Circuit realization of truth tables and Boolean functions in Agda

Chih-Hsiang, Chuang
Institute of Information Science, Academia Sinica
ChuangChihHsiang@gmail.com

Abstract

Logic synthesis is a process that transfers abstract functionality into circuits realized by a series of elementary gates. We propose a method to look at logic synthesis in Agda, which focuses on an inductive structure of truth tables that is isomorphic to Boolean functions. We analyze its components separately and synthesize various kinds of circuits. We formalize this notion and verify that the transition is indeed well-behaved. This approach provides a potential framework for future development of logic synthesis algorithms and verification that is checkable by computers.

Key Word: logic synthesis, Formal verification

Introduction

We propose an inductive data type $Table\ n$ as our representation for the truth table and the main focus for this research. The data type models the structure of a form of ordered binary decision diagram (BDD) [1] with n input Boolean variables. $Table\ 0$ can be seen as an output entry for the truth table, while $Table\ (suc\ n)$ is a cluster of two subtables of $Table\ n$ that can be treated as a decision branch:

$$\frac{b : Bool}{output\ b : Table\ 0} \qquad \frac{t_0, t_1 : Table\ n}{decision\ t_0\ t_1 : Table\ (suc\ n)}$$

We prove that this structure is indeed isomorphic to Boolean functions with n inputs by making a choice that assigns the first subtable as the false branch and the second subtable as the true branch. This isomorphism allows us to focus on truth tables for the following works in logic synthesis, and we may translate the results back into general Boolean functions. The structure of the truth table allows us to do proofs more easily; often times we can prove a statement on the base case in output using Boolean logic and then inductively prove the statement for any truth tables, which creates a solid ground for later development of our works presented below.

In this work, we mainly consider disjunctive normal form (DNF), which can easily transform into a circuit, as our target of synthesis. The DNFs can be defined as an inductive structure by the following rules in the graph, with the transition from a DNF to a truth table also being defined inductively.

$$\frac{x : Variable}{pos\ x : Literal} \qquad \frac{x : Variable}{neg\ x : Literal} \qquad \frac{}{\emptyset : Conj}$$

$$\frac{c : Conj \quad l : Literal}{c\ \&\ l : Conj} \qquad \frac{}{const\ 0 : DNF} \qquad \frac{d : DNF \quad c : Conj}{d + c : DNF}$$

Our goal is to find algorithms for creating a DNF from a given truth table and verify that the result DNF does transist into the same table as the original one using Agda. The problem reduces to proving that the two tables imply each other, that is, if an output spot of a table has a value true, then the same spot of another table must also be true. Our (naive) approach for DNF is to create a list of conjunction terms that satisfy the table and combine them to form a DNF.

To verify they are indeed the same, we also need a tool to access and separately check every output of a given truth table. For this case, we use another data type (entries n), which is an n-input truth table paired with a vector of boolean values with length n that represent different spots on a truth table. By piecing evidence on every separate entry of a given table together, we can often make a statement about the whole structure of the truth table.

Our work provides a framework for analyzing the structure of a truth table and proving possible statements between truth tables and their circuit realization with inductive proofs. This method can serve as a foundation and be extended for other problems in logic synthesis, some of which are listed in the following section.

Future works

With the framework presented above, we can utilize and extend it to explore further topics we care about regarding logic synthesis; the following are some of the possible examples.

Different outputs for truth table : We only consider the truth table, where every entry is either true or false, but we may also consider a don't care term, that is, an output that can be either 0 or 1. After the transition from the truth table to a circuit, we verify it to be compatible with the original circuit; that is, its truth table should be completely the same with the original truth table except the don't care spot.

Circuit cost analysis : The well doing of a circuit is closely tied to its amounts of elementary gates; since more gates means increasing cost of space and computing delay, we can compute the cost of a circuit given a specific circuit model and compare the performance between different algorithms.

Different circuit models : There are various models of circuits we can take, with one of the most common being the And-inverter graph (AIG). We can establish a connection between AIG and the truth table we construct and study the already existing algorithm for transforming AIG, such as the transduction method [2]. With the transition of AIG to the truth table, we argue that these transformations, while able to reduce the cost of a circuit, still preserve the same logical functionality of the original circuit.

Algorithm complexity analysis : With already existing programs like CALF [3], we may analyze the computation complexity with different approaches to create a circuit and compare their respective performance in terms of the time complexity and the cost of the final results.

References

[1] Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[2] Y. Niu, J. Sterling, H. Grodin, and R. Harper, “A cost-aware logical framework,” *Proc. ACM Program. Lang.*, vol. 6, no. POPL, jan 2022. [Online]. Available: https://doi.org/10.1145/3498670

[3] Y. Miyasaka, “Transduction method for aig minimization,” in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 398–403.