Objectives and Instructions for Artifact Submission with Paper No. 194

1 Introduction

This artifact is associated with Submission No. 194, paper titled *Synthesis of co-ordination programs from linear temporal logic*.

The artifact in its current form is a prototype implementation of our synthesis procedure for a coordinating program from an *input specification* consisting of three parts (a). a *single* CSP description of the environment, and (b) safety LTL specification and (c) a liveness LTL specification. The implementation details can be found in Section 7.1 of the submitted paper.

1.1 Objective of the artifact evaluation

Our goal with the artifact evaluation is to demonstrate the reproducibility of the results presented in Section 7.2 of the paper.

The following observations MUST be reproducible:

1. OneHot-encoding is faster than QBF encoding.
2. If an input specification is reported as realizable in Section 7.2, its run with the OneHot-encoding must return realizable by synthesizing a controller.
3. If an input specification is reported as unrealizable in Section 7.2, its run with both, the OneHot-encoding and the QBF-encoding, is expected to Timeout.

The following observations MAY NOT be identical:

1. Exact runtime may differ from what is reported. We expect the runtime to be slower on the VM.
2. For realizable input specifications, the synthesized coordinator may differ from the one presented in the paper. This is because the constraint solver on top of which our prototype is built-on may return different solutions in different runs.

2 Details on the Artifact

Our artifact is a VirtualBox VM Image of an Ubuntu 16.04 system on which all the necessary dependencies to run our prototype have been installed.

Username: anon194

Password: anon194

2.1 File structure

- All major folders are present at ~/csp-synthesis/code/PyAsynchSynth
- The three folders at this location are
  1. bosy/

     This folder contains our local version of the synchronous synthesis tool BoSy. Our local version adds the feature of the OneHot-encoding on top of the original source code available for BoSy.
  2. poplbenchmarks/

     This folder contains files for all the CSP environment benchmarks that have been evaluated in Section 7.2. The CSP files use the extension .csp.

     Details on the .csp file format are given in the local README file.
  3. pycsp/

     This folder contains all source files of our prototype. It also contains a file named INSTRUCTIONS that contains all the commands the reviewer may execute to examine our results.

     We have two main source files (a) fdrextract.py and (b) run.py.

     Details of fdrextract.py, (b) run.py and (c) INSTRUCTION are given below.

2.2 fdrextract.py

This prototype reads in a CSP description of the environment written in the language CSP_M, and produces its flattened description by taking the synchronized product of the CSPs present in the CSP description.

We use this tool to flatten the CSP descriptions of of the thermostat in Section 7.2.2 and the arbiters in Section 7.2.3.

The CSP_M description of the thermostat is present in the file ~/csp-synthesis /code/PyAsyncSynth/poplbenchmarks/CSPFiles/thermostat.csp. It contains formal descriptions of Fig 6 - Fig 8 from the paper, and the description of ENV from Section 7.2.2.

Similarly, the CSP_M description of the arbiter with n processes is present in the file ~/csp-synthesis /code/PyAsyncSynth/poplbenchmarks/CSPFiles/arbiter$n$.csp. The file also contains a description of $\mathsf{ENV_n}$ from Section 7.2.3.

The application of fdrextract.py on the aforementioned files generates a single/flat CSP environment that represents the appropriate synchronization product in each case.

For sake of ease, we have already generated the flattened CSP environment in these three cases. The flattened CSP files are present in the folder poplbenchmarks/ under the name of thermostatflat.csp, arbiter2flat.csp, and arbiter3flat.csp.

An interested reviwerer may find instruction on using fdrextract.py in the README file at its location.

2.3 run.py

This is the prototype for our coordination synthesis procedure. It takes a flattened CSP environment E, safety specification S and liveness specification L, and synthesizes a controller/coordination program C such that $E|||C$ satisfies (S, L), if such a C exists. It returns "unrealizable" otherwise.

Details on using run.py may be found in the README file at its location. We illustrate the output on some sample executions of run.py .

2.3.1 When input specification is realizable

Here is the output of the prototype for input specification with safety specification false, liveness specification $\mathsf{FG}(\neg \mathsf{b})$ and environment CSP from Example 0. We know that this is realizable.

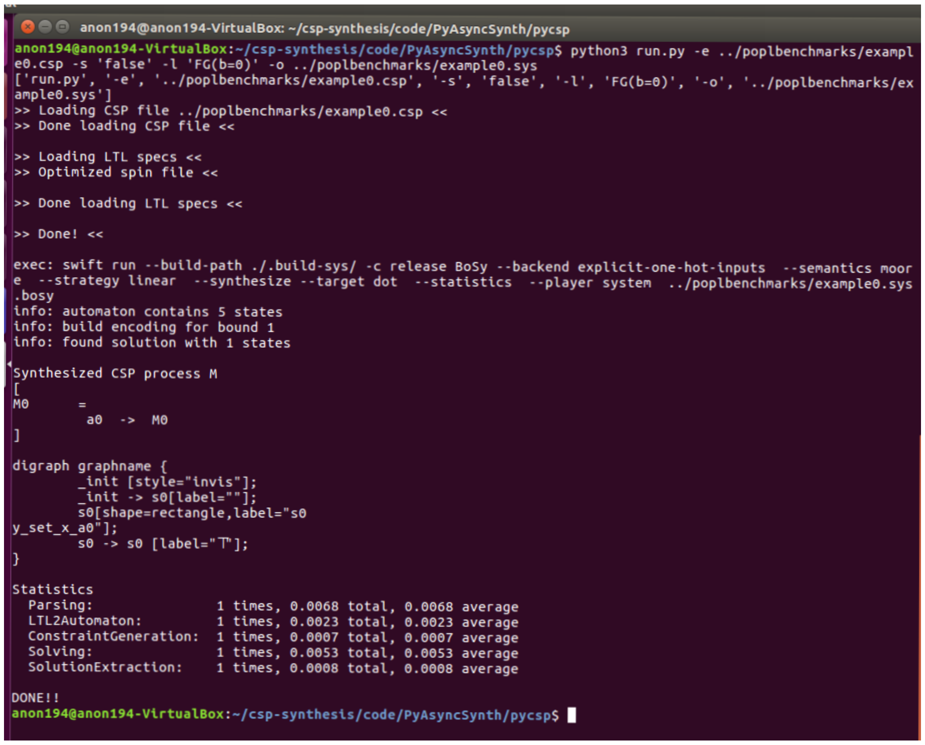
With one-hot encoding The output obtained from our prototype tool with the OneHot-encoding is given in Figure 1.

Here the CSP environment is stored in the file ../poplbenchmarks/example0.csp. For this part, we assume that all the CSP files are given as a single environment. The reduction from multiple environments to a single will be explained later in this document. The safety and liveness specifications are coupled with the -s and -l flags respectively. The rootname of the intermediate and final output files is coupled with the -o flag.

First, the CSP files, and both the LTL inputs are read into the program. The CSP environment and the LTL are then converted into the *specification automaton* using the symbolic construction described in Section 6 of the paper. The specification automaton is stored in the file ../poplbenchmarks/example0.sys.temp.spin. Note that the rootname is taken from the -o flag. The specification automaton is optimized by reducing its number of states. The optimized specification automaton is stored in the file ../poplbenchmarks/example0.sys.spin. The end of this phase is marked by the following sentence in the output: >> Done! <<

The next phase is where synchronous synthesis of the specification automaton occurs. The command exec: swift run –build-path ... invokes the synchronous synthesis tool BoSy on the specification automaton. The input file to BoSy is ../poplbenchmarks/example0.sys.bosy. This file takes the specification automaton from ../poplbenchmarks/example0.sys.spin for synchronous synthesis.

There are four parts to the rest of the output



```
anon194@anon194-VirtualBox: ~/csp-synthesis/code/PyAsyncSynth/pycsp
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$ python3 run.py -e ../poplbenchmarks/exampl
e0.csp -s 'false' -l 'FG(b=0)' -o ../poplbenchmarks/example0.sys
['run.py', '-e', '../poplbenchmarks/example0.csp', '-s', 'false', '-l', 'FG(b=0)', '-o', '../poplbenchmarks/ex
ample0.sys']
>> Loading CSP file ../poplbenchmarks/example0.csp <<
>> Done loading CSP file <<

>> Loading LTL specs <<
>> Optimized spin file <<

>> Done loading LTL specs <<

>> Done! <<

exec: swift run --build-path ./.build-sys/ -c release BoSy --backend explicit-one-hot-inputs  --semantics moor
e  --strategy linear  --synthesize --target dot  --statistics  --player system  ../poplbenchmarks/example0.sys
.bosy
info: automaton contains 5 states
info: build encoding for bound 1
info: found solution with 1 states

Synthesized CSP process M
[
M0       =
          a0  ->  M0
]

digraph graphname {
        _init [style="invis"];
        _init -> s0[label=""];
        s0[shape=rectangle,label="s0
y_set_x_a0"];
        s0 -> s0 [label="T"];
}

Statistics
  Parsing:                 1 times, 0.0068 total, 0.0068 average
  LTL2Automaton:           1 times, 0.0023 total, 0.0023 average
  ConstraintGeneration:    1 times, 0.0007 total, 0.0007 average
  Solving:                 1 times, 0.0053 total, 0.0053 average
  SolutionExtraction:      1 times, 0.0008 total, 0.0008 average

DONE!!
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$
```

Figure 1: Realizable sample with OneHot encoding

1. The info: statements

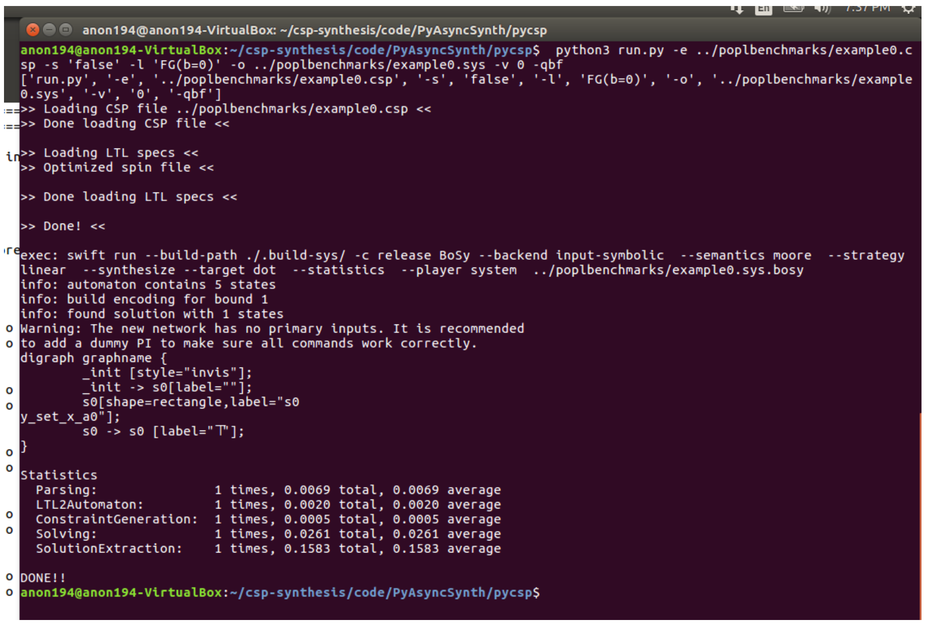
   Statement info: automaton contains k states: The automaton in ../poplbenchmarks/example0.sys.spin has k states.

   The other info: statements indicate progress of BoSy. Statement info: build encoding for bound k indicates that BoSy is checking if there exists a controller with k states such that the controller satisfies the input specification/specification automaton. If BoSy succeeds in finding a controller with k states, then it outputs info: found solution with k states. Else, it searches for a solution with $k + 1$ states. Note that BoSy guarantees that if there exists a solution with k states, i.e if the input specification is realizable, it will find it.

2. The solution CSP

   The solution controller obtained by BoSy is given in Synthesized CSP process M [. . .]. The initial state of the output CSP is always M0. The CSP follows notation given in the paper.

3. Raw BoSy output



```
anon194@anon194-VirtualBox: ~/csp-synthesis/code/PyAsyncSynth/pycsp
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$  python3 run.py -e ../poplbenchmarks/example0.c
sp -s 'false' -l 'FG(b=0)' -o ../poplbenchmarks/example0.sys -v 0 -qbf
['run.py', '-e', '../poplbenchmarks/example0.csp', '-s', 'false', '-l', 'FG(b=0)', '-o', '../poplbenchmarks/example
0.sys', '-v', '0', '-qbf']
>> Loading CSP file ../poplbenchmarks/example0.csp <<
>> Done loading CSP file <<

>> Loading LTL specs <<
>> Optimized spin file <<

>> Done loading LTL specs <<

>> Done! <<

exec: swift run --build-path ./.build-sys/ -c release BoSy --backend input-symbolic  --semantics moore  --strategy
linear  --synthesize --target dot  --statistics  --player system  ../poplbenchmarks/example0.sys.bosy
info: automaton contains 5 states
info: build encoding for bound 1
info: found solution with 1 states
Warning: The new network has no primary inputs. It is recommended
to add a dummy PI to make sure all commands work correctly.
digraph graphname {
        _init [style="invis"];
        _init -> s0[label=""];
        s0[shape=rectangle,label="s0
y_set_x_a0"];
        s0 -> s0 [label="T"];
}

Statistics
  Parsing:               1 times, 0.0069 total, 0.0069 average
  LTL2Automaton:         1 times, 0.0020 total, 0.0020 average
  ConstraintGeneration:  1 times, 0.0005 total, 0.0005 average
  Solving:               1 times, 0.0261 total, 0.0261 average
  SolutionExtraction:    1 times, 0.1583 total, 0.1583 average

DONE!!
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$
```

Figure 2: Realizable sample with QBF encoding

The solution CSP presented in the previous part is extracted form the raw output generated by BoSy. The raw output is given as a transition system in .dot format. This is given in digraph graphname $\{\ldots\}$.

4. Statistics

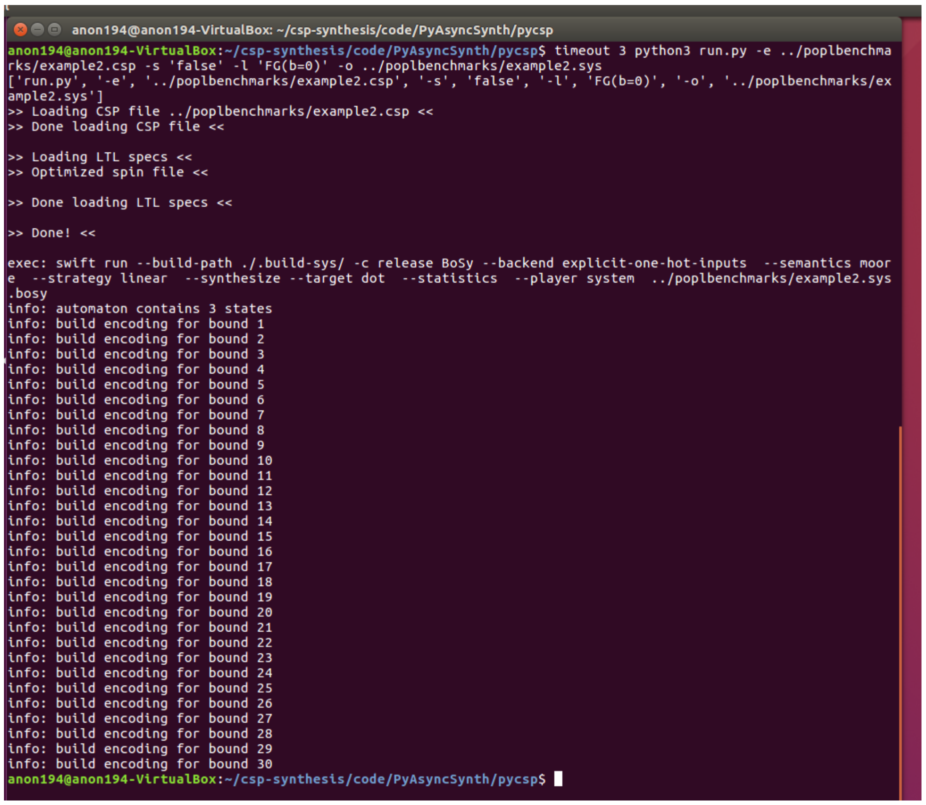
   The statistics is runtime information collected by BoSy during its execution. This does not include the time taken in constructing the specification automaton.

With QBF encoding The output in this case, illustrated in Figure 2, is similar to the one from the previous case. The only difference is that we have not implemented a procedure to extract a clean CSP structure from the raw transition systems that BoSy outputs.

2.3.2 When input specification is unrealizable

Here is the output of the prototype for input specification with safety specification false, liveness specification $\mathsf{FG}(\neg \mathsf{b})$ and environment CSP from Example 2. We know that this is unrealizable. The output obtained from our prototype tool is as given in Figure 3. The first phase of construction of the specification automaton and the .spin file containing it is the same as above.

The invocation of BoSy to construct the controller, if it exists, from the specification automaton is also the same. But, the output differs since BoSy



```
anon194@anon194-VirtualBox: ~/csp-synthesis/code/PyAsyncSynth/pycsp
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$ timeout 3 python3 run.py -e ../poplbenchma
rks/example2.csp -s 'false' -l 'FG(b=0)' -o ../poplbenchmarks/example2.sys
['run.py', '-e', '../poplbenchmarks/example2.csp', '-s', 'false', '-l', 'FG(b=0)', '-o', '../poplbenchmarks/ex
ample2.sys']
>> Loading CSP file ../poplbenchmarks/example2.csp <<
>> Done loading CSP file <<

>> Loading LTL specs <<
>> Optimized spin file <<

>> Done loading LTL specs <<

>> Done! <<

exec: swift run --build-path ./.build-sys/ -c release BoSy --backend explicit-one-hot-inputs  --semantics moor
e  --strategy linear  --synthesize --target dot  --statistics  --player system  ../poplbenchmarks/example2.sys
.bosy
info: automaton contains 3 states
info: build encoding for bound 1
info: build encoding for bound 2
info: build encoding for bound 3
info: build encoding for bound 4
info: build encoding for bound 5
info: build encoding for bound 6
info: build encoding for bound 7
info: build encoding for bound 8
info: build encoding for bound 9
info: build encoding for bound 10
info: build encoding for bound 11
info: build encoding for bound 12
info: build encoding for bound 13
info: build encoding for bound 14
info: build encoding for bound 15
info: build encoding for bound 16
info: build encoding for bound 17
info: build encoding for bound 18
info: build encoding for bound 19
info: build encoding for bound 20
info: build encoding for bound 21
info: build encoding for bound 22
info: build encoding for bound 23
info: build encoding for bound 24
info: build encoding for bound 25
info: build encoding for bound 26
info: build encoding for bound 27
info: build encoding for bound 28
info: build encoding for bound 29
info: build encoding for bound 30
anon194@anon194-VirtualBox:~/csp-synthesis/code/PyAsyncSynth/pycsp$
```

Figure 3: Unrealizable sample

is unable to find a solution, and it keeps searching for a solution with larger number of states. In general, BoSy will keep searching for solution with larger number of states, however in this case it terminates at info: build encoding for bound 30 since we executed our initial command line with a timeout of 3 sec .

The same output is generated irrespective of the encoding.

2.4 INSTRUCTION

This file describes and lists-out all the commands that the reviewer must execute to reproduce the results of the case studies present in Section 7.2.1-7.2.3.