

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. Listing 7 is an example of sokie that exploits redundancy.

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 8 is an example of sokie that exploits such extra functionality.

Caching and optimization. Listing 9

Platform specificities Listing 14

Plastic computation / specification Listing 1

Checks Listing 4

2. GOOD SOSIES

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

²<http://people.csail.mit.edu/rinard/paper/oopsla07.comfortZone.pdf>

Listing 1 is a good example of a sokie in a function which specification is flexible, i.e., it exhibits specification diversity. **elaborate on the expected behavior of hashCode**

Listing 1: hashCode in Rhino and a sokie

```
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 //sokie
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 sokie 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 JSON with slightly different formats, and none of these formatting decisions are part of the specified domain (and actually, specifying the exact formatting of the JSON String could be considered as over-specification).

Here, sosiefication 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) {
4     ...
5     case JAVA_OBJECT_TYPE:
      return arg;
7     default:
      throw new IllegalArgumentException();
9   }
10 }
11 //sosie
  public static Object convertArg(Context cx,
    Scriptable scope, Object arg, int typeTag){
13   switch (typeTag) {
14     ...
15     case JAVA_OBJECT_TYPE:
      if (arg != null){return arg;}
17     default:
      throw new IllegalArgumentException();
19   }
20 }

```

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

Listing 7 is an example of `sosie` that exploits redundancy in the code. The statement `if (isEmpty(padStr)) padStr = SPACE;` at line 4 assigns a value to the `padStr` variable, then this variable is passed when calling methods `leftPad` and `rightPad`. Yet, each of these two methods include the exact same statement, which will eventually assign a value to `padStr`. So, the statement is redundant and can be removed from the `sosie`.

Listing 7: `center` in `commons.lang` and a `sosie`

```

1 //original
  public static String center(String str, final int
    size, String padStr) {
3   if (str == null || size <= 0) {return str;}
4   if (isEmpty(padStr)) {
      padStr = SPACE;
5   }
7   final int strLen = str.length();
8   final int pads = size - strLen;
9   if (pads <= 0) {return str;}
10  str = leftPad(str, strLen + pads / 2, padStr);
11  str = rightPad(str, size, padStr);
12  return str;}
13 //sosie
  public static String center(String str, final int
    size, String padStr) {
15   if (str == null || size <= 0) {return str;}
16   if-stmt deleted
17   final int strLen = str.length();
18   final int pads = size - strLen;
19   if (pads <= 0) {return str;}
20   str = leftPad(str, strLen + pads / 2, padStr);
21   str = rightPad(str, size, padStr);
22   return str;}

```

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

Listing 8 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 8: `canonicalize` in `EasyMock` and a `sosie`

```

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

```

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

Listing 9 shows a `sosie` for the `toString()` method in the `Range` class of `commons.lang`. This method builds a `String` value and saves it in the `toString` attribute for future usage (a sort of cache to save computation in the future). The `sosie` removes this cache operation thus reducing a bit the performance while maintaining the same service.

Listing 9: toString in commons.lang and a sosie

```

1 //original
2 public String toString() {
3     String result = toString();
4     if (result == null) {
5         final StringBuilder buf = new StringBuilder(32);
6         buf.append('[');
7         buf.append(minimum);
8         buf.append("..");
9         buf.append(maximum);
10        buf.append(']');
11        result = buf.toString();
12        toString = result;
13    }
14    return result;
15 //sosie
16 public String toString() {
17     String result = toString();
18     if (result == null) {
19         final StringBuilder buf = new StringBuilder(32);
20         buf.append('[');
21         buf.append(minimum);
22         buf.append("..");
23         buf.append(maximum);
24         buf.append(']');
25         result = buf.toString();
26         assignment deleted
27     }
28     return result;

```

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

Listing 10: decode in commons.codec and a sosie

```

// Original program
2 void decode(final byte[] in, int inPos, final int
   inAvail, final Context context) {
   switch (context.modulus) {
4       case 0 : // impossible, as excluded above
5       case 1 : // 6 bits - ignore entirely
6           // not currently tested; perhaps it is
              impossible?
              break;
8   }

10 // sosie
11 void decode(final byte[] in, int inPos, final int
   inAvail, final Context context) {
12     switch (context.modulus) {
13         case 0 : // impossible, as excluded above
14         case 1 :

```

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

Listing 11: 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 // sosie
7 public double getEntry(final int row, final int
   column) throws OutOfRangeException {
8     return data[row][column];
9 }

```

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

Listing 12: Two variants of setSeed in commons.maths

```

// 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 }

```

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

Listing 13: Two variants of invoke in EasyMock

```

//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 14, the variable c is never equal to 'r' (maybe different on windows)

Listing 14: `getCharIgnoreLineEnd` in Rhino and a sosie

```

1 //original
private int getCharIgnoreLineEnd() throws IOException
{
    ...
    if (c <= 127) {
        if (c == '\n' || c == '\r') {
            lineEndChar = c;
            c = '\n';
        }
    } else {
        ...
    }
}
11 private int getCharIgnoreLineEnd() throws IOException
{
    ...
    if (c <= 127) {
        if (c == '\n' || c == '\r') {
            lineEndChar = c;
            c = (c) > c ? c : c;
        }
    } else {
        ...
    }
}
21 }

```

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

3. USELESS SOSIES

Listing 15: `toString` in EasyMock and a sosie

```

1 //original
public String toString() {
    if (values.isEmpty()) {
        return "Nothing captured yet";
    }
    if (values.size() == 1) {
        return String.valueOf(values.get(0));
    }
    return values.toString();
}
11 //sosie
public String toString() {
    if (values.isEmpty()) {
        return "Nothing captured yet";
    }
    if (values.size() == 1) {
        if ((values.size()) == 1) {
            return String.valueOf(values.get(0));
        }
    }
    return values.toString();
}
21 }

```

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

Listing 16 is a valid sosie, not beautiful: the first assignment of `index` is useless

Listing 16: `ensureIndex` in Rhino and a sosie

```

2 //original
private int ensureIndex(int key, boolean intType) {
    int index = -1;
    //end transformation
    int firstDeleted = -1;
    int[] keys = this.keys;
    if (keys != null) {
        int fraction = key * A;
        index = fraction >>> (32 - power);
        ...
    }
    // Inserting of new key
    if (check && keys != null && keys[index] != EMPTY)
        Kit.codeBug();
    if (firstDeleted >= 0) {
        index = firstDeleted;
    }
    else {
        // Need to consume empty entry: check
        // occupation level
        if (keys == null || occupiedCount * 4 >= (1 <<
            power) * 3) {
            // Too little unused entries: rehash
            rehashTable(intType);
            return insertNewKey(key);
        }
        ++occupiedCount;
    }
    keys[index] = key;
    ++keyCount;
    return index;
}
30 }
//sosie
32 private int ensureIndex(int key, boolean intType) {
    int index = 65536;
    //end transformation
    int firstDeleted = -1;
    int[] keys = this.keys;
    if (keys != null) {
        int fraction = key * A;
        index = fraction >>> (32 - power);
        ...
    }
    // Inserting of new key
    if (check && keys != null && keys[index] != EMPTY)
        Kit.codeBug();
    if (firstDeleted >= 0) {
        index = firstDeleted;
    }
    else {
        // Need to consume empty entry: check
        // occupation level
        if (keys == null || occupiedCount * 4 >= (1 <<
            power) * 3) {
            // Too little unused entries: rehash
            rehashTable(intType);
            return insertNewKey(key);
        }
        ++occupiedCount;
    }
    keys[index] = key;
    ++keyCount;
    return index;
}
60 }

```

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

4. TO-BE-DISCUSSED SOSIES

Listing 17: 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    }
    type = cls;
    return typeVariableMap;
42 }
44 }
```

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

Listing 18: 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) {
  if (this.expected == null) {
10    return true;
  }
12  return expected.equals(actual);
14 }
```

#tc	#assert	transfo	node	min	max	median	mean
type	type	depth	depth	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 19: 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	depth	depth
rep							

Listing 20: 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	depth	depth
rep							

5. BAD SOSIES