MPASS: An Efficient Tool for the Analysis of Message-Passing Programs*

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[1], Gaurav Saini[2], and Subham Modi[3]

[1] Uppsala University, Sweden
[2] Indian Institute of Technology, Ropar
[3] Indian Institute of Technology, Kanpur

Abstract. MPASS is a freely available, open source, tool for the verification of message passing programs communicating in an asynchronous manner over unbounded channels. The verification task is non-trivial as it involves exploring state spaces of arbitrary or even infinite sizes. Even for programs that only manipulate finite range variables, the sizes of the channels could grow unboundedly, and hence the state spaces that need to be explored could be of infinite size. MPASS addresses the bounded-phase reachability problem, where each process is allowed to perform a bounded number of phases during any run of the system. In each phase, a process can perform either send transitions or receive transitions (but not both). However, this doesn't bound the number of context switches between processes or the size of the channels but just the number of alternations between receive and send transitions of each process. Currently, MPASS can decide bounded-phase reachability problem for three types of channel semantics, namely lossy, stuttering and unordered channels. Messages inside these channels can be lost, duplicated and re-arranged, respectively. MPASS efficiently and uniformly reduces the bounded-phase reachability problem into the satisfiability of quantifier-free Presburger formula for each of the above mentioned semantics.

1 Introduction

MPASS is an open source and efficient tool for the verification of message-passing programs communicating in an asynchronous manner over unbounded channels. The verification problem such as the reachability problem are either undecidable [5] or have high complexity [1,8,10,12], even under the assumption that each process is finite-state.

MPASS verifies the reachability problem of message-passing programs or protocols in which each process is restricted to performing at most k phases (for some natural number k). A *phase* is a run where the process performs either send or receive operations (but not both). The bounded-phase restriction does not limit the *number* of sends or receives, and in particular it does not put any restriction on the length of the run or the size of the buffers.

MPASS can handle different variants of channel semantics, such as *lossy*, *stuttering*, and *multiset* that allow the messages inside the channels to be lost, duplicated, and reordered respectively.

* This research was in part funded by the Uppsala Programming for Multicore Architectures Research Center (UPMARC)

MPASS is based on the framework described in [2] which translates bounded phase reachability problem into the satisfiability of quantifier-free Presburger formula in polynomial time, along with several optimizations. This allows leveraging the full power of state-of-the-art SMT-solvers for obtaining a very efficient solution to the bounded-phase reachability problem for all above mentioned semantics.

MPASS turns this verification task into a push-button exercise, it can also be applied on a number of message-passing program with promising results. More precisely:

- MPASS is an open source [3] verification tool that analyzes reachability problem in message-passing programs having their phase bounded
- If MPASS finds a bug then it is a real bug of the input program.
- If the bound on the number of phases increases then the set of explored traces increases, and in the limit every run of the programs is then explored.
- It converts bounded phase reachability problem into quantifier-free presburger formula and then call SMT Solver to output the satisfiability results.
- It provides user with the computational time used both by the constraint generation as well as SMT solver to analyze the complexity of the provided protocol. Details regarding these two extractions are given in section 2.
- It provides user with the satisfiability results in an user-friendly manner by generating a separate tex file. More information regarding this can be found within the tool [3].

Targeted users MPASS can be used by these group of users :

1. Various researchers and Computer scientists can use the freely available and open source code of MPASS to compare with other approaches for the verification of message-passing programs, to improve and optimize the implemented techniques (by interfacing with other other efficient SMT-solver) or by improving the used data structures for handling the protocols, or to target new platforms and programs (e.g., adding shared variables or other channel semantics).
2. Teachers of distributed systems and concurrent programming classes can use (and augment) MPASS to familiarize their students with message-passing programs. In particular the precision of MPASS can concretely illustrate the difficulty of writing a correct concurrent programs.
3. Software developers who are working on message-passing programs can use MPASS to verify their tentative soluteions. It can also be used to check faultiness of a particular protocol by verifying the reachability of an Invalid state.

2 MPass Tool

MPASS performs two different levels of extraction in order to analyze the reachability problem for a given program. First it translates the protocol defined in `xml-files` into *Non-Deterministic Finite Automaton* (NFA) and then generates quantifier-free Presburger formula from this NFA .

2.1 Xml to Automata

Protocols are used in the format of `xml-files` present within the tool repository inside the 'Includes' folder. The protocols, thus, can be modified in a simple manner by changing the fields in xml-files. The first task of MPass is to translate the protocols defined in `xml-files` into *Non-Deterministic Finite Automata* (NFA). In order to achieve this, it takes xml-file path of the Protocol as an input and then uses C++ library of `lemon` to translate the protocol into NFA as described below:

```
<rule id="Q0__ack1__INBOUND">
  <pre>
    <current_state>Q0</current_state>
    <received_message>ack1</received_message>
    <channel>c1</channel>
  </pre>
  <post>
    <next_state>Q1</next_state>
    <send_message>mesg0</send_message>
    <channel>c0</channel>
  </post>
</rule>
```

Fig. 1. An example of xml code for ABP Protocol

The above rule adds two transitions from the state `Q0` to the state `Q1`. The first transition defines the rule of receiving the message `ack1` from the channel `c1` whereas the second transition defines the rule of sending the message `mesg0` into the channel `c0`. Each process in a protocol contains one or more of such rules which in together defines the automaton for that process within the given protocol.

Now, since for each process we bound the number of phases, where each phase contains either send or receive transitions (but not both), we make two automata for each process, one containing all except the receive transitions (send copy of that process) and the other containing all except the send transitions (receive copy of that process).

In this way, we have constructed *2*Number of Process* automata for the input protocol.

2.2 Automata to Quantifier-free Presburger Formulas

Verification of protocol for reachability problem is analyzed by generating a *quantifier-free Presburger formula* from the automaton constructed as shown in `figure 1` and then using the help of modern SMT solver namely *Z3 theorem prover* to check the satisfiability of this formula. More information regarding the translation of reachability for *bounded-phase-automata* into the satisfiability of *quantifier-free Presburger formulas* can be found in [2].

In order to generate the quantifier-free Presburger formula, some variables of a particular *form* have to be defined and thus, for all the transitions present within each automata, we'll introduce a number of variables as shown below in table Table 1:

Variable name	Variable code
Index-variable	i-var
Occurence-variable	o-var
Match-variable	m-var

Table 1. Variables associated with each transitions

Variable declaration and definition are explained briefly in [2].

Since we *bounded-phase* reachability problem, MPASS generates variables for both send and receive copy of each process and then duplicates them k times (where k is some natural number denoting the bound for the number of phases within each process) which are then further used to generate Presburger formulas. In this way we have ignored the inefficient process of making multiple copies for each process as done in [2].

If the result of the SMT-solver for these set of formulas (one of them being displayed in Figure 2) is satisfiable (sat), then the *Bad State* is reachable and we have an UNSAFE (U) condition otherwise we have a SAFE (S) condition, ie, *Bad State* is not reachable for the given bound.

$$(\texttt{occ}(t) = 1) \wedge (\texttt{occ}(t') = 1) \wedge (\texttt{index}(t) < \texttt{index}(t')) \implies (\texttt{match}(t) < \texttt{match}(t')).$$

Fig. 2. An example of a *quantifier-free Presburger formulas*

3 Implementation

MPASS tool is implemented in C++ language with the help of `lemon` and `pugixml` library. Tool is programmed in an user friendly manner and is, thus, easy for any further extension or use.

Optimisations. Various optimization techniques were implemented to increase the efficiency of checking the bounded-phase reachability problem (in comparison to the approach described in [2]) such as:

- *Ignoring multiple copies for each process.* Instead of making 'k' copies for each process (where k is some natural number denoting the bound for the number of phases within each process), we make only two copies per process (send and receive copy) as described in section 2.1.

- *Removal of strongly connected component.* We evaluate all the sets of strongly connected components in the send copy of each process. Then, we replace each strongly connected component by two new states. We add a send operation between these two newly added states if this operation appears in this set of strongly connected component. Out of these two added states, the initial state will now be the target state for all the transitions entering into this set of strongly connected component and the other state (final state) will be the source state for all the transition leaving from this set. Therefore, each send copy of each process is optimized in such a way to reduce the number of formed constraints .

4 Experimental Results

The tool is available on GitHub [3], where we also supply the source of all experiments listed below. The tool uses the frontend of the implementation provided by Marques et al in [6], to get XML representations of the protocols from spreadsheets.

Table 2 displays the results of running MPASS tool on some examples; whereas in Table 3, we show the results of running MPASS on some protocols which are intentionally modified to introduced some errors. The displayed examples are taking from [7] which are further described in [9] and [11]. Bounded Retransmission Protocol (BRP) is also adapted from [4].

Both the tables should be interpreted as follows :

Column P lists the name of protocol under analysis.
Column Bad lists the name of the process followed by the state in that process which we are trying to reach.
Column Sem lists the channel semantics used for the specific analysis.
Column channel lists the type of channel used in the message passing.
Column Constraint Generation, SMT and Total lists the time taken to generate the formula, time taken by the SMT solver to check the satisfiability of this formula and the total time of the analysis respectively.
Column Assert lists the number of assertions fed to the SMT solver
Column Ph. lists the bound on the number of phases, and finally,
Column Res lists the result of the analysis. The results are listed as **U** for Unsafe or Bad state is reachable, ie, **sat** or **S** for possibly safe or Bad state is not reachable (under the bounded-phase assumption), ie, **unsat**.

Results displayed here are for few protocols only. Rest of the results are available online at [3] in the 'MPass_result' folder which is inside the 'MPass-master' directory.

P	Bad	Sem	Channel	Const. gen.	SMT	Total	Assert	Ph.	Res
ABP	RECEIVER Invalid	UCS	process	0.07 sec	74 sec	74.07 sec	4025	4	U(sat)
ABP	RECEIVER Invalid	LCS	process	0.04 sec	1 sec	1.04 sec	2519	3	S(unsat)
ABP	RECEIVER Invalid	SLCS	process	0.03 sec	2 sec	2.03 sec	2519	3	S(unsat)
BRP	RECEIVER Invalid	UCS	process	0.3 sec	2 sec	2.3 sec	23461	3	S(unsat)
BRP	RECEIVER Invalid	LCS	process	0.28 sec	2 sec	2.28 sec	23461	3	S(unsat)
BRP	RECEIVER Invalid	SLCS	process	0.28 sec	1 sec	1.28 sec	23461	3	S(unsat)
STP	A Invalid	UCS	process	0.03 sec	4 sec	4.03 sec	2354	6	U(sat)
STP	A Invalid	LCS	process	0.02 sec	0 sec	0.02 sec	1348	4	S(unsat)
STP	A Invalid	SLCS	process	0.02 sec	0 sec	0.02 sec	1348	4	S(unsat)

Table 2. Verification Results for examples from [9] and [7]

P	Bad	Sem	Channel	Const. gen.	SMT	Total	Assert	Ph.	Res
ABP_F	RECEIVER Invalid	SLCS	process	0.04 sec	1 sec	1.04 sec	1275	2	U(sat)
ABP_F	RECEIVER Invalid	UCS	process	0.04 sec	1 sec	1.04 sec	1275	2	U(sat)
SlidingWindow_F	RECEIVER Invalid	SLCS	process	0.02 sec	0 sec	0.02 sec	913	1	U(sat)
SlidingWindow_F	RECEIVER Invalid	UCS	process	0.03 sec	0 sec	0.03 sec	913	1	U(sat)
Synchronous_F	B Invalid	SLCS	process	0.02 sec	0 sec	0.02 sec	713	3	U(sat)
Synchronous_F	B Invalid	UCS	process	0.02 sec	0 sec	0.02 sec	713	3	U(sat)

Table 3. Verification Results for buggy (faulty) examples

References

1. P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *LICS*, 1993.
2. P. A. Abdulla, M. F. Atig, and J. Cederberg. Analysis of message passing programs using smt-solvers. Uppsala University.
3. P. A. Abdulla, M. F. Atig, S. Modi, and G. Saini. https://github.com/vigenere92/MPass.
4. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. In *FMSD*, page 25(1):39–65, 2004.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
6. A. P. M. Jr., A. Ravn, J. Srba, and S. Vighio. csv2uppaal. https://github.com/csv2uppaal.
7. A. P. M. Jr., A. Ravn, J. Srba, and S. Vighio. csv2uppaal. https://github.com/csv2uppaal.
8. R. Lipton. The reachability problem requires exponential time. Technical Report TR 66, 1976.
9. A. P. Marques, A. P. Ravn, J. Srba, and S. Vighio. Tool supported analysis of web services protocols. In *TTCS*, pages pages 50–64, University of Oslo, 2011.
10. C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
11. A. P. Ravn, J. Srba, and S. Vighio. Modelling and verification of web services business activity protocol. In *TACAS*, LNCS, pages pages 357–571. Springer, 2011.
12. P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, Sept. 2002.

A Installation

This appendix is intended to guide the reader to use of MPASS . It will first describe installation of MPASS , then give a tutorial on using the MPASS . MPASS can be downloaded from [3] (https://github.com/vigenere92/MPass). The sources are available as a tar ball via the "Downloads" section (recommended) as well as by git clone.

A.1 Requirements

- A C++ compiler supporting C++11. For example g++ version 4.6 or higher.
- Z3 SMT 2.0 Solver.
- Lemon Graph Library.
- pdflatex (Optional).

The Z3 SMT Solver can be downloaded from http://z3.codeplex.com. It should be installed in the bin directory, so that it can be called using "z3 -smt2 filename" from the terminal. Lemon Graph Library can be downloaded from http://lemon.cs.elte.hu. pdflatex is an optional feature. It is required only for creating pdf's for the results for various runs.

A.2 Installation

The MPASS Tool can be installed by using the following procedure.

- Copy the MPass_result folder from the MPass directory to your home folder. This is done to make the MPass tool always consistent irrespective of the location where parent directory is downloaded by the user.
- Using terminal, go to the location of the MPass folder and execute the following commands.

```
$ touch NEWS README AUTHORS ChangeLog
$ autoreconf --force --install
$ ./configure
$ make
$ sudo make install
```

A.3 Installation Options

The configure script is built with GNU autotools, and should accept the usual options and environment variables.

Changing Installation Directory
The command 'sudo make install' will install MPass Tool in the directories which are standard on your system. To override this behaviour add the switch –prefix to the './configure' command:

$ ./configure –prefix=/your/desired/install/path

B MPASS Tutorial

This section gives you a short tutorial on the usage of MPASS .

B.1 Creating the Settings File

Settings file contains the information on the path to the XML file, semantics of the channels, bound, the states for which the reachability is to be checked and the type of channels to be formed for calculating the reachability.The format of Settings file is as follows:

file	path_to_xml_file
semantics	channel_semantics
bound	bound_for_the_processes
bad	bad_process bad_state
channel_type	type_of_channel

`path_to_xml_file` is path to the xml-files of protocols present inside the MPass directory. So before calling the MPass tool, change `path_to_xml_file` argument inside the 'Settings.txt' file. By default, xml-files are present in the 'Includes/Protocols' folder which is inside the 'MPass directory'.

B.2 Description of Settings File

- **file**: It refers to the location of the XML file to be verified.
- **semantics**: It refers to the semantics of the channels. It can be lossy, stuttering or unordered.
- **bound**: It refers to the number of bounds for the automata.
- **bad**: It takes pairs of values. First is the name of the process in which the reachability is to be checked and the second is the set of states for which reachability is being checked.
  NOTE: *Bad state for more one process can also be entered one after the other in the same format as diplayed in the example below.*

  Example:

  ```
  bad SENDER Q0 Q1 RECEIVER INVALID
  bad SENDER Q0 RECEIVER INVALID
  bad RECEIVER INVALID
  ```

- **channel_type**: It refers to the type of channel. It can be :

  `prefix` : For making channels based on the first alphabet of the messages.
  `process`: For making one channel for each process.
  `xml` : For making channels in the same way as specified in the xml file of the protocol.

Figure 3 shows a sample Settings file for the Alternating Bit Protocol with its XML file(ABP.xml) in the default 'Includes/Protocols' folder.

```
file /home/rator/MPass/Includes/Protocols/ABP.xml
semantics lossy
bound 3
bad RECEIVER INVALID
channel_type process
```

Fig. 3. An example of Settings file

In case the path to XML file for the protocol is incorrect in the Settings file, you will get the following error message:

```
----------
----------
; loading xml file
Xml File not found!
Exiting
```

If so, please make sure the path to the XML file for the protocol is correct.

B.3 Running the MPASS for the given Protocol

The MPASS can be called from the command line using 'MPass' followed by `Path to the Settings file` which is expected as the first argument.By default, it is present in the 'src' folder which is inside the 'MPass directory'.Thus MPass tool can be called as (if Settings file path is not changed)

```
MPass src/Settings.txt
```

In case the Settings file is not found, you will receive the following error message:

```
Settings File not found
```

If so, please make sure the path to Settings file is correct with respect to the current directory.
When the MPASS is called for a Settings file, it prints the results in 4 sections. In the first section it prints (step-by-step) various operations that are being applied to the Automata. Figure 4 shows the screen dump, when MPASS is run on the Settings file mentioned in Figure 3.

————-
————-
; loading xml file

; adding states for process SENDER
; adding transitions for process SENDER
; adding states for process RECEIVER
; adding transitions for process RECEIVER
; removing send loops
; declaring index and occurence variables for send transitions
; declaring index, occurence and match variables for receive transitions
; declaring index and occurence variables for eps transitions between two phases
; declaring index and occurence variables for send transitions
; declaring index, occurence and match variables for receive transitions
; declaring index and occurence variables for eps transitions between two phases
; adding in and out flow constraint
; Initial state constraint
; Final state constraint
; adding computation order constraint and ensuring connectivity
; adding in and out flow constraint
; Initial state constraint
; Final state constraint
; adding computation order constraint and ensuring connectivity
; declaring distinct index condition for each pair of non unique transitions
; enforcing matching of each occuring receive transitions
; enforcing computation order for transitions pair per channel

Fig. 4. Operations applied to the Automata

After applying various operations to the automata of the protocol, it then prints out the statistics of the automata, ie, the number of states and transitions that were added to the Automata. Figure 5 shows the screen dump for the statistics on Automata. Detailed information regarding the notations of transitions can be found in [2]

————-AUTOMATA STATISTICS————-
Total Number of non-unique transitions: 249
Total Number of unique transitions: 6
Total Number of states: 156
Number of Assertions : 2519

Fig. 5. Automata Statistics

After displaying the statistics of the automata formed, MPass then prints out the time taken to generate quantifier-free Presburger formula and along with the time taken by Z3 SMT-solver to compute the satisfiability of the generated quantifier-free Presburger formula. Figure 6 shows the screen dump for the time statistics.

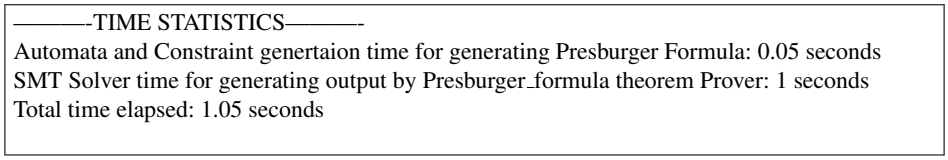

Fig. 6. Time Statistics

In this fourth and the final section, MPass display the results that whether the given state is reachable or not. Along with this it also displays the values that were provided in the Settings file. Figure 7 shows the screen dump for the result.

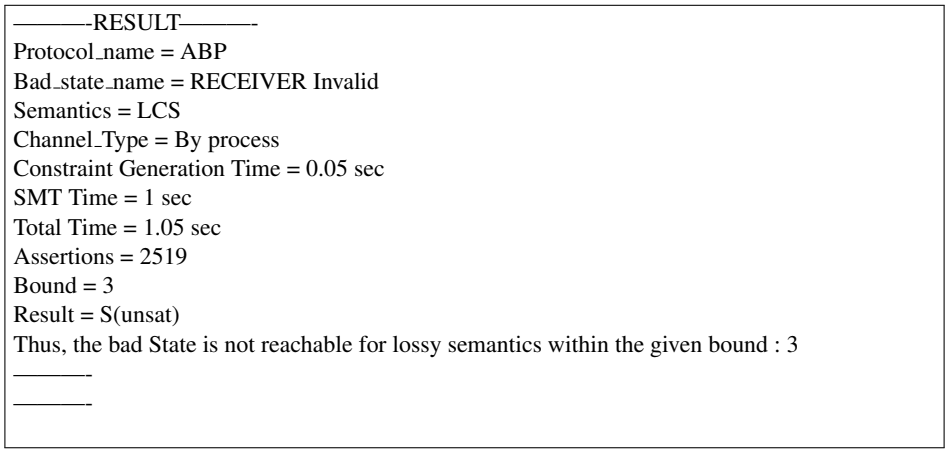

Fig. 7. Reachability Result

B.4 Other Optional Arguments

A pdf containing the results of the MPass can also be generated if pdflatex is installed in the system.This can be done using `new` or `old` arguments in addition to the one which we were using before. In addition to the pdf containing the result, two text files, one containing the Presburger formula generated for the Automata and the other containing the result of SMT solver are also generated. These files are placed in MPass_result folder. This option can be used as follows :

1. Using the new option:
A new pdf containing the results can be generated using this option. The pdf is placed in the `MPass_result/Result` folder in the home directory.
It is used as follows :

```
MPass path_to_settings_file new
```

2. Using the old option:
The result for this run of the MPass is appended to the previously genereated pdf. The pdf is placed in the `MPass_result/Result` folder in the home directory.
It is used as follows :

```
MPass path_to_settings_file old
```