CS-472: Design Technologies for Integrated Systems

Final Project: Option 3 Due date: 19/12/2020, 23:59

Extended Binary Decision Diagram Package

1 Introduction

A very simple ROBDD package was implemented in a previous programming assignment, but many common features [1, 2] were missing, such that the package is very time- and memory-inefficient. The goal of this project is to extend and improve this package.

1.1 Operation Cache

During the recursive BDD operations, some computations may be repeated several times. An operation cache, or often called the *computed table*, is a hash table mapping from the operands and the operation type to the computed result.

1.2 Complemented Edges

To reduce memory usage, complement Boolean functions can be represented with only one node and two edges, a normal one and a complemented one, pointing to it. In order to remain canonical, we avoid complementing the THEN edge, but complement the ELSE edge and the incoming edge instead.

1.3 Node Reference Count

In the naive implementation, there are often some nodes created during the recursive computations but discarded afterwards. In other words, the total number of nodes existing in the BDD package is more than the number of nodes reachable from the root node. Hence, a smarter way to reduce memory usage is keeping a *reference count* of each node, which is the number of edges pointing to the node. When a node has a reference count of zero, we call it *dead* and it can be deleted when we are out of memory.

2 Project Tasks

First, extend the provided truth table data structure to support functions of more than 6 variables. It may be helpful to reference to the implementation in the truth table library *kitty* [3].

Then, extend our simple BDD package to include the following features:

1. Operation cache (computed table).
2. Complemented edges.
3. Node reference count tracing.
4. (Optional) Garbage collection (deleting dead nodes and reusing the memory they occupy).
5. (Optional) variable reordering [4].

There exist several open-source BDD packages with all these techniques implemented, such as *CUDD* [5] and *BuDDy* [6], which could be useful references.

Note on the complemented edges: In the simple implementation, we simply use `index_t`, which is referring to a node, to represent Boolean functions. But now with the complemented edges, a Boolean function is represented by an *edge*, which can be complemented, but not a *node* anymore. Hence, you may want to wrap `index_t` with another data structure to store the complementation information.

3 Submission and Grading

To start with, fork and clone the project repository from GitHub:

1. Create a GitHub account if you do not have one yet.
2. Fork the *BDD* repository from: https://github.com/EPFL-CS-472/BDD to your account.
3. Clone your fork to your computer (from your account, so that you have the permission to push to it). Move into the cloned directory.
4. (Optional) Create a new working branch and switch to the branch.

There is also a demonstration video on Moodle for git commands.

Simple tests from the previous assignment are in `simple.cpp` and they should be all passing already. It can be compiled with `make simple` and run with `./bdd_simple`. The main tests are defined in `main.cpp`, which will only pass if the requirements are correctly implemented. It can be compiled with `make` and run with `./bdd`. Feel free to define more to test your code thoroughly (but do NOT commit them). These tests will be run automatically by the GitHub Actions when you push a commit or make a pull request. The grading will be based on passing all these tests. More tests may be released later (one week before deadline at the latest).

To submit the final project, make a pull request to `EPFL-CS-472/BDD`. Write a short report of **no more than 2 pages** and submit to Moodle. The report should include: the project choice (extended BDD package), ID of your pull request (e.g., #17), and brief description of your algorithm/implementation. You should not make further commits to the branch where you create the pull request after the deadline. (You can create another branch if you still want to play around with the project after the deadline.)

References

[1] Brace, K. S., Rudell, R. L., & Bryant, R. E. (1990). Efficient implementation of a BDD package. In *27th ACM/IEEE design automation conference* (pp. 40-45). IEEE.

[2] Minato, S. I., Ishiura, N., & Yajima, S. (1990). Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In *27th ACM/IEEE Design Automation Conference* (pp. 52-57). IEEE.

[3] *kitty*: truth table library. Source: https://github.com/msoeken/kitty
Documentation: https://libkitty.readthedocs.io/en/latest/?badge=latest

[4] Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)* (pp. 42-47). IEEE.

[5] Somenzi, F. (2009). CUDD: CU decision diagram package-release 2.4.0. *University of Colorado at Boulder*.

[6] BuDDy: binary decision diagram package. http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.itu.dk/research/buddy/