[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 **??**.

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
```

It prints out

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

The first line defines a guide-type operator $T_1[\cdot]$ as

$$T_1[X] \coloneqq X \mathbin{\&} (\mathbb{R} \wedge \mathbb{R} \wedge T_1[T_1[X]]),$$

where X is a type variable that stands for a continuation type (i.e., the type of the communication protocol that we run after T_1 is finished). The type T_1 means the coroutine receives a branch selection (from another coroutine that it communicates with), and proceeds to either X (i.e., the continuation guide type) or $\mathbb{R} \wedge T_1[T_1[X]]$, depending on which branch is taken. The guide type $\mathbb{R} \wedge T_1[T_1[X]]$ means the coroutine first sends a message of type $\mathbb{R}$ (i.e., real numbers) and then proceeds to the guide type $T_1[T_1[X]]$. Here, $T_1[T_1[X]]$ can be interpreted as a sequential composition of two instances of T_1, followed by the original continuation X.

To algorithmically decide the structural type equality between the guide-type operators $T_1[X]$ and $T_2[X]$, run

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

The command successfulyl determines that the two type operators are structurally equal, as indicated by the following line towards the end of the command's printout:

```
Types T1 and T2 are equal
```

This is as expected because the two type operators $T_1[X]$ and $T_2[X]$ represent the same type—they only differ in the names of type operators.

Next, to check the structural type equality between $T_1[X]$ and $T_3[X]$, run

```
# dune exec gtypes type-equality type-equality-sample T1 T3
```

It displays towards the end of the printout that

```
Types T1 and T3 are unequal
```

which is a correct result because the right branch of $T_1[X]$ only sends one message before possibel termination, while the right branch of $T_3[X]$ sends two messages before possible termination.

2.2 Guide-Type Inference

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