SAGA User Manual

Stijn de Gouw

1 Introduction

SAGA is a run-time verifier for Java programs. During execution of a Java program, SAGA intercepts method calls and method returns ("events"). Which events are monitored is determined by a communication view, see Section 2. SAGA then checks whether the trace of calls and returns conforms to a specification, given by the user.

Attribute grammars are specifications, see Section 3. The underlying context-free grammar determines the order in which events should occur, in other words: the context-free grammar specifies the desired control-flow of a Java program. Attributes define properties of the data-flow of the program, and assertions over these attributes specify the desired values of the attributes.

2 Communication Views

A communication view is a partial mapping which assigns a terminal (name) to an event. The terminals are used in the grammar (the grammar productions determine the valid orderings between terminals). Partiality allows filtering out irrelevant events. The EBNF syntax of communication views is as follows:

⟨*View*⟩ ::= LocalView: `local view` ⟨*Identifier*⟩ `grammar` ⟨*Filename*⟩ `specifies` ⟨*ClassOrInterfaceType*⟩ `{` (⟨*LocalTokenDef*⟩ `,`)+ `}`
| GlobalView: `global view` ⟨*Identifier*⟩ `grammar` ⟨*Filename*⟩ `{` (⟨*GlobalTokenDef*⟩ `,`)+ `}` ;

⟨*Filename*⟩ ::= `/`? (⟨*Identifier*⟩ `/`)+ (`.` ⟨*Identifier*⟩)? ;

⟨*LocalTokenDef*⟩ ::= ⟨*InEvent*⟩ ⟨*Identifier*⟩
| ⟨*OutEvent*⟩ ⟨*Identifier*⟩

⟨*GlobalTokenDef*⟩ ::= ⟨*OutEvent*⟩ ⟨*Identifier*⟩

⟨*InEvent*⟩ ::= InCall: (`call` | `return`) `ExcludeSelfCalls`? ⟨*MethodHeader*⟩
| InCons: ⟨*ConsHeader*⟩

⟨*OutEvent*⟩ ::= OutCall: (`call` | `return`) `ExcludeSelfCalls`? ⟨*GlobalMethodHeader*⟩
| OutCons: (`call` | `return`) ⟨*GlobalConsHeader*⟩

⟨*GlobalMethodHeader*⟩ ::= ⟨*Modifier*⟩* ⟨*MethodRes*⟩ ⟨*ClassOrInterfaceType*⟩ `.` ⟨*MethodDeclarator*⟩ ⟨*Throws*⟩?

⟨*ConsHeader*⟩ ::= ⟨*Modifier*⟩* `new (` ⟨*FormalParameters*⟩ `)` ⟨*Throws*⟩?

⟨*GlobalConsHeader*⟩ ::= ⟨*Modifier*⟩* ⟨*ClassOrInterfaceType*⟩ `. new (` ⟨*FormalParameters*⟩ `)` ⟨*Throws*⟩?

The syntax of the unspecified non-terminals (like $Identifier$, $FormalParameters$, $Modifier$, $MethodHeader$ etc) is that of Java.

There are two types of views: local and global views. A local view selects the relevant events that occur in the history of a *single object* (i.e. the history "local" to that object). The type of that object is given after the "specifies" keyword. A global view selects events that occur anywhere in the Java program during execution (i.e. events from a "global" history). For an example of a local communication view, see src/main/specifications/nl/cwi/saga/breader/spec/BReader.view in the BufferedReader project. The PingPong project contains an example global communication view.

A global view contains $OutEvent$s: in addition to the method signature to be monitored, the class or interface type containing the method must also be named explicitly (since there can be multiple classes containing the same signature method). A local view contains also $InEvent$s. An $InEvent$ refers to a call or return of a method implemented by the object under consideration. Since this is always the same type (namely that given after the "specifies" keyword), only the method signature should be given.

3 Attribute Grammars

SAGA currently uses ANTLR v3 [**?**] to generate a Java parser for the attribute grammar. This means two things: grammars must be given in ANTLR 3 syntax, and all features (and limitations) supported by ANTLR can be used inside the grammars. See the BufferedReader and PingPong example grammars for a general template.

In addition to the fact that grammars must conform to the ANTLR syntax, the following rules apply:

- The start non-terminal of the grammar must be named "start".
- All user-defined types (classes or interfaces) that appear in the communication view must be imported in an @header section in the ANTLR grammar. This includes parameter types, return types, and types named in outgoing events (see Section 2 for details).
- The methods recoverFromMismatchedToken and recoverFromMismatchedSet must be overridden in a @parser::members section, and RecognitionExceptions must be caught and thrown in an @rulecatch section (see the example BReader.g grammar inside the BufferedReader project).

- Attributes of grammar terminals (i.e. built-in attributes referring to the objects involved in a method call or return) can be accessed by the getters: caller (), callee (), result () and name(), where name is the parameter of the parameter as used in the view.
- To access attributes of a terminal inside the grammar, the terminal must be dereferenced to the correct type. If the view is called BReaderHistory, and the view contains an event called CLOSE referring to a call to the close method, then the type is:((BReaderHistoryAspect.call_close)$CLOSE. If the the CLOSE event instead referred to returns of close, then the type is: ((BReaderHistoryAspect.call_close)$CLOSE).

4 Usage

To compile the BufferedReader and PingPong project, we use Maven. All dependencies used inside the Maven project are available from Maven's central repository, except the Rascal artifact. Prerequisites:

- Rascal 0.6.2.201405101911 (Command-Line version). See `http://www.rascal-mpl.org/` for the official homepage and `https://github.com/cwi-swat/saga/lib` for the Rascal JAR[1].

To install the Rascal JAR to the local repository, execute the following command in the directory containing the Rascal JAR (named rascal−0.6.2.201405101911.jar):

```
mvn org.apache.maven.plugins:maven-install-plugin:2.5.1:install-file
\
-Dfile=rascal-0.6.2.201405101911.jar \
-DgroupId=org.rascalmpl \
-DartifactId=rascal-shell \
-Dversion=0.6.2.201405101911 \
-Dpackaging=jar
```

Both the BufferedReader and the PingPong project can then be compiled and packaged by executing (inside the folder with pom.xml):

```
mvn clean package
```

The BufferedReader, which takes a file "FILEPATH" as a parameter, can then be executed by:

```
java -cp target/lib/antlr-runtime-3.4.jar: \
target/lib/aspectjrt-1.7.3.jar: \
target/breader-1.0-SNAPSHOT.jar \
-ea nl.cwi.saga.breader.BReader FILEPATH
```

[1]The official homepage offers only the latest version, which is not necessarily backwards compatible

The -ea parameter, which enables assertion checking, must be used to turn on SAGA. Executing the above command should lead to an error reported by SAGA: the grammar specifies that the BufferedReader should be closed by the same object that created it, but the client program violates this property. SAGA will print a counterexample in the form of a sequence diagram. The sequence diagram can be visualized the Quick Sequence Diagram Editor (`http://sdedit.sourceforge.net/`).

To create your own project, take the BufferedReader project as a template and adjust the values of the properties specification .src and rascal.target in pom.xml. When executing your program, make sure to add the ANTLR and AspectJ JARs to your classpath.