Chronosymbolic Learning

Artifact for the paper "Chronosymbolic Learning: Efficient CHC Solving with Symbolic Reasoning and Inductive Learning"

- See `./experiment` for some significant results and configurations
- See `./examples` for examples of how our tool works

Requirement (To set up our environment)

Python (3.7.0 or higher recommended, and Anaconda recommended to set up a new environment)

- Install packages in `requirements.txt`: `pip install -r requirements.txt`
  - Though not integrated in our artifact, you can also try to use scikit-learn-intelex to speed up CART DT if possible
- May have to manually set up `PYTHONPATH` and `PATH` properly, `PYTHONPATH=$Z3_BIN/python`, `PATH=$PATH:$Z3_BIN`
- If the C5.0/LIBSVM binary cannot be executed properly, may have to recompile them in your OS and specify the binary executable files in `utils/dt/dt.py` in `class C5DT`, `C5_exec_path` and `utils/svm/svm.py` in `class LibSVMLearner`, `svm_exec_path`

Chronosymbolic Learning

- Support SMT-LIB2 format (check-sat) and Datalog format (rule-query)
- Have executable binaries of decision tree and SVM for Linux and MacOS, and can automatically adapt to the OS (Linux/Mac)
- Control flow implemented in `learner/run_agent.py` `run_Agent` function
- Hyperparameters in `config.yml`
- Temp files generated when calling decision tree and SVM are in `tmp/`
- Implemented some optimization for SMTLIB files generated by SeaHorn
- Run: `python test.py` with the parameters below:
  - Specify instance file name using `-f FILE_NAME` to run a single instance
  - Specify the log file (which records how the tool solves the CHC system) using `-l FILE_NAME`
  - Specify directory name using `-r -f DIR_NAME` to run a test suite (logs are automatically generated in log/DIR_NAME, see `experiment/README.md` to better understand the logs)
    - e.g. `python test.py -f tests/safe/ -a -r -v -t 360 -o result/result.log`
  - Or specify a file list using `-b -f FILELIST` (run files specified in the file list whose format is one file path in each line)
    - e.g. `python test.py -a -v -b -f tests/filtered.txt -a -t 360 -o result/result.log`
  - Increase log file verbosity using `-v` (not effective in output on screen)
  - Adjust timeout using `-t TIMEOUT`, only effective in directory mode

    - Specify the result summary log file using `-o FILE_NAME`; Export an additional result summary CSV `FILE_NAME_prefix.csv` (with success and timing statistics, and `is_correct` column indicates the satisfiability of the CHC system *if successfully solved* (`is_successful=1`); if `is_successful=0`, `is_correct` field is not meaningful) using `-a`; The summary is only available when running multiple instances (directory mode or file list mode)
    - Start solving from the file index `K` in the folder `-s K` (`K` is the index starting from zero)
    - If you want to run multiple instances, make sure to use different `FILE_NAME`-s in the config file to avoid clash (`config.yml` in default)
    - More options see `--help`
- After finishing running, the `./tmp` directory can be deleted safely

To reproduce the major result: Chronosymbolic-single

Please refer to the configuration in `./experiment/result_summary.log` and `./experiment/README.md` (where settings for other minor experiments are also provided). Using the default config in `config.yml` should also be decent. Even fixed random seeds can cause minor randomness that may slightly affect the performance.

- `python test.py -f tests/safe -a -r -v -t 360 -o result/result_safe.log`
- `python test.py -f tests/unsafe -a -r -v -t 360 -o result/result_unsafe.log`

To run the baselines

Spacer and GSpacer

- Configure z3-gspacer-branch, `chmod +x z3` (pre-built binary of z3 with Spacer and GSpacer for Ubuntu)
- Specify the path of pre-built z3 (with Spacer and GSpacer) binary in `utils/run_spacer.py` and `utils/run_spacer_filtered.py`, at line 5
- Specify a target folder in `utils/run_spacer.py` or specify a file list in `utils/run_spacer_filtered.py`, at line 4
- Use GSpacer as the solving engine: `enable_global_guidance = 1`; Use Spacer as the solving engine: `enable_global_guidance = 0`, at line 8
- Check `utils/run_spacer.py` and `utils/run_spacer_filtered.py` line 4-20 for other settings
- After configuration, run `python utils/run_spacer.py`

LinearArbitrary and FreqHorn

Refer to LinearArbitrary and FreqHorn.

A prebuilt docker image is available on Docker Hub.

The pre-built binary for FreqHorn is also provided here: freqhorn and expl.

For LinearArbitrary, you can also try our optimized data-driven learner implementation (set `ClassAgent = Chronosymbolic` to `ClassAgent = DataDrivenLearner` in `test.py` and run it in the same way as Chronosymbolic does)

Manually "guess" an inductive invariant (hard to scale up)

In `test.py` `guess_manually` function:

- Modify `s = 'v_0 == v_1'` to indicate the inductive invariant
- Modify `db = load_horn_db_from_file(args.file_name, z3.main_ctx())` or pass the parameter in through command line to indicate SMTLIB2 file name

Benchmarks

CHC-COMP

FreqHorn

LinearArbitrary

Reference

chc-tools

libsvm

ICE-C5

LinearArbitrary

z3

SeaHorn