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 sosie 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 sosie that exploits such extra functionality.

Caching and optimization. Listing 9

Optimizations and shortcuts. Listing 11

Platform specificities Listing 16

Plastic computation / specification Listing 10

Checks Listing 4

[1] `https://hal.archives-ouvertes.fr/file/index/docid/938855/filename/sosies.pdf`
[2] `http://people.csail.mit.edu/rinard/paper/oopsla07.comfortZone.pdf`

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

Listing 1: `hashCode` in Rhino and a sosie

```
1  //original
   public int hashCode(){}
3    if (hashCode == -1) {
     int h1 = className.hashCode();
5    int h2 = name.hashCode();
     int h3 = type.hashCode();
7    hashCode = h1 ^ h2 ^ h3;
     }
9    return hashCode;
   }
11 //sosie
   public int hashCode(){}
13   if (hashCode == -1) {
     int h1 = className.length();
15   int h2 = name.hashCode();
     int h3 = type.hashCode();
17   hashCode = h1 ^ h2 ^ h3;
     }
19   return hashCode;
   }
```

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

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

```
// Original program
void toJson(Object src, Type typeOfSrc, JsonWriter
    writer){
  *{\color{grey}...}*)
  finally {
    writer.setLenient(oldLenient);
    writer.setHtmlSafe(oldHtmlSafe);
    writer.setSerializeNulls(oldSerializeNulls); }}
//sosie
void toJson(Object src, Type typeOfSrc, JsonWriter
    writer){
  finally {
    writer.setLenient(oldLenient);
    writer.setHtmlSafe(oldHtmlSafe);
    statement replaced by following
    writer.setIndent("  ");}}
```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean 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 colleagues [3].

Listing 3: `writeStringToFile` in commons.io

```
//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;
  statement replaced by following
  out = new FileOutputStream(file, append);
  IOUtils.write(data, out, encoding);
  out.close(); }
```

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

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

Listing 4: `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 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
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 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

```
//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					

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

```
//original
public static String center(String str, final int
     size, String padStr) {
  if (str == null || size <= 0) {return str;}
  if (isEmpty(padStr)) {
    padStr = SPACE;
  }
  final int strLen = str.length();
  final int pads = size - strLen;
  if (pads <= 0) {return str;}
  str = leftPad(str, strLen + pads / 2, padStr);
  str = rightPad(str, size, padStr);
  return str;}
//sosie
public static String center(String str, final int
     size, String padStr) {
  if (str == null || size <= 0) {return str;}
  if-stmt deleted
  final int strLen = str.length();
  final int pads = size - strLen;
  if (pads <= 0) {return str;}
  str = leftPad(str, strLen + pads / 2, padStr);
  str = rightPad(str, size, padStr);
  return str;}
```

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

```
//original
public static Type canonicalize(Type type) {
  if (type instanceof Class) {
    Class<?> c = (Class<?>) type;
    return c.isArray() ? new GenericArrayTypeImpl(
        canonicalize(c.getComponentType())) : c;
  }
  else
    if (type instanceof ParameterizedType) {
      ParameterizedType p = (ParameterizedType) type;
      return new ParameterizedTypeImpl(p.getOwnerType
          (),p.getRawType(), p.getActualTypeArguments
          ());
    }
    else
      if (type instanceof GenericArrayType) {
        GenericArrayType g = (GenericArrayType) type;
        return new GenericArrayTypeImpl(g.
            getGenericComponentType());
      }
      else
      if (type instanceof WildcardType) {
        WildcardType w = (WildcardType) type;
        return new WildcardTypeImpl(w.getUpperBounds
            (), w.getLowerBounds());
      }
      else {
        return type;
      }
}
//sosie
public static Type canonicalize(Type type) {
  return type;
}
```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean depth
623	1041	rep	if	1	3817	1862	1863.216

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

```
//original
public String toString() {
  String result = toString;
  if (result == null) {
    final StringBuilder buf = new StringBuilder(32);
    buf.append('[');
    buf.append(minimum);
    buf.append("..");
    buf.append(maximum);
    buf.append(']');
    result = buf.toString();
    toString = result;
  }
  return result;}
//sosie
public String toString() {
  String result = toString;
  if (result == null) {
    final StringBuilder buf = new StringBuilder(32);
    buf.append('[');
    buf.append(minimum);
    buf.append("..");
    buf.append(maximum);
    buf.append(']');
    result = buf.toString();
    assignment deleted
  }
  return result;}
```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean depth
2		del	stmt list	1	2	2	1.5

Listing 10 is an example where sosiefication exploits plasticity in the computation, which can be found in many programs. The `hashCode()` method must return an integer value that can be used to quickly retrieve a value in a collection. Yet, the exact value of this integer is not part of the specification, i.e., there are many ways to compute this value. Thus, removing a statement in this method does not change the validity of the service provided by the function.

Listing 10: `hashCode` in commons.lang and a sosie

```
//original
public int hashCode() {
  int result = hashCode;
  if (hashCode == 0) {
    result = 17;
    result = 37 * result + getClass().hashCode();
    result = 37 * result + minimum.hashCode();
    result = 37 * result + maximum.hashCode();
    hashCode = result;
  }
  return result;}
//sosie
public int hashCode() {
  int result = hashCode;
  if (hashCode == 0) {
    result = 17;
    result = 37 * result + getClass().hashCode();
    assignment deleted
    result = 37 * result + maximum.hashCode();
    hashCode = result;
  }
  return result;}
```

#tc	#assert	transfo type	node type	min depth	max depth	median depth	mean depth
1		del	stmt list	1	1	1	1

Listing 11 displays an excerpt of the `nextLong()` method in GSon. Under certain conditions, the method returns a value, skipping all the computation performed after the `try` block. Yet, this is a shortcut in the computation, i.e., if we remove the `return` statement, the subsequent code will return exactly the same value, with some additional computation.

Listing 11: `nextLong` in GSon and a sosie

```
//original
public long nextLong() throws IOException {
  *{\color{grey}...}*)
  if (p == PEEKED_NUMBER) {
    peekedString = new String(buffer, pos,
        peekedNumberLength);
    pos += peekedNumberLength;
  } else if (p == PEEKED_SINGLE_QUOTED || p ==
      PEEKED_DOUBLE_QUOTED) {
    peekedString = nextQuotedValue(p ==
        PEEKED_SINGLE_QUOTED ? '\'' : '"');
    try {
      long result = Long.parseLong(peekedString);
      peeked = PEEKED_NONE;
      pathIndices[stackSize - 1]++;
      return result;
      }
  *{\color{grey}...}*)
  return result;}
//sosie
public long nextLong() throws IOException {
  *{\color{grey}...}*)
  if (p == PEEKED_NUMBER) {
    peekedString = new String(buffer, pos,
        peekedNumberLength);
    pos += peekedNumberLength;
  } else if (p == PEEKED_SINGLE_QUOTED || p ==
      PEEKED_DOUBLE_QUOTED) {
    peekedString = nextQuotedValue(p ==
        PEEKED_SINGLE_QUOTED ? '\'' : '"');
    try {
      long result = Long.parseLong(peekedString);
      peeked = PEEKED_NONE;
      pathIndices[stackSize - 1]++;
      assignment deleted
      }
  *{\color{grey}...}*)
  return result;}
```

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

Listing 12: `decode` in commons.codec and a sosie

```
// 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 13: Two variants of `getEntry` in commons.maths

```
// 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 14: Two variants of `setSeed` in commons.maths

```
// 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) ((18124332531 * (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) ((18124332531 * (l ^ (l >> 30)) + i)
        & 0xffffffffL);
    }
  }
  index = 0;

}
```

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

Listing 15: Two variants of `invoke` in EasyMock

```
//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					

In listing 16, the variable `c` is never equal to 'r' (maybe different on windows)

Listing 16: `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					

3. FOOLING SOSIES

Listing 17: `toString` in EasyMock and a sosie

```
//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 18 is a valid sosie, not beautiful: the first assignment of `index` is useless

Listing 18: `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					

4. TO-BE-DISCUSSED SOSIES

Listing 19: `createTypeVariableMap and a sosie` in EasyMock

```
// 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 20: 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 21: 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					

Listing 22: `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					

5. BAD SOSIES