[Artifact391] Programmable MCMC with Soundly Composed Guide Programs

Long Pham, Di Wang, Feras Saad, Jan Hoffmann

July 1, 2024

1 What is Included

From Zenodo, you should obtain the following items:

- This `README.pdf` document
- Archived and compressed Docker image `guide_types.tar.gz`

2 Getting Started Guide

Our artifact is a program analysis tool with the following capabilities:

1. Check structural type equality of (possibly context-free) guide types
2. Infer guide types of model and guide programs (using the structural-type-equality checking algorithm)
3. Check whether the support of sequentially composed guide programs coincides with the support of a model program.

In probabilistic programming with our new coroutine-based programmable inference framework, the user provides a model program and a sequential composition of guide programs. The model program specifies a probabilistic model for Bayesian inference. The sequential composition of guide programs customizes the Block Metropolis-Hastings (BMH) algorithm, where we successively run the guide programs, each of which is followed by an MH acceptance routine. Each guide program only updates a subset (i.e., block) of random variables. The model and guide programs are coroutines that communicate with other by message passing, and their communication protocols are described by guide types. To algorithmically decide structural type equality of guide types, our artifact implements the bisimilarity-checking algorithm by Hirshfeld and Moller [1] for context-free processes with finite norms, which model guide types. This structural-type-equality checking algorithm is also incorporated into the type-inference algorithm for guide types. Finally, to verify the soundness of the BMH algorithm, our artifact checks whether the support of the sequentially composed guide programs covers all traces all possible traces admitted by the model program.

The artifact is wrapped inside the accompanying Docker image `guide_types.tar.gz`. Before running it, first install Docker as instructed here: https://docs.docker.com/engine/install/. To see if Docker has been installed properly, run

```
$ docker --version
Docker version 27.0.3, build 7d4bcd8
```

Load a Docker image by running

```
$ docker load --input guide_types.tar.gz
```

It creates an image named `guide_types` and stores it locally on your Docker may create an image with a slightly different name from `guide_types`. To check the name of the image, display all Docker images on your local machine by running

```
$ docker images
```

To run the image `guide_types`, run

```
$ docker run --name guide_types -it --rm guide_types
root@5b1c8c873064:/home/GuideTypes#
```

It creates a Docker container (i.e., a runnable instance of the Docker image) named `guide_types` and starts a shell inside the container. If the command does not run properly, you can instead build the image locally on your machine as instructed in §4.2.

Throughout this document, any command line starting with `#` is executed inside the Docker container, and any command line starting with `$` is executed in your local machine's terminal.

2.1 Structural-Type-Equality Checking

The initial working directory `type-equality` contains a file `type-equality-sample` storing several guide-type definitions. To display the content of the file, we run

```
# cat type-equality-sample
```

and it displays

```
type T1 = &{ $ | real /\ T1[T1] }
type T2 = &{ $ | real /\ T2[T2] }
type T3 = &{ $ | real /\ real /\ T3[T3] }
```

It defines three type definitions:

$$T_1 \coloneqq \&\{\$ \mid \text{real} \wedge T_1[T_1]\}$$

To algorithmically decide the structural type equality between the types `T1` and `T2`, run

```
# dune exec gtypes type-equality type-equality-sample T1 T2
```

3 Step-by-Step Instructions

3.1 Benchmark Suite

The artifact contains 10 benchmark programs for Hybrid AARA as listed in Section 7 of the paper. These benchmarks are: `MapAppend`, `Concat`, `InsertionSort2`, `QuickSort`, `QuickSelect`, `MedianOfMedians`, `ZAlgorithm`, `BubbleSort`, `Round`, and `EvenOddTail`. Inside the artifact, the benchmark `EvenOddTail` goes by its full name, `EvenSplitOddTail`, instead of its shorthand `EvenOddTail` used in the paper.

The paper evaluates both purely data-driven resource analysis and hybrid resource analysis on the first seven benchmarks, and only data-driven resource analysis on the remaining three benchmarks.

Inside the benchmark suite directory `/home/hybrid_aara/benchmark_suite`, each benchmark has its own directory, which contains the following three subdirectories:

1. `utility`: Python source files that contain (i) benchmark-specific configurations for Hybrid AARA and (ii) functions for analyzing the inference results.
2. `bin`: input and output files for Hybrid RaML.
3. `images`: plots of inferred cost bounds.

3.2 Display Experimental Results

In this section, we demonstrate how to produce tables and plots in the paper from the experimental results that are stored in each benchmark's `bin` directory. In the Docker container, the `bin` directory of each benchmark already contains the experimental results that we reported in the paper. We will demonstrate how to generate the experimental results from scratch in §3.3.

Go to the directory `/home/hybrid_aara/benchmark_suite/toolbox`. Then run

```
# python3 run_analysis.py <table-or-plot-type>
```

where `<table-or-plot-type>` specifies the type of tables or plots to produce.

Soundness proportions In Table 1 in the paper, the 4th and 5th columns display the percentages of sound cost bounds inferred by data-driven and hybrid resource analyses, respectively. To display these two columns, run

```
# python3 run_analysis.py soundness
```

This table supports the two key claims in our paper (Section 7). Firstly, in both data-driven and hybrid resource analysis, BayesWC and BayesPC have higher percentages of sound cost bounds than Opt. Secondly, hybrid resource analysis (Hybrid BayesWC and Hybrid BayesPC) has higher percentages of sound cost bounds than purely data-driven resource analysis (Data-Driven BayesWC and Data-Driven BayesPC).

Analysis time The 6th and 7th columns of Table 1 display the analysis time of data-driven hybrid resource analyses, respectively. To display these two columns, run

```
# python3 run_analysis.py time
```

The table of analysis time produced by the artifact looks different from the one in the submitted version of the paper, in terms of both the numbers and layout. This is because, since the submission, we have changed the hyperparameters of Bayesian inference and rerun experiments. In the camera-ready version of the paper, we will update the analysis time in Table 1.

Relative errors Figure 5 in the paper displays relative errors of 5 selected benchmarks. To produce this plot, run

```
# python3 run_analysis.py plot_relative_errors
```

This command produces a plot and stores it as `relative_errors.pdf` inside the directory `/home/hybrid_aara/benchmark_suite/images`. To view the plot, we first transfer it from the Docker container to your local filesystem. Run the following command in your local machine's terminal:

```
$ docker cp hybrid_aara:/home/hybrid_aara/benchmark_suite\
/images/relative_errors.pdf \
<path-to-local-directory>
```

where the second argument `<path-to-local-directory>` is a desired destination directory on your local filesystem. You can then view the plot on your local machine.

For Tables 2–11 in the paper, to create and display them on the stdout, run

```
# python3 run_analysis.py relative_errors
```

Plots of inferred cost bounds To produce Figures 6–24 in the paper, run

```
# python3 run_analysis.py plot
```

This command creates all plots of inference results and stores them in the directory `benchmark_suite/<benchmark-name>/images`. It should terminate in 4 minutes.

For example, for Hybrid BayesWC performed on the benchmark `QuickSort`, its plot is the file `inferred_cost_bound.pdf` in the directory

```
benchmark_suite/QuickSort/images/hybrid/bayeswc
```

To view the plot, we transfer it from the Docker container to your local filesystem. Run this command in the local machine's terminal:

```
$ docker cp hybrid_aara:/home/hybrid_aara/benchmark_suite\
/QuickSort/images/hybrid/bayeswc/inferred_cost_bound.pdf \
<path-to-local-directory>
```

You can then view the plot on your local machine.

3.3 Generate Experimental Results

This section demonstrates how to perform Hybrid AARA on the 10 benchmark programs. First go to the directory `/home/hybrid_aara/benchmark_suite/toolbox`. Then run

```
# python3 run_experiment.py <options>
```

where `<options>` specifies the benchmark name and possibly also the analysis mode. The inference result will be stored inside each benchmarks' `bin` directory.

For example, to perform Data-Driven Opt on the benchmark `QuickSort`, run

```
# # python3 run_experiment.py QuickSort data_driven opt
```

The analysis should finish in 15 seconds. To perform all data-driven resource analyses on the benchmark `QuickSort`, run

```
# python3 run_experiment.py QuickSort data_driven
```

It runs Data-Driven Opt, Data-Driven BayesWC, and Data-Driven BayesPC. The analysis should finish within 2 minutes. If we instead want to perform all hybrid resource analyses, replace `data_driven` with `hybrid`:

```
# python3 run_experiment.py QuickSort hybrid
```

It runs Hybrid Opt, Hybrid BayesWC, and Hybrid BayesPC. The analysis should finish in 11 minutes. To run both purely data-driven and hybrid resource analyses on the benchmark `QuickSort`, run

```
# python3 run_experiment.py QuickSort
```

without specifying the analysis mode.

Finally, to run all 10 benchmarks (under both purely data-driven and hybrid analyses), run

```
# python3 run_experiment.py all
```

This can take up to 2 hours to run[1].

To view the updated tables and plots of the new experimental results, follow the steps in §3.2. The experimental results for (Data-Driven and Hybrid) BayesWC and BayesPC after rerunning the experiments may look different from the tables and plots reported in the updated version of the paper. This is because BayesWC and BayesPC run sampling-based Bayesian inference algorithms that are stochastic. Nonetheless, there should not be any significant deviation from the results reported in the paper.

4 Build and Customize the Artifact

This section describes how to build and customize Hybrid RaML.

[1]We have tried Python's multiprocessing to run multiple benchmarks in parallel, but it did not work properly in Docker. So our artifact can only run benchmarks sequentially.

4.1 Source Code

Our artifact contains three tools that we have built by ourselves: (i) Hybrid RaML (in OCaml), (ii) the volesti-RaML interface (in C++), and (iii) the execution and analysis of benchmark suite (in Python). Their source code is stored in directories `raml`, `volesti_raml_interface`, and `benchmark_suite` inside the project directory `/home/hybrid_aara` in the Docker image.

The project directory contains two additional directories that come from other sources. The third-party C++ library volesti[2] is stored in the directory `volesti`. The linear-program solver COIN-OR CLP[3] is stored in the directory `clp`.

4.2 Build the Docker Image

The code for building a Docker image is available on GitHub: https://github.com/LongPham7/hybrid_aara_artifact. To build a Docker image, clone the GitHub repository and then run (in the root directory)

```
$ docker build -t hybrid_aara .
```

We need a period at the end of the command to indciate that `Dockerfile` exists in the current working directory. The build will take 20–30 minutes. The resulting image is named `hybrid_aara` and is stored locally on your machine.

To run the image `hybrid_aara`, run

```
$ docker run --name hybrid_aara -it --rm hybrid_aara
```

It creates and runs a Docker container with the same name `hybrid_aara`. If you want to save the image as a tar archive and compress it, run

```
$ docker save hybrid_aara | gzip > hybrid_aara.tar.gz
```

4.3 Test Custom Input OCaml Programs

Suppose we want to analyze a custom OCaml function f using Hybrid AARA. We first prepare two files: (i) an OCaml source file and (ii) a JSON configuration file.

The OCaml source file stores (i) the OCaml code of the function f (and possibly other auxiliary functions) and (ii) the OCaml code to be executed in order to collect runtime cost samples of the function f. For example, in the file `append.ml` described in **??**, the OCaml code used for generating runtime cost data has the form

```
let input_dataset = [([31; 11; 26; 25], [10; 36; 20; 24]); ...] in
map input_dataset (fun (x, y) -> append x y)
```

[2]Available here: https://github.com/GeomScale/volesti.

[3]Available here: https://github.com/coin-or/Clp.

This code first creates a list of inputs (i.e. pairs of integer lists) to the function `append` and then runs the function on every input. During its execution, whenever we encounter an annotated code fragment `Raml.stat(...)`, we record its runtime sample, namely the input, output, and cost. We aggregate all runtime cost samples into a dataset, which will later be used in data-driven resource analysis.

For convenience, our artifact comes with a Python script for generating a collection of inputs for OCaml programs. The file `input_data_generation.py` inside the directory `/home/hybrid_aara/benchmark_suite/toolbox` contains a function `create_exponential_lists`, which creates a collection of integer lists whose sizes grow exponentially (e.g., 1, 2, 4, 8, etc.). The content of each integer list is chosen randomly. The function `convert_list_python_to_ocaml` then converts integer lists from the Python format to the OCaml format. To see an example, in the directory `/home/hybrid_aara/benchmark_suite/toolbox`, run

```
# python3 input_data_generation.py
```

It prints out a collection of lists whose sizes grow exponentially.

Finally, the JSON configuration file specifies hyperparameters of Hybrid AARA. To learn its syntax, you can take a look at the file `config.json` inside each benchmark's `bin` directory.

4.4 Modify and Compile Source Code inside the Docker Container

To modify files inside the Docker container while it runs, we can use the text editor Vim. If you wish to use a different text editor, modify `Dockerfile` in the GitHub repository and rebuild the artifact as instructed in §4.1. Alternatively, while the Docker container runs, you can install the text editor by

```
# apt update && apt install <text-editor> -y
```

To recompile code *inside* the Docker container after the code has been modified, follow these steps. For the OCaml source code of Hybrid RaML, go to the directory `/home/hybrid_aara/raml` and then run

```
# eval $(opam env)
# make
```

The first line, which initializes OCaml-related environment variables, is only necessary the first time you recompile Hybrid RaML's source code.

To compile the C++ source code of the volesti-RaML interface, go to the directory `/home/hybrid_aara/volesti_raml_interface/build` and then run

```
# cmake .. && cmake --build .
```

All the other code in the artifact, such as Python scripts inside the benchmark suite, does not require recompilation after modification.

References

[1] Y. Hirshfeld and F. Moller. A Fast Algorithm for Deciding Bisimilarity of Normed Context-Free Processes. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory*, Lecture Notes in Computer Science, pages 48–63, Berlin, Heidelberg, 1994. Springer. ISBN 978-3-540-48654-1. doi: 10.1007/978-3-540-48654-1_5. 1