KBehavior Documentation

Contents

Chapter 1

KBehavior User Manual

This document describes how to use the standalone implementation of the KBehavior algorithm detailed in [3].

The command to run KBehavior is:

```
java -jar kbehavior-XXXXXXXX.jar <traceFile> <confFile> <outputFile> <showFsa> [<fsaToExtend>]
```

where:

<traceFile>	is the trace file
<confFile>	is the configuration file
<outputFile>	is the file where the output FSA will be saved
<showFsa>	it can be either true or false, it specifies if you want to graphically display the inferred FSA
<fsaToExtend>	indicates the name of a file name with a FSA that needs to be extended with the input trace file. If you want to generate a new FSA omit this parameter.

If you need to graphically visualize a saved FSA, the following command can be executed:

```
java -cp kbehavior-XXXXXX.jar tools.ShowFSA <path/to/fsa>
```

Note that the FSA is visualized by JFLAP (`www.jflap.org`).

If you need to textually visualize a saved FSA, the following command can be executed:

```
java -cp kbehavior140907.jar tools.FSAInspector -print myFSA.fsa
```

The FSAInspector can provide further information about a saved FSA. The command `java -cp kbehavior-XXXXXXXX.jar tools.FSAInspector` visualizes the list of available options.

1.1 Configuration

This section overviews the configuration file. Table 1.1 describes parameters while Figure 1.1 shows an example configuration file.

Parameter name	**Description**	**Example**	**Mandatory**
logger	It indicates where the logging should print information. Can be one of: console, file.	console	Y
level	Logger verbosity level: • 0 : log only critical events • 1 : log unexpected events • 2 : log events (FSA Extension etc) • 3 : log info (addBranch, addTail, merge) • 4 : log debugging info (trace all events)	1	Y
logfile	The name of the file where you the logging info are saved if you set *file* as a logger.	event.log	N
enableMinimization	Type of minimization used by kBehavior: • none : no minimization • end : minimize after all the traces have been processed • step : minimize after a trace has been processed	end	Y
minTrustLen	Minimum length of the behaviors considered during the matching process.	2	Y
cutoffSearch	enables an optimization in the inference process.	true	Y
maxTrustLen	If *cutoffSearch* is set to true, kBehavior matches two behaviors of length greater or equal maxTrustLen, independently from the existence of unknown longer behaviors.	6	Y
stepSave	Set to true if you want to save all partial FSAs obtained after processing of every trace.	false	N

Table 1.1: Parameters of KBehavior configuration file

```
level = 3
logger = console
logfile = event.log

minTrustLen = 2
maxTrustLen = 4

enableMinimization = step

cutOffSearch = true
stepsave = false
```

Figure 1.1: An example configuration file

1.2 Example

A simple example is provided in the file *kebehavior-example.zip*.

1.3 Additional Inference Engines

The kBehavior jar comes with three additional inference engines kTailEngine, cookEngine, and reissEngine that respectively implement the KTail inference algorithm [1] plus the versions of the algorithm with the optimizations introduced by Cook et al. [2] and Reiss et al. [4].

These three additional inference engines are implemented in the package *grammarInference.Engine*.

You can invoke the three engines by specifying the name of the algorithm to execute:

- *grammarInference.Engine.kTailEngine*
- *grammarInference.Engine.cookEngine*
- *grammarInference.Engine.reissEngine*

Example (pay attention to replace *-jar* with *-cp*):

```
java -cp kbehavior-XXXXXXXX.jar grammarInference.Engine.kTailEngine
<traceFile> <confFile> <outputFile> <showFsa>
```

The only difference between these three additional engines and KBhevaior is that they do not allow to provide an <fsaToExtend> argument.

Furthermore the property *minimizationOption* of the configuration file accepts only the values "none" or "end" (this depends upon the fact that these engines are not incremental and thus we cannot use the option "step" to incrementally minimize the FSA).

Options *maxTrustLen*, *stepSave*, and *cutoffSearch* are ignored.

Option *minTrustLen* is used to specify the parameter k of the three algorithms, alternatively you can specify an option whose name is k.

Bibliography

[1] A. W. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, pages 592–597, 1972.

[2] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[3] L. Mariani, F. Pastore, and M. Pezze. Dynamic analysis for diagnosing integration faults. *IEEE Transactions on Software Engineering*, 37(4):486–508, 2011.

[4] M. Renieres and S. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'03).*, pages 30–39, Oct. 2003.