Partial evaluation via code generation for static stochastic reaction network models (Software Appendix)

DOI https://doi.org/10.18453/rosdok_id00002648

Authorship: Till Köster (till.koester@uni-rostock.de), University of Rostock, Institute for Visual and Analytic Computing (2020)

Abstract Succinct, declarative, and domain-specific modeling languages have many advantages when creating simulation models. However, it is often challenging to efficiently execute models defined in such languages. We use code generation for model-specific simulators. Code generation has been successfully applied for high-performance algorithms in many application domains. By generating tailored simulators for specific simulation models defined in a domain-specific language, we get the best of both worlds: a succinct, declarative and formal presentation of the model and an efficient execution. We illustrate this based on a simple domain-specific language for biochemical reaction networks as well as on the network representation of the established BioNetGen language. We implement two approaches adopting the same simulation algorithms: one generic simulator that parses models at runtime and one generator that produces a simulator specialized to a given model based on partial evaluation and code generation. Akin to profile-guided optimization we also use dynamic execution of the model to further optimize the simulators. The performance of the approaches is carefully benchmarked using representative models of small to mid-sized biochemical reaction networks. The generic simulator achieves a performance similar to state of the art simulators in the domain, whereas the specialized simulator outperforms established simulation algorithms with a speedup of more than an order of magnitude.

This repository contains the code generation software as described in the 2020 PADS paper *Partial evaluation via code generation for static stochastic reaction network models.*

The scripts are designed to run on a linux machine.

Usage for simulation

If you want to run a simulation on a model (some are provided in the models folder) use the `runSimulation.py` python script provided.

To run the provided multistate model run

```
python3 runSimulation.py models/multistate.net \
    --optimize --steps=1000 --until=10 --target=rust
```

You can set the number of observations (steps) and the simulation time (until). Target specifies what simulator to use. There are the following options

Target	Description
rust	Generates specialized simulator in Rust (default)
gcc	Generates specialized simulation in C++, compiled with GCC
clang	Generates specialized simulation in C++, compiled with Clang
generic	Uses the generic simulator, written in Rust
bionetgen	Uses the run network capabilities build into bionetgen

When called for the first time, there is some overhead for some of the simulators, as there is some one-time compilation cost for used libraries. However once compiled they can be reused.

The `--optimized` flag makes a pilot run using the generic simulator, to create an optimized binary.

The output format of all simulation targets is the same. They produce a text file containing columns with the values. Time is in the first column.

Technical Requirements

Rust

The rust programming environment (rustc, cargo, etc) is best installed via rustup available at rustup.rs. Rust is also needed for the generic simulator.

Python3

Python is required, for code generation.

C++ Compilers

C++ compilers are needed to compile some of the artefacts. For the paper we used GCC and clang, both of which are available in the standard repos of linux distributions.

Bionetgen

BioNetGen is an existing software tool we compare our approach against. The tool will try and download bionetgen 2.5 into the folder BioNetGen-2.5.0, via `wget`. If you would like to provide another path or installation, please modify the function `get_bionetgen_path()` in the `runSimulation.py` accordingly.

Directory Structure

Models

The models that were used (as well as others) can be found in the models folder. The `.net` files are in the format as generated by BioNetGen. The `.reaction_model` files are in the reaction DSL as defined in the paper.

Generic Simulator

The generic Simulator is implemented in Rust and may be found in the generic simulator folder. It has the option `--count`, that will create the `reaction_counts.toml` file containg how often each reaction was fired.

Generated Code

This folder contains all the generated code. If you need it, the binary for the generated rust simulator can be found in

```
generated_code/target/release/generic_simulator
```

Original Publication

This software artifact relates to the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation publication *Partial evaluation via code generation for static stochastic reaction network models.* It was authored by Till Köster at the Institute for Visual and Analytic Computing. Funding was provided by the Deutsche Forschungsgemeinschaft (DFG) research grant 'ESCeMMo' (UH-66/13).