Artifacts for Scaling Up Livelock Verification for Network-on-Chip Routing Algorithms

By Landon Taylor and Zhen Zhang, Utah State University.

Abstract

This artifact contains the models described in the paper “Scaling Up Livelock Verification for Network-on-Chip Routing Algorithms” along with a way to reproduce many of the results from the paper. The models presented in this abstract demonstrate principles from Sections 5-10 of the paper.

Downloadable Virtual Machine

The virtual machine that contains the artifact can be found at `https://doi.org/10.5281/zenodo.5512664`. The artifact’s DOI is `10.5281/zenodo.5512664`. An alternative download link is provided at `https://drive.google.com/file/d/1xlrBJPzDYrKEkeGIHEVY3nH1aMa5rLco/view?usp=sharing`

Checksum

The SHA256 Checksum of the archive file is `ff96202a8c0c310f1487c1aa9eb9a887bd634b15fe7b0dc2022655aa8c10d013`

Virtual Machine Instructions

This information is included in the virtual machine archive as well.

Operating System: Ubuntu 18
RAM: 4GB
Username: `user`
Password: `user`

Tested in Virtualbox on Windows 10 and Mac OSX. The lowest system specifications used to test this virtual machine are a *Win□dows 10 machine (version 1903) with an Intel Core i7 4-Core 2 GHz Processor and 8 GB memory.*

Step-By-Step Loading Instructions

1. Download and import IVy-Models.ova into Virtualbox. Start the VM.
2. When prompted, the username is `user` and the password is `user`.
3. Two folders are present on the desktop. The folder called `Quick-Models` contains models mentioned in the paper and documented to provide reproducible results. The folder called `IVy-Models` is provided for reference only. It contains the full files and results from our paper, and is not as user-friendly as `Quick-Models`. Open the folder `Quick-Models` in the file explorer or terminal.

4. Because the screen size of the virtual machine is small, we recommend that you load the online repository alongside it or on a separate machine. This will provide access to the README files in a more readable and accessible way. This repository is available at https://github.com/formal-verification-research/IVy-Models/tree/master/Quick-Models and contains identical files and README documents to the virtual machine.
5. Each subfolder of `Quick-Models` includes a README file specific to the model found in that folder. The information from these README files is also available below. Each subfolder contains a model described in the paper.

Model Execution Instructions

Once in the `Quick-Models` folder, the following subsections accompany their identically-named subfolders.

Refutation-Based Simulation

Every reference to a directory will be from the perspective of the `Quick-Models/simulation` subfolder (i.e. directory `a` refers to the directory `~/Desktop/Quick-Models/simulation/a`).

This folder contains a sample of the C++ simulation model and IVy livelock verification model used for simulation and livelock verification, respectively. The work from this subfolder is described in Section 5 of the paper.

This process will generally take under 30 minutes but may take more depending on available CPU

To Reproduce Simulation Results

1. To initiate the C++ simulation on Algorithm 1 on a 3×3 NoC, navigate to the `3x3` directory and use the `make` command. The C++ simulation should run in a matter of seconds. Simulation detects potential livelock scenarios, and their traces are found in the `3x3/livelocktrace` folder. Several `.txt` files are generated to report the simulation's findings. To see the number of potential livelocks when 2 faults are present in the network, open the file `3x3/_2_report.txt`. It reports that 9 potential livelocks were found during simulation.
2. To prove that a potential livelock scenarios are indeed livelock, navigate to the `3x3/livelocks` directory and execute `make`. This will examine the traces returned in the `3x3/livelocktrace` folder and populate the `3x3/livelocks/ivyfiles` directory with IVy models. These models implement the livelock verification description from Section 5.2 in the paper. As indicated in `3x3/_2_report.txt`, 9 potential livelocks are detected, so 9 IVy files are generated. Each of these IVy models (for instance, `2s00_d22_f11w_f21n.ivy`) represents a single potential livelock. The file name above indicates a 2-fault NoC with the packet starting at coordinates (0,0), destination (2,2), with faulty links at (1,1,west) and (2,1,north). The invariants described at the end of these files are the implementations of equations 1-4 with values from a specific potential livelock. Finally, it will use a script to use IVy to check each model.
3. The `3x3/livelocks/ivyfiles/tests` directory contains the results of IVy's verification. Open any file in this directory and find the text `OK` at the end. This indicates that IVy was able to

successfully verify that the potential livelock is indeed a true livelock trace, and that a packet is unable to escape its cyclical pattern (as described in Section 5 of the paper).
4. To analyze and group the livelock patterns we verified in Step 3, navigate to the `pattern_finder` folder (in the `simulation` directory) and execute `make`. A script will prompt for a folder to crawl. Input `3x3` and click enter. It will quickly examine each livelock trace and group them into patterns. Once it is finished, type `quit` to terminate the script and navigate to the `pattern_finder/test` folder. Open the file `3x3_patterns.txt` to view the grouping of livelock patterns from Algorithm 1. The user is able to identify the decision that initiates the greatest number of livelock scenarios, as shown in Figures 2-3 and the preceding paragraph in Section 6 of the paper.
5. Note also that unroutable traces, as described in Section 7 of the paper, are found in the `3x3/cannotroutetrace` folder. These traces aid the user to understand the impact of adding livelock resistance to a network.

Zone Abstraction

Every reference to a directory will be from the perspective of the `Quick-Models/zone_abstraction` subfolder (i.e. directory `a` refers to the directory `~/Desktop/Quick-Models/zone_abstraction/a`).

This folder contains the model used to produce abstract zones with the help of IVy. The work from this subfolder is described in Section 8 of the paper.

This process will generally take under 30 minutes but may take more depending on available CPU

To Reproduce Results

1. Navigate to the `5x5` directory and run the command `ivy_check 5x5.ivy > test.txt`. This model is an implementation of Figure 4(c) and its description in Section 8 of the paper.
2. After waiting for a moment, a lengthy trace will appear in the file `5x5/test.txt` ending with the keyword `FAIL`. In the case of this model, that indicates that two zones satisfy identical conditions. To discover which zones are identical, look for lines which read `zone.ok(j) = true` and `zone.ok(k) = true`. This indicates that zones J and K in Figure 4(c) can be combined into a single zone. This allows us to modify the model and continue to verify until the zones are all unique. The files `5x5/5x5_1.ivy` through `5x5/5x5_7.ivy` demonstrate our process of combining zones. These files show the intermediate steps between Figure 4(c) and Figure 4(d) in the paper.
3. When all the zones are unique and the conditions from Section 8 of the paper are all satisfied, IVy will verify it. In the `5x5` directory, execute the command `ivy_check 5x5_7.ivy > final.txt`. The trace in `final.txt` indicates a `PASS` at the end of the file. This means that all zones are unique. This is a verification of the zone model implemented in Figure 4(d) of the paper.

Zone Verification

Every reference to a directory will be from the perspective of the `Quick-Models/zone_verification` subfolder (i.e. directory `a` refers to the directory `~/Desktop/Quick-Models/zone_verification/a`).

This folder contains the model used to produce abstract zones with the help of IVy. The work from this subfolder is described in Sections 9-10 of the paper.

This process will generally take under 30 minutes but may take more depending on available CPU

To Reproduce Results

1. Execute `make`. When prompted for the name of a file, enter `minimal_copy`, then enter `n` when asked if you'd like dropped flit testing. This `minimal_copy` model is a variation of Algorithm 1 and produces several livelock scenarios. As IVy detects livelock scenarios, they will appear on the terminal. IVy will provide a list of variables at the time livelock was detected. What follows the words `found livelock:` is the addition to the invariant used to detect additional livelock scenarios. It is, in effect, a livelock trace. That is, what follows `found livelock:` is σ_1 as explained in Section 10 of the paper (on page 17).
2. Allow the terminal to run for several executions, then close it to kill the process (otherwise it will discover livelock scenarios for several hours). Examine the file at `minimal_copy_n_tests_<timestamp>/0.ivy`. This is the model that IVy first verified after executing the script. Now examine the file at `minimal_copy_n_tests_<timestamp>/1.ivy`. The livelock trace discovered in Step 1 is appended to the invariant at the end of `1.ivy` just as σ_1 is appended to σ_0 in Section 10 of the paper (on page 17).

Potential Error Correction

Minimum RAM Requirement

The virtual machine is allocated 4GB of RAM. It has been tested on a Windows 10 computer with 8GB of RAM.

IVy Test Errors

If in the *Refutation-Based Simulation* section, the `ivyfiles` folder is populated with files containing errors, a likely cause is duplication within the IVy files. That is, generation of new IVy files appends to existing files (instead of overwriting them). If the script ran twice, the IVy checks will return errors the second time. A solution to this issue is deleting each file in the `ivyfiles` and `ivyfiles/tests` folders and running `make` from the `livelocks` directory again. This `make` command should be executed only once.