Using Hash Tables to Store BDDs

Néstor Cataño C.
ncatano@puj.edu.co

Preliminaries

Binary Decision Diagrams (Based on "Model Checking" by E.M. Clarke)

Binary Decision Diagrams (BDDs) is a data structure that is frequently used for verification purposes. BDDs are a canonical representation for boolean formulas. They are often more compact than traditional normal forms such as conjunctive normal form and disjunctive normal form, and they can be manipulated very efficiently. To motivate our discussion on BDDs we first consider *binary decision trees*. A binary decision tree is a rooted, directed tree that consists of two types of vertices, terminal vertices and nonterminal vertices. Each nonterminal vertex v is labelled by $var(v)$ and has two successors: $low(v)$ corresponding to the case when the variable v is assigned 0, and $high(v)$ corresponding to the case where v is assigned 1. Each terminal vertex v is labelled by $value(v)$ which is either 0 or 1. A binary decision tree for a two-bit comparator system, given by the boolean formula $f(a, b, c, d) = (a \leftrightarrow c) \wedge (b \leftrightarrow d)$ is shown in Figure 1 (a). One can decide whether a particular truth assignment to the variables makes the formula true or not by traversing the tree from the root to a terminal vertex. If the variable is assigned 0, the the next vertex on the path from the root to the terminal vertex will be $low(v)$. If v is assigned 1 the the next vertex on the path will be $high(v)$. The value that labels the terminal vertex will be the value of the function for this assignment. For example, the assignment $a := 1$, $b := 0$, $c := 1$, $d := 1$ leads from the root to a leaf vertex labelled 0; hence the formula is false for the assignment.

Binary decision trees do not provide a very concise representation for boolean functions, in fact they are the same size as truth tables. For example, in Figure 1 (a) there are eight subtrees with roots labelled by d, but only three of them are distinct. To obtain a concise representation for boolean formulas binary decision trees must be provided with a *canonical representation*. A

Definition 1 (Canonical Binary Decision Diagrams) A canonical binary decision diagram is a binary decision tree with the following properties :

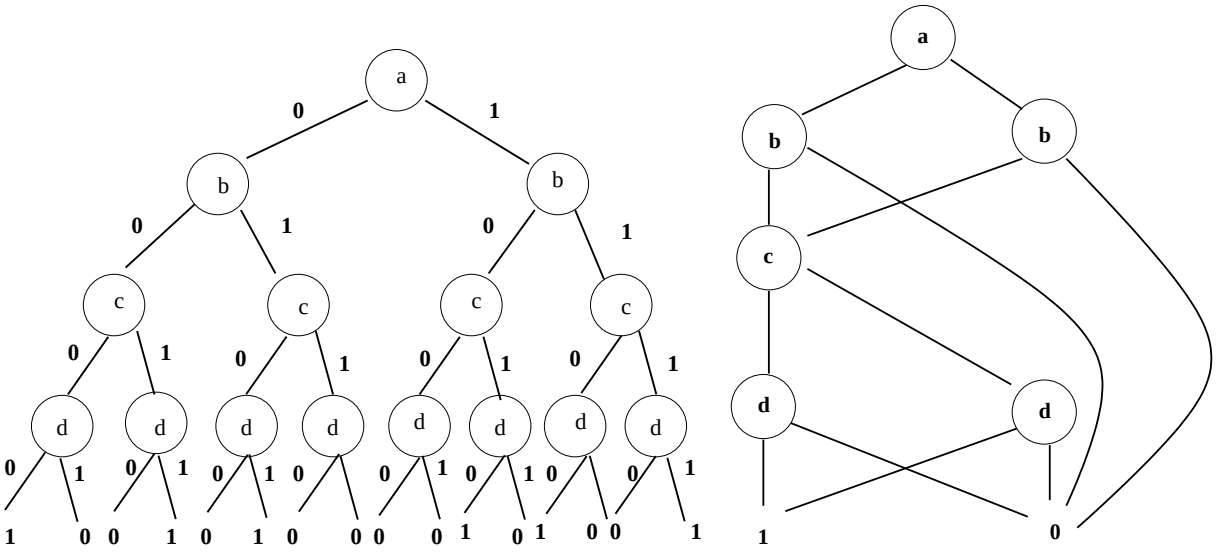
1. *Ordered.* Variables that label the vertices in the binary decision diagram are imposed a total ordering $<$ requiring that for any vertex u in the diagram, if u has a nonterminal successor v then $var(u) < var(v)$.

2. *Reduced.* For any nonterminal vertex u, $low(u)$ leads to a vertex which is different from $high(u)$.

Binary decision diagrams are required to have two terminal vertices only. The first terminal vertex labelled by 0, the second one labelled by 1. Figure 1 (b) shows a canonical representation for the two-bit comparator example in which we have chosen the ordering $a < b < c < d$.

Hash Tables

Hash tables are data structures in which *keys*, for instance a person's name or a BDD node, are mapped into values, for instance a person's social security number. Ideal hash functions are collision free. Two different keys collide if their hash numbers are the same. In practice, however, hash table implementations allow hash functions to collide to a certain extent, and different techniques to deal with collisions are implemented in the hash table.



(a) Binary decision tree for two-bit comparator

(b) Canonical Binary decision tree for two-bit comparator

Figure 1: The two-bit comparator example

Modelling Hash Tables for Storing BDDs in PVS

You shall model a hash table library suitable for storing canonical Binary Decision Diagrams in PVS. To this end you shall :

- formalise Binary Decision Diagrams as a PVS `datatype`.
- use a `table` PVS `theory` to formalise hash tables. The hash table will be collision-free so that a bijective function between keys and values should exist.
- formalise a predicate `ordered?` which is true only if the BDD store at some key is ordered according to Definition 1.
- formalise a predicate `reduced?` which is true only if the BDD stored at some key is reduced according to Definition 1.
- prove all the TCCs generated by your formalisation. You are not allowed to use any of the `(grind)`, `(ground)`, `(subtype-tcc)`, and `(termination-tcc)` PVS commands.