroms = null;
24 }
  ...

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 6 is a beautiful sosie, which reduces the output space of the program, increasing safety.

³<http://people.csail.mit.edu/rinard/paper/oopsla05.pdf>

Listing 6: `convertArg` in Rhino and a `sosie`

```

1 //original
  public static Object convertArg(Context cx,
    Scriptable scope, Object arg, int typeTag){
3     switch (typeTag) {
        ...
5         case JAVA_OBJECT_TYPE:
            return arg;
7         default:
            throw new IllegalArgumentException();
9     }
}

11 //sosie
  public static Object convertArg(Context cx,
    Scriptable scope, Object arg, int typeTag){
13     switch (typeTag) {
        ...
15         case JAVA_OBJECT_TYPE:
            if (arg != null){return arg;}
17         default:
            throw new IllegalArgumentException();
19     }
}

```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean depth
rep							

Listing 7 is one extreme case found in Google’s GSON library (v. 2.3). The `sosie` completely removes the body of the method, which is supposed to transform the type passed as parameter into an equivalent version that is serializable, and instead it returns the parameter. The `sosie` is covered by 624 different test cases, it is executed 6000 times and all executions complete successfully and all assertions in the test cases are satisfied. This is an example of an advanced feature implemented in the core part of GSON that is not necessary to make the library run correctly.

Listing 7: `canonicalize` in EasyMock and a `sosie`

```

1 //original
2 public static Type canonicalize(Type type) {
3     if (type instanceof Class) {
4         Class<?> c = (Class<?>) type;
5         return c.isArray() ? new GenericArrayTypeImpl(
6             canonicalize(c.getComponentType())) : c;
7     }
8     else
9     if (type instanceof ParameterizedType) {
10        ParameterizedType p = (ParameterizedType) type;
11        return new ParameterizedTypeImpl(p.getOwnerType(
12            ), p.getRawType(), p.getActualTypeArguments(
13            ));
14    }
15    else
16    if (type instanceof GenericArrayType) {
17        GenericArrayType g = (GenericArrayType) type;
18        return new GenericArrayTypeImpl(g.
19            getGenericComponentType());
20    }
21    else
22    if (type instanceof WildcardType) {
23        WildcardType w = (WildcardType) type;
24        return new WildcardTypeImpl(w.getUpperBounds(
25            ), w.getLowerBounds());
26    }
27    else {
28        return type;
29    }
30 }
31 //sosie
32 public static Type canonicalize(Type type) {
33     return type;
34 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
		rep					

Listing 8: `decode` in `commons.codec` and a `sosie`

```

1 // Original program
void decode(final byte[] in, int inPos, final int
    inAvail, final Context context) {
3     switch (context.modulus) {
        case 0 : // impossible, as excluded above
5         case 1 : // 6 bits - ignore entirely
            // not currently tested; perhaps it is
                impossible?
                break;
7     }
9
10    // sosie
11    void decode(final byte[] in, int inPos, final int
        inAvail, final Context context) {
        switch (context.modulus) {
13            case 0 : // impossible, as excluded above
14            case 1 :
15        }

```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean depth
del							

Listing 9: Two variants of `getEntry` in `commons.maths`

```

1 // Original program
2 public double getEntry(final int row, final int
   column) throws OutOfRangeException {
3     MatrixUtils.checkMatrixIndex(this, row, column);
4     return data[row][column];
5 }
6
7 // sosie
8 public double getEntry(final int row, final int
   column) throws OutOfRangeException {
9     return data[row][column];
10 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
del							

Listing 10: Two variants of `setSeed` in `commons.maths`

```

1 // Original program
2 public void setSeed(final int[] seed) {
3     if (seed == null) {
4         setSeed(System.currentTimeMillis() + System.
           identityHashCode(this));
5         return;
6     }
7     System.arraycopy(seed, 0, v, 0, FastMath.min(seed.
           length, v.length));
8     if (seed.length < v.length) {
9         for (int i = seed.length; i < v.length; ++i) {
10             final long l = v[i - seed.length];
11             v[i] = (int) ((18124332531 * (1 ^ (1 >> 30)) + i)
               & 0xffffffffL);
12         }
13     }
14     index = 0;
15     clear();
16 }
17 //sosie
18 public void setSeed(final int[] seed) {
19     if (seed == null) {
20         setSeed(System.currentTimeMillis() + System.
           identityHashCode(this));
21         return;
22     }
23     System.arraycopy(seed, 0, v, 0, FastMath.min(seed.
           length, v.length));
24     if (seed.length < v.length) {
25         for (int i = seed.length; i < v.length; ++i) {
26             final long l = v[i - seed.length];
27             v[i] = (int) ((18124332531 * (1 ^ (1 >> 30)) + i)
               & 0xffffffffL);
28         }
29     }
30     index = 0;
31 }
32 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
del							

Listing 11: Two variants of `invoke` in `EasyMock`

```

1 //original
2 public Object invoke(final Invocation invocation)
   throws Throwable {
3     behavior.checkThreadSafety();
4     if (behavior.isThreadSafe()) {
5         lock.lock();
6         try {
7             return invokeInner(invocation);
8         } finally {
9             lock.unlock();
10        }
11    }
12    return invokeInner(invocation);
13 }
14 //sosie
15 public Object invoke(final Invocation invocation)
   throws Throwable {
16     behavior.checkThreadSafety();
17     if (behavior.isThreadSafe()) {
18         new locks.ReentrantLock();
19     }
20    return invokeInner(invocation);
21 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

In listing 12, the variable `c` is never equal to 'r' (maybe different on windows)

Listing 12: `getCharIgnoreLineEnd` in `Rhino` and a `sosie`

```

1 //original
2 private int getCharIgnoreLineEnd() throws IOException
   {
3     ...
4     if (c <= 127) {
5         if (c == '\n' || c == '\r') {
6             lineEndChar = c;
7             c = '\n';
8         }
9     } else {
10        ...
11    }
12    private int getCharIgnoreLineEnd() throws IOException
   {
13        ...
14        if (c <= 127) {
15            if (c == '\n' || c == '\r') {
16                lineEndChar = c;
17                c = (c) > c ? c : c;
18            }
19        } else {
20            ...
21        }
22    }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

3. USELESS SOSIES

Listing 13: toString in EasyMock and a sosie

```

1 //original
2 public String toString() {
3     if (values.isEmpty()) {
4         return "Nothing captured yet";
5     }
6     if (values.size() == 1) {
7         return String.valueOf(values.get(0));
8     }
9     return values.toString();
10 }
11 //sosie
12 public String toString() {
13     if (values.isEmpty()) {
14         return "Nothing captured yet";
15     }
16     if (values.size() == 1) {
17         if ((values.size()) == 1) {
18             return String.valueOf(values.get(0));
19         }
20     }
21     return values.toString();
22 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 14 is a valid sosie, not beautiful: the first assignment of `index` is useless

Listing 14: ensureIndex in Rhino and a sosie

```

2 //original
3 private int ensureIndex(int key, boolean intType) {
4     int index = -1;
5     //end transformation
6     int firstDeleted = -1;
7     int[] keys = this.keys;
8     if (keys != null) {
9         int fraction = key * A;
10        index = fraction >>> (32 - power);
11        ...
12    }
13    // Inserting of new key
14    if (check && keys != null && keys[index] != EMPTY)
15        Kit.codeBug();
16    if (firstDeleted >= 0) {
17        index = firstDeleted;
18    }
19    else {
20        // Need to consume empty entry: check
21        // occupation level
22        if (keys == null || occupiedCount * 4 >= (1 <<
23            power) * 3) {
24            // Too little unused entries: rehash
25            rehashTable(intType);
26            return insertNewKey(key);
27        }
28        ++occupiedCount;
29    }
30    keys[index] = key;
31    ++keyCount;
32    return index;
33 }
34 //sosie
35 private int ensureIndex(int key, boolean intType) {
36     int index = 65536;
37     //end transformation
38     int firstDeleted = -1;
39     int[] keys = this.keys;
40     if (keys != null) {
41         int fraction = key * A;
42         index = fraction >>> (32 - power);
43         ...
44     }
45    // Inserting of new key
46    if (check && keys != null && keys[index] != EMPTY)
47        Kit.codeBug();
48    if (firstDeleted >= 0) {
49        index = firstDeleted;
50    }
51    else {
52        // Need to consume empty entry: check
53        // occupation level
54        if (keys == null || occupiedCount * 4 >= (1 <<
55            power) * 3) {
56            // Too little unused entries: rehash
57            rehashTable(intType);
58            return insertNewKey(key);
59        }
60        ++occupiedCount;
61    }
62    keys[index] = key;
63    ++keyCount;
64    return index;
65 }

```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

4. TO-BE-DISCUSSED SOSIES

Listing 15: createTypeVariableMap and a sosie in EasyMock

```
// Original program
2 private static Map<TypeVariable<?>, Type>
  createTypeVariableMap(final Class<?> cls) {
    final Map<TypeVariable<?>, Type> typeVariableMap =
      new HashMap<TypeVariable<?>, Type>();
4    extractTypeVariablesFromGenericInterfaces(cls.
      getGenericInterfaces(), typeVariableMap);
    Type genericType = cls.getGenericSuperclass();
    Class<?> type = cls.getSuperclass();
6    while (!Object.class.equals(type)) {
8      if (genericType instanceof ParameterizedType) {
        final ParameterizedType pt = (ParameterizedType)
          genericType;
10       populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
      }
12      extractTypeVariablesFromGenericInterfaces(type.
        getGenericInterfaces(), typeVariableMap);
        genericType = type.getGenericSuperclass();
        type = type.getSuperclass();
14    }
16    type = cls;
    while (type.isMemberClass()) {
18      genericType = type.getGenericSuperclass();
      if (genericType instanceof ParameterizedType) {
20        final ParameterizedType pt = (
          ParameterizedType) genericType;
        populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
22      }
      type = type.getEnclosingClass();
24    }
    return typeVariableMap;
26 }
//sosie
28 private static Map<TypeVariable<?>, Type>
  createTypeVariableMap(final Class<?> cls) {
    final Map<TypeVariable<?>, Type> typeVariableMap =
      new HashMap<TypeVariable<?>, Type>();
30    extractTypeVariablesFromGenericInterfaces(cls.
      getGenericInterfaces(), typeVariableMap);
    Type genericType = cls.getGenericSuperclass();
    Class<?> type = cls.getSuperclass();
32    while (!Object.class.equals(type)) {
34      if (genericType instanceof ParameterizedType) {
        final ParameterizedType pt = (ParameterizedType)
          genericType;
36       populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
      }
38      extractTypeVariablesFromGenericInterfaces(type.
        getGenericInterfaces(), typeVariableMap);
        genericType = type.getGenericSuperclass();
        type = type.getSuperclass();
40    }
42    type = cls;
    return typeVariableMap;
44 }
```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
del							

Listing 16: Two variants of matches in EasyMock

```
//original
2 public boolean matches(final Object actual) {
  if (this.expected == null) {
4    return actual == null;
  }
6  return expected.equals(actual);
}
//sosie
8 public boolean matches(final Object actual) {
10  if (this.expected == null) {
    return true;
12  }
    return expected.equals(actual);
14 }
```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

I think this one is a good one because it just makes the behavior more conservative (no multi threading), but can this prevent some computation?

Listing 17: Two variants of isThreadSafe in EasyMock

```
//original
2 public boolean isThreadSafe() {
  return this.isThreadSafe;
4 }
//sosie
6 public boolean isThreadSafe() {
  return false;
8 }
```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

Listing 18: setLp in Rhino and a sosie

```
//original
2 public void setLp(int lp) {
  this.lp = lp;
4 }
//sosie
6 public void setLp(int lp) {
  this.lp = -1;
8 }
```

#tc	#assert	transfo	node	min	max	median	mean
		type	type	depth	depth	depth	depth
rep							

5. BAD SOSIES