The COMICS Tool
Version 1.0

Computing Minimal Counterexamples for Discrete-Time Markov Chains

Manual

Erika Ábrahám, Nils Jansen, Ralf Wimmer
# Contents

1 Introduction 3

2 Input Format 5
   2.1 The .dtmc-Format ................................. 5
   2.2 The .tra-Format .................................. 6
   2.3 The .lab-Format ................................. 6
   2.4 The .xml-Format .................................. 7
   2.5 The .conf-Format .................................. 7

3 Usage 11
   3.1 Command-line Mode ................................. 11
      3.1.1 Basic Options ................................. 11
      3.1.2 Heuristics .................................. 13
      3.1.3 Benchmarks .................................. 13
      3.1.4 Output ...................................... 14
1 Introduction

COMICS is a stand-alone tool which performs model checking and the generation of counterexamples for discrete-time Markov Chains (DTMCs). For an input DTMC COMICS computes an abstract system that carries the model checking information and uses this result to compute a critical subsystem, which induces a counterexample. This abstract subsystem can be refined and concretized hierarchically. The tool comes with a command line version as well as a graphical user interface which allows the user to interactively influence the refinement process of the counterexample. For more details on the approaches implemented in this tool, we refer to [2, 3, 1].

This is a preliminary version of the manual, which describes only the command-line version of COMICS.

We apologize that this manual is still in in a preliminary state. However, please consider to visit our homepage again in a few days where you will find a new version of this document!

http://www-i2.informatik.rwth-aachen.de/i2/comics/

In case of any questions or remarks, please feel free to contact

Nils Jansen
Theory of Hybrid Systems
RWTH Aachen
52056 Aachen
Germany
nils.jansen@cs.rwth-aachen.de
2 Input Format

The input format of COMICS may either describe the explicit representation of a DTMC or an abstract graph which can be concretized. The .dtmc-files describe a DTMC with target states and one unique initial state. .dtmc-files are the original input for COMICS. .tra- and .lab-files are the DTMC-input for MrMC and can be imported\(^1\) into COMICS. Abstract graphs are represented by .xml-files.

In this chapter we will introduce all formats in detail and give some examples. Consider the following DTMC with input state 1 and target states 3 and 8.

![Figure 2.1: Example DTMC](image)

2.1 The .dtmc-Format

The .dtmc-files contain the number of states and transitions of an DTMC, an unique initial state and an arbitrary number of target states are defined, and the transition probability matrix is given. The abstract syntax is defined as follows:

<table>
<thead>
<tr>
<th>Syntax of .dtmc-Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATES &lt;number of states&gt;</td>
</tr>
<tr>
<td>TRANSITIONS &lt;number of transitions&gt;</td>
</tr>
<tr>
<td>INITIAL &lt;initial state&gt;</td>
</tr>
<tr>
<td>(TARGET &lt;target state&gt;\n)* //Arbitrarily many target states</td>
</tr>
<tr>
<td>(&lt;start state&gt; &lt;end state&gt; &lt;probability&gt;\n)* //Transitions</td>
</tr>
</tbody>
</table>

States are presented as positive integers starting with 1. Transitions are given by two integers identifying the start state and the end state of the transition and its probability which has to be out of \([0, 1]\). The transitions have to be ordered in ascending order w.r.t. to firstly the start states and secondly the end states. Consider Example 2.2 which represents the example DTMC depicted in Figure 2.1.

\(^1\)So far only by using the graphical user interface.
2 Input Format

The .tra-format is the input format of MRMC and defined similar to the .dtmc-format. The only difference is that it contains no information about target states and input state. The .dtmc-format was designed to have the same syntax in order to guarantee the compatibility of both tools and to easily use the PRISM export facilities. Note, that - so far - .tra-files are only supported when using the graphical user interface of COMICS.

**Syntax of .tra-Files**

```plaintext
STATES <number of states>
TRANSITIONS <number of transitions>
(<start state> <end state> <probability>
∗
//Transitions
```

2.3 The .lab-Format

.lab-files contain information about the list of atomic propositions and their assignment to the states of a DTMC. It consists firstly of the declaration of a list of atomic propositions and secondly of states and a list of propositions they are labeled with. This list has to be subset of the declared list of propositions. Note, that - so far - .lab-files are only supported when using the graphical user interface of COMICS.

**Syntax of .lab-Files**

```plaintext
#DECLARATION
<proposition_list>
#END
(<state proposition_list>)∗
```

Consider Figure 2.3 and Example 2.4 which together define again the DTMC of Figure 2.1 while state 8 is additionally labeled with ERRORLABEL. Note, that one node can have several labels.
2.4 The .xml-Format

The .xml-format stores the whole abstract graph which results from SCC-based Model Checking [2]. The root node is named dtmc. It has children scc, target, and prob. scc has as attributes a unique id and a identifying node node0. It’s children are inp for input states, out for output states, vtx for the remaining states, and edge for the graph edges. Moreover, again scc can be child of scc. edge has a flag abs wether it is abstract or not. target identifies the target nodes of the DTMC and prob stores the Model Checking probability. Note, that for the .xml-input no Model Checking is performed, as the probability of reaching target states in the whole system is already saved inside the document. A deeper understanding of this format is not crucial for using COMICS, as the .xml-files should not be edited. They are only used to save the abstract graph structure, changes are done at the risk of having an inconsistent system. However, consider Example 2.5 which depicts the abstract graph resulting from SCC-based Model Checking. The .xml-file was generated from the input of Example 2.2 while a new target states 9 with edges coming from the former target states was inserted for computation purposes.

2.5 The .conf-Format

The .conf-files contains all information needed for a counterexample search. Although all parameters can also be set as command-line parameters or inside the graphical user interface, this simplifies structuring the tasks to perform.
2 Input Format

<table>
<thead>
<tr>
<th>TASK counterexample</th>
<th>modelchecking</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBABILITY_BOUND double</td>
<td></td>
</tr>
<tr>
<td>DTMC_FILE *.tra</td>
<td>*.xml</td>
</tr>
<tr>
<td>REPRESENTATION subsystem</td>
<td>pathset</td>
</tr>
<tr>
<td>SEARCH_ALGORITHM global</td>
<td>local</td>
</tr>
<tr>
<td>ABSTRACTION concrete</td>
<td>abstract</td>
</tr>
</tbody>
</table>

Syntax of .conf-Files

The main task may be the search for a counterexample or just getting the Model Checking result. A probability bound can be defined which shall not exceed the Model Checking probability of the input system, otherwise the counterexample search will not be carried out. The input files are of the formats mentioned above. The search algorithm can be specified. Finally, the user can choose between performing the hierarchical counterexample search on an abstract system or just search for a critical subsystem directly on the concrete graph.
2.5 The .conf-Format

Figure 2.5: Example .xml-file
3 Usage

In this chapter, we describe the usage of COMICS as well as the possibilities to get it to run. So far, we can only offer pre-compiled binaries for Debian systems, but very soon there will be the sources available.

3.1 Command-line Mode

If you are using one of the pre-compiled binaries and you are running a 64 bit Debian system, you can start COMICS by changing into directory comics-1.0 and typing

./comics

If this does not work, make sure that the binary file is marked as executable:

chmod +x comics

If you are running a 32 bit Debian system, you have to type

./comics_32

By invoking COMICS without parameters or by

./comics --help

the help output is displayed, which is depicted in Example 3.1.

Many of the basic parameters can already be predefined by loading a .conf-file. If parameters of the .conf-file and the command-line are inconsistent, those given in the command-line are chosen. We will shortly explain all of the possible parameters. Comics can be started with an input .conf-file by

./comics <inputFile.conf>

In case, a .dtmc-file was specified inside the .conf-file, the corresponding tasks are performed directly. If an .xml-file was specified, Model Checking will not be performed as the result is already saved inside the .xml-file. Thus, it makes no sense to invoke Model Checking as only task when loading an .xml-file.

3.1.1 Basic Options

--outputdtmc <filename>

The resulting critical subsystem is saved in the current directory as <outputFilename.dtmc>. If nothing is specified, a file “result.dtmc” is saved.

--outputxml <filename>

The resulting critical subsystem is saved as abstract graph possibly containing information about abstracted SCCs and the Model Checking result for the subsystem. The filename is <outputFilename.xml>
3 Usage

This is COMICS 1.0. Possible ways to call are:
./scc_mc_cex filename [options]
NOTICE: filepaths have to be relative to the executable!

with options:

--outputdtmc <filename> (Save result as <outputFilename.dtmc>)
--outputxml <filename> (Save result as <outputFilename.xml>)
--saveIterations (Save results of all iterations)
--dtmc (DTMC file only, no config file)
--abstract | --concrete (Abstract or concrete search algorithm)
--global | --local (Global or local path search)
--subsystem | --pathset (Subsystem- or path-based counter example)
--iterationcount <count> (Do <count> many iterations)
--stepsize <size> (One iteration has <size> concretization steps)

Heuristics for the choice of abstract state to concretize:
--choose_by_probability (By highest outgoing probability)
--choose_by_degree (By lowest input/output degree)
--choose_by_membership (By highest relative membership in subsystem)

Benchmarking:
--search-benchmark (Try global, local, concrete, abstract)
--concretization-benchmark (Try concretizing 1, sqrt, all SCCs in each step)
--heuristics-benchmark (Try different concretization heuristics)
--complete-benchmark (Combine all of the above benchmarks - slow!)

Default behavior: --abstract --global --closure

Figure 3.1: Example help-output

--saveIterations

Every time a critical subsystem is computed for an abstract graph, the corresponding .dtmc-file is stored in the current directory before further concretization is performed and a new computation is invoked.

--dtmc

A .dtmc-file is given as input instead of the standard .conf-file.

--abstract

The search for a counterexample is performed on the abstract graph involving concretization.

--concrete

The search for a counterexample is performed directly on the concrete graph.

--global

The global search approach is used to find a counterexample, which may be represented as a critical subsystem or a set of paths of the input system.
3.1 Command-line Mode

--local
The local search approach is used to find a critical subsystem.

--subsystem
The resulting counterexample is represented as critical subsystem.

--pathsum
The resulting counterexample is represented as set of paths.

--iterationcount <count>
The number of search iterations is fixed by <count>. Afterwards, the program terminates with a possibly still abstract counterexample.

3.1.2 Heuristics
We have implemented a large number of heuristics for the automatic choice of the next abstract state to concretize. The three heuristics which performed best are offered to choose by the user.

--choose_by_probability
The abstract node which has the highest outgoing probability inside the critical subsystem is chosen.

--choose_by_degree
The abstract node with the lowest input/output-degree is chosen.

--choose_by_membership
The abstract node with the highest number of ingoing and outgoing transitions is chosen. To this we refer as the highest-membership inside the critical subsystem.

In our tests, the highest-membership heuristic performed best, while the choice by input/output degree delivered mostly worse results than using no heuristics.

3.1.3 Benchmarks
In order to do extensive tests with our tool, the user can choose between a number of benchmarking options, were several tasks are performed.

--search-benchmark
A test for the global search vs. the local search is performed both on abstract and on concrete systems.

--concretization-benchmark
A hierarchical search on an abstract system is performed, where

1. At each step only 1 abstract SCC is concretized.

2. For $n$ visible abstract SCCs in the system, at each step $\sqrt{n}$ SCCs are concretized. (SQRT-heuristics).
3 Usage

3. At each step all abstract SCCs are concretized.

In our tests, the SQRT-heuristic performed best.

--heuristics-benchmark

The three heuristics for automatic choice of next abstract states are tested against each other, see Section 3.1.2.

3.1.4 Output

As mentioned before, the resulting critical subsystem is stored in the current directory either as .dtmc- or as .xml-file. Additionally, the intermediate results may be saved. In any case, a short summary of the search process is saved as

counter_example_summary.txt

An example of such a summary is depicted in Figure 3.2. The task was as given in crowds3_3.conf which is available in our benchmark set.

```
----------- STATISTICS ----------
Task: Counterexample for P(t)<0.18
Counter example size: 116 states, 173 transitions
Model Checking result of original system: 0.189123
Closure probability: 0.180516
Time of counter example computation: 0 secs [Without Pre-/Post processing]
Search params: ABSTRACT, LOCAL, CLOSURE-based
Base node selection for concretization: Select All Visible
Additional selection criterion: None
Number of concretizations per step: SQRT(#CANDIDATES)
During the 6 refinement steps the following number of shortest paths/closures were computed:

<table>
<thead>
<tr>
<th>step</th>
<th>#paths</th>
<th>#closures</th>
<th>#conc. scc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

Total #paths: 89, total #closures: 89, total #concretized SCCs: 15
```

Figure 3.2: Example: Summary-file of counterexample generation
Bibliography

