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.

2. GOOD SOSIES

In listing 1, 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 colleagues [2].

[1] `https://hal.archives-ouvertes.fr/file/index/docid/938855/filename/sosies.pdf`
[2] http://people.csail.mit.edu/rinard/paper/oopsla05.pdf

Listing 1: `writeStringToFile` in commons.io

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

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

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

```java

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

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

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

Listing 3: `decode` in commons.codec and a sosie

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

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

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

Listing 4: Two variants of `getEntry` in commons.maths

```java
// Original program
public double getEntry(final int row, final int
     column) throws OutOfRangeException {
  MatrixUtils.checkMatrixIndex(this, row, column);
  return data[row][column];
}

// sosie
public double getEntry(final int row, final int
     column) throws OutOfRangeException {
  return data[row][column];
}
```

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

Listing 5: Two variants of `setSeed` in commons.maths

```java
// Original program
public void setSeed(final int[] seed) {
  if (seed == null) {
    setSeed(System.currentTimeMillis() + System.
         identityHashCode(this));
    return;
  }
  System.arraycopy(seed, 0, v, 0, FastMath.min(seed.
       length, v.length));
  if (seed.length < v.length) {
   for (int i = seed.length; i < v.length; ++i) {
    final long l = v[i - seed.length];
    v[i] = (int) ((1812433253l * (l ^ (l >> 30)) + i)
         & 0xffffffffL);
    }
  }
  index = 0;
  clear();
}
//sosie
public void setSeed(final int[] seed) {
  if (seed == null) {
    setSeed(System.currentTimeMillis() + System.
         identityHashCode(this));
    return;
  }
  System.arraycopy(seed, 0, v, 0, FastMath.min(seed.
       length, v.length));
  if (seed.length < v.length) {
   for (int i = seed.length; i < v.length; ++i) {
    final long l = v[i - seed.length];
    v[i] = (int) ((1812433253l * (l ^ (l >> 30)) + i)
         & 0xffffffffL);
    }
  }
  index = 0;

}
```

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

Listing 6: `createTypeVariableMap and a sosie` in EasyMock

```java
// Original program
private static Map<TypeVariable<?>, Type>
    createTypeVariableMap(final Class<?> cls) {
  final Map<TypeVariable<?>, Type> typeVariableMap =
      new HashMap<TypeVariable<?>, Type>();
  extractTypeVariablesFromGenericInterfaces(cls.
      getGenericInterfaces(), typeVariableMap);
  Type genericType = cls.getGenericSuperclass();
  Class<?> type = cls.getSuperclass();
  while (!Object.class.equals(type)) {
    if (genericType instanceof ParameterizedType) {
      final ParameterizedType pt = (ParameterizedType
          ) genericType;
      populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
    }
    extractTypeVariablesFromGenericInterfaces(type.
        getGenericInterfaces(), typeVariableMap);
    genericType = type.getGenericSuperclass();
    type = type.getSuperclass();
  }
  type = cls;
  while (type.isMemberClass()) {
    genericType = type.getGenericSuperclass();
    if (genericType instanceof ParameterizedType) {
      final ParameterizedType pt = (
          ParameterizedType) genericType;
      populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
    }
    type = type.getEnclosingClass();
  }
  return typeVariableMap;
}
//sosie
private static Map<TypeVariable<?>, Type>
    createTypeVariableMap(final Class<?> cls) {
  final Map<TypeVariable<?>, Type> typeVariableMap =
      new HashMap<TypeVariable<?>, Type>();
  extractTypeVariablesFromGenericInterfaces(cls.
      getGenericInterfaces(), typeVariableMap);
  Type genericType = cls.getGenericSuperclass();
  Class<?> type = cls.getSuperclass();
  while (!Object.class.equals(type)) {
    if (genericType instanceof ParameterizedType) {
      final ParameterizedType pt = (ParameterizedType
          ) genericType;
      populateTypeMapFromParameterizedType(pt,
          typeVariableMap);
    }
    extractTypeVariablesFromGenericInterfaces(type.
        getGenericInterfaces(), typeVariableMap);
    genericType = type.getGenericSuperclass();
    type = type.getSuperclass();
  }
  type = cls;
  return typeVariableMap;
}
```

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

Listing 7: `toString` in EasyMock and a sosie

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

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

Listing 8: Two variants of `invoke` in EasyMock

```java
//original
public Object invoke(final Invocation invocation)
    throws Throwable {
  behavior.checkThreadSafety();
  if (behavior.isThreadSafe()) {
    lock.lock();
    try {
      return invokeInner(invocation);
    } finally {
      lock.unlock();
    }
  }
  return invokeInner(invocation);
}
//sosie
public Object invoke(final Invocation invocation)
    throws Throwable {
  behavior.checkThreadSafety();
  if (behavior.isThreadSafe()) {
    new locks.ReentrantLock();
  }
  return invokeInner(invocation);
}
```

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

3. TO-BE-DISCUSSED SOSIES

Listing 9: Two variants of `matches` in EasyMock

```
//original
public boolean matches(final Object actual) {
  if (this.expected == null) {
    return actual == null;
  }
  return expected.equals(actual);
}
//sosie
public boolean matches(final Object actual) {
  if (this.expected == null) {
    return true;
  }
  return expected.equals(actual);
}
```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean 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 10: Two variants of `isThreadSafe` in EasyMock

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

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

4. BAD SOSIES

5. SOSIES IN RHINO

In listing 11, the variable `c` is never equal to ‘r’ (maybe different on windows)

Listing 11: `getCharIgnoreLineEnd` in Rhino and a sosie

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

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

Listing 12 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

Listing 12: `hashCode` in Rhino and a sosie

```
//original
public int hashCode(){}
  if (hashCode == -1) {
  int h1 = className.hashCode();
  int h2 = name.hashCode();
  int h3 = type.hashCode();
  hashCode = h1 ^ h2 ^ h3;
  }
  return hashCode;
}
//sosie
public int hashCode(){}
  if (hashCode == -1) {
  int h1 = className.length();
  int h2 = name.hashCode();
  int h3 = type.hashCode();
  hashCode = h1 ^ h2 ^ h3;
  }
  return hashCode;
}
```

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

Listing 13: `setLp` in Rhino and a sosie

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

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

```
//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 litle unused entries: rehash
        rehashTable(intType);
        return insertNewKey(key);
      }
    ++occupiedCount;
  }
  keys[index] = key;
  ++keyCount;
  return index;
}
//sosie
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 litle unused entries: rehash
        rehashTable(intType);
        return insertNewKey(key);
      }
    ++occupiedCount;
  }
  keys[index] = key;
  ++keyCount;
  return index;
}
```

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

Listing 15 is an example of sosie that removes checks on inputs (very much ala Rinard in failure oblivious or correctness envelop).

Listing 15: **setAttributes** in Rhino and a sosie

```
//original
public void setAttributes(String name, int attributes
      ){
  ScriptableObject.checkValidAttributes(attributes);
   ...
}
//sosie
public void setAttributes(String name, int attributes
      ){
  while (attributes < attributes) {
    ++attributes;
    attributes <<= 1;
  }
   ...
}
```

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

Listing 16 is an example of sosie that hits a rigidity: assigning 0 or 4 to **itsMaxStack** has no incidence on the behavior of Rhino.

Listing 16: **stopMethod** in Rhino and a sosie

```
//original
public void stopMethod(short maxLocals) {
   ...
  itsMaxStack = 0;
  itsStackTop = 0;
  itsLabelTableTop = 0;
  itsFixupTableTop = 0;
  itsVarDescriptors = null;
  itsSuperBlockStarts = null;
  itsSuperBlockStartsTop = 0;
  itsJumpFroms = null;
}
//sosie
public void stopMethod(short maxLocals) {
   ...
  itsMaxStack = 4;
  itsStackTop = 0;
  itsLabelTableTop = 0;
  itsFixupTableTop = 0;
  itsVarDescriptors = null;
  itsSuperBlockStarts = null;
  itsSuperBlockStartsTop = 0;
  itsJumpFroms = null;
}
 ...
```

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

Listing 17 is a beautiful sosie, which reduces the output space of the program, increasing safety.

Listing 17: `convertArg` in Rhino and a sosie

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

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