The PaMC Manual

1 Introduction

PaMC is a model checker specialized for verifying safety properties of parameterized systems which consists of an indefinite number of identical processes running in parallel. It implements a very powerful abstraction technique known as parameter abstraction and guard strengthening, using a heuristics-based automatic procedure to compute auxiliary invariants. PaMC is constructed on the top of the TLV model checker. Its syntax parser and BDD library operation procedure are adapted directly from TLV. However, PaMC implements the whole procedure of the parameter abstraction, invariants computing and guard strengthening procedure by itself. PaMC is not a stand-alone model checker. It uses TLV as its default model checking engine. It can also use any model checker which accepts SMV input language, such as Cadence SMV and NuSMV, as its model checking engine.

2 Installation

Please check the following libraries installed in your system before getting PaMC to compile:

- flex
- bison

If you choose the -m32 option in Makefile for 64 bit linux system, the gcc-multilib library is needed.

You can also use the executable files directly. We have tested them in Debian, Ubuntu, Fedora and CentOS Linux environment.

Step 1 Download the zipped executable or source files of PaMC. The latest versions are PaMC-0.3.1-bin.tgz and PaMC-0.3.1-src.tgz respectively. You can use the archive to compile PaMC on Linux systems as a Bash Shell user. If you choose the executable files, the make procedure in step 2 is omitted.

Step 2 Save the file to one directory and uncompress the archive. We'll assume this directory is /home/yourname, but you can uncompress it wherever you want.

```
$ tar −xvzf PaMC−0.3.1−src.tgz
$ cd /home/yourname/PaMC−0.3.1/src
$ make
$ make install
```

Step 3 Put /home/yourname/PaMC-0.3.1/bin in your PATH variable. In bash use something like the following:

```
$ export PATH=$PATH:/home/yourname/PaMC−0.3.1/bin
```

Step 4 To run PaMC on a file file.pam simply do

```
$ PaMC −f file.pam
```

3 Input language

The input language pam of PAMC can be seen as a parameterized extension of SMV language. Here we only describe the new parameterized features of pam. For more details on SMV language we refer the users to the SMV Manual.

For the sake of describing parameterized systems, it uses the reserved word Proc to denote the parameter of the system. The parameterized system variables ranging over Proc can be declared as follows:

```
VAR
    CurPtr : Proc;
    InvSet : array Proc of boolean;
    MSG[Proc] : array Proc of Proc;
```

There are two types of array of modules which are indexed by the parameter Proc. The structure modules describe the system's variables hierarchically, while the transition modules define the transitions of the system with the reserved word transition prefixed to their declarations. For instance,

```
VAR
    array Node[Proc] : node_module(Proc);
```

Transitions are given in the usual guarded command form in transition module. The reserved word transition guarantees that all transition modules can be nondeterministically asynchronously interleaving executed.

```
VAR
    array Request[Proc] : transition request(Node[Proc].State, x);
    array Enter[Proc] : transition enter(Node[Proc].State, x);
```

The parameterized properties to be verified are denoted with prefix FORALL reversed word, and appear after the reserved word SPEC.

```
SPEC
   FORALL(i,j)(i!=j -> !(Node[i].State=crit & Node[j].State=crit))
```

Only part of pam input language has been implemented in PaMC. Other parts will be implemented in the future.

PaMC can accept input model written by SMV language directly. However, there are some restrictions on SMV language for describing parameterized systems. The word `N` is reserved in SMV model to determine the number of processes in reference model, and we need to assign the value of `N` in the smv file beforehand. The value of `N` is set to 3 by default. However, it is set to 4 for German-2004 protocol. The words `array of 1..N` of types are also reserved which be used to define indexed array parameterized variables. All array indexed modules in SMV model must be expanded and instantiated according to the number `N`.

4 Options

PaMC is a bash shell script, and can be run with different options that are shown below:

ProcNum(-p val)	Denote the process number of reference model needed by parameter abstraction. It is just the number of preserved process in abstract model plus one. This value is set to 3 by default.
MCEngine(-m val)	Set the model checking engine. Cadence SMV is used while value is set to 1, and NuSMV is used while value is set to 2. TLV is used by default.
Scale(-s val)	-v denote all intermediate file will be reserved, which can be used in the procedure of analyzing the counterexample.
Verbose(-v)	Denote the size of BDD's cache and key table with respect to the system's scale. 2 for large scale, 1 for middle scale, 0 for small scale by default.
File(-f file)	Denote the file need to be verified. Either pam file or smv file can be accepted by PaMC.

5 Examples

We use the MUX-SEM mutual exclusion algorithm as an example to illustrate the modeling language of PaMC.

```
1 MODULE main
```

```
VAR
  x : boolean;
  array Node[Proc] : node_module;
  array Request[Proc] : transition request(Node[Proc].State, x);
  array Enter[Proc] : transition enter(Node[Proc].State, x);
  array Release[Proc] : transition release(Node[Proc].State, x);
  array Idling[Proc] : transition idling(Node[Proc].State, x);
ASSIGN
  init(x) := 1;
SPEC
  FORALL(i,j)(i!=j -> !(Node[i].State = crit & Node[j].State = crit))
MODULE node_module
VAR
  State : {idle,try,crit,exiting};
ASSIGN
  init(State) := idle;

MODULE request(state, semaphore)
ASSIGN
  next(state) :=
    case
      state = idle : try;
      1 : state;
    esac;
MODULE enter(state, semaphore)
ASSIGN
  next(state) :=
    case
      state = try & semaphore : crit;
      1 : state;
    esac;
  next(semaphore) :=
    case
      state = try & semaphore : 0;
      1 : semaphore;
    esac;
MODULE release(state, semaphore)
ASSIGN
  next(state) :=
    case
      state = crit : exiting;
      1 : state;
    esac;
MODULE idling(state, semaphore)
ASSIGN
  next(state) :=
    case
      state = exiting : idle;
      1 : state;
```

```
51      esac;
52    next(semaphore) :=
53      case
54        state = exiting : 1;
55        1 : semaphore;
56      esac;
```