BPPChecker: An SMT-based Model Checker on Basic Parallel Processes

This is a Python tool and library that uses the Z3 SMT solver(4.8.5) for verifying a subclass of CTL on BPP, which includes bounded model checking liveness over BPP and deciding reachability of BPP. Basic Parallel Process (BPP), as a subclass of Petri nets, can be used as a model for describing and verifying concurrent programs with lower complexity.

Build and Run

The environment we recommend is Python3.8. We use the python package management setuptools and give a setup.py to help users install the dependencies automatically. You can install the package in the root directory with:

```
% python setup.py install
```

You need to install the lark library if it is not installed in your environment:

```
% pip install lark
```

If the paper is accepted, we can package and distribute BPPChecker as a library.

ACS2BPP

We realize the support of transferring Actor Communicating System model to BPP model named as ACS2BPP module. In our work, we give Actor Communicating System (ACS) the over-approximation BPP-based semantics to reduce ACS to BPP. With the support of the Erlang verifier Soter and ACS2BPP module, we can easily transfer Erlang programs to ACS and then verify EF-formulas defined safety properties on ACS.

You can build the ACS2BPP module (if the executable file ACS2BPP is not contained in the directory) with:

```
% g++ ACS2BPP.cpp -o ACS2BPP
```

Then given the ACS input, use ACS2BPP to get the over-approximation BPP of an ACS:

```
% ./ACS2BPP <acs_file> <bpp_file>
```

Model Checking on BPP

The executable module of BPPChecker is the run.py in the root directory. Here is the description of usage:

```
usage: run.py [-h] [-b BOUND] [-o OUT] file

BPPChecker [version 1.0.0]

positional arguments:
  file                  input file

optional arguments:
  -h, --help            show this help message and exit
  -b BOUND, --bound BOUND
                        the bound in bounded model checking liveness
  -o OUT, --out OUT     output file for additional information(unsat constraints)
```

To test the case given in the directory benchmarks/, you can use:

```
% python3 run.py ./benchmarks/ring_bpp
unsat
time used: 0.13913s
```

Bounded Model Checking EG-formulas (liveness)

BPPChecker supports the bounded model checking of EG-formulas on BPP, so you can input the bound k as parameter through command line:

```
% python3 run.py -b <k> <BPP file>
```

You can also run the script we give and here is a successful execution of bounded model checking with various step size k:

```
% cat test_bl_exp.sh
for k in $(seq 0 2)
do
  echo "-----result of bl_$k-----"
  python3 run.py -b 5 ./benchmarks/"bl_$k"
  python3 run.py -b 10 ./benchmarks/"bl_$k"
  python3 run.py -b 20 ./benchmarks/"bl_$k"
  python3 run.py -b 50 ./benchmarks/"bl_$k"
done

% sh test_bl_exp.sh
-----result of bl_0-----
sat
time used: 0.03114s
sat
time used: 0.05414s
sat
time used: 0.10571s
```

```
sat
time used: 0.36295s
-----result of bl_1-----
sat
time used: 0.04535s
sat
time used: 0.07605s
sat
time used: 0.14085s
sat
time used: 0.36743s
-----result of bl_2-----
sat
time used: 0.10960s
sat
time used: 0.35071s
sat
time used: 1.30813s
sat
time used: 13.31456s
```

Benchmarks

The benchmark we use are offered by Osualdo's work named Soter, an automatic and effcient ACS-based model checking tool for Erlang. The BPP models with formulas of a subclass of CTL are located in the directory benchmarks/. The grammar-defined input of BPP is:

Input Grammar of BPP

```
PROBLEM → BPP "formula" FORMULA
BPP → "initial" SYMBOLS "rules" RULES
SYMBOLS → VAR | SYMBOLS "," SYMBOLS
RULES → RULE | RULES RULES
RULE → VAR "→" SYMBOLS | VAR "→" LABEL "→" SYMBOLS

FORMULA → ATOM
        | UNARY "(" FORMULA ")"
        | BINARY "(" FORMULA "," FORMULA ")"
        | NEXT "(" LABEL "," FORMULA ")"

ATOM → ACC COMPARE NUMBER
ACC → MULT | ACC CONNECT ACC
MULT → VAR | VAR "*" NUMBER

CONNECT → [+-]
COMPARE → "==" | "!=" | ">=" | "<=" | ">" | "<"
UNARY → "Neg" | "EG" | "AF" | "EF"
BINARY → "Conj" | "Disj" | "Imp"
NEXT → "EX" | "AX"

VAR → [a-zA-Z][a-zA-Z0-9]*
```

```
LABEL → [a-zA-Z][a-zA-Z0-9]*
NUMBER → [1-9][0-9]*
```

We experimentally compare BPPChecker with Soter's backend BFC, where BFC verifies reachabillity (i.e. EF-formula) on ACS of Erlang programs and BPPChecker verifies BPPs generated by our ACS2BPP module. So we also give the test benchmarks with the suffix .mdrp we used on BFC, which are located in the directory bfc/. The document of BFC can be found on the website of BFC: https://mjolnir.cs.ox.ac.uk/soter/doc/index.html. You may try to test cases on BFC use instructions like:

```
% cd bfc
% ./bfc --target "0|61,61" ./parikh.mdrp
```

Related Publications

The whole algorithms and detailed implementation can be found in our full version of paper:

Zhao Ying, T.J., Guoqiang, L.: BPPChecker: An SMT-based Model Checker on Basic Parallel Processes(Full Version) (2021), http://arxiv.org/abs/2110.09414