Modelling a Hash Table Library for Storing BDDs in PVS

Néstor Cataño

ncatano@puj.edu.co

Faculty of Engineering

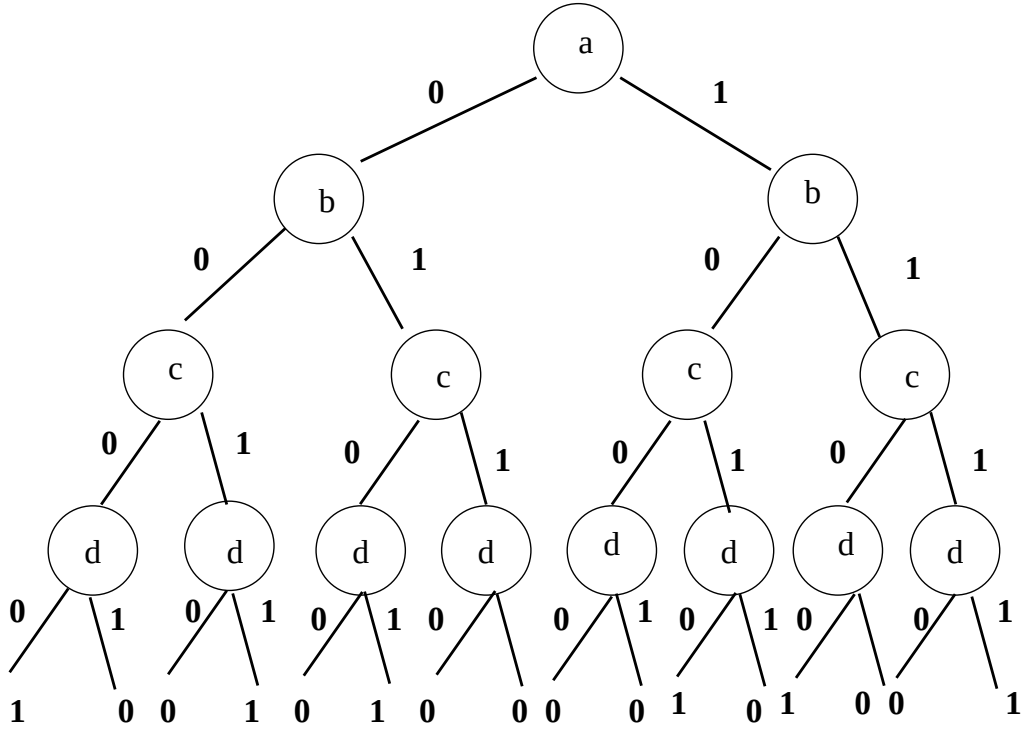
Pontificia Universidad Javeriana

Binary Decision Diagrams (BDDs)

- frequently used in verification for storing boolean formulas
- often more compact than traditional normal forms (e.g., conjunctive normal) when used in canonical form

Binary Decision Trees

Two-bit comparator example



$$f(a, b, c, d) = (a \leftrightarrow c) \wedge (b \leftrightarrow d)$$

Binary Decision Trees

- Each nonterminal vertex v is labelled by $var(v)$ and has two successors: $low(v)$ and $high(v)$
  - $low(v)$ corresponds to the case when the variable v is assigned 0
  - $high(v)$ corresponds to the case when the variable v is assigned 1
- Each terminal vertex v is labelled by $value(v)$ which is either 0 or 1

Binary Decision Trees

- Each nonterminal vertex v is labelled by $var(v)$ and has two successors: $low(v)$ and $high(v)$
  - $low(v)$ corresponds to the case when the variable v is assigned 0
  - $high(v)$ corresponds to the case when the variable v is assigned 1
- Each terminal vertex v is labelled by $value(v)$ which is either 0 or 1

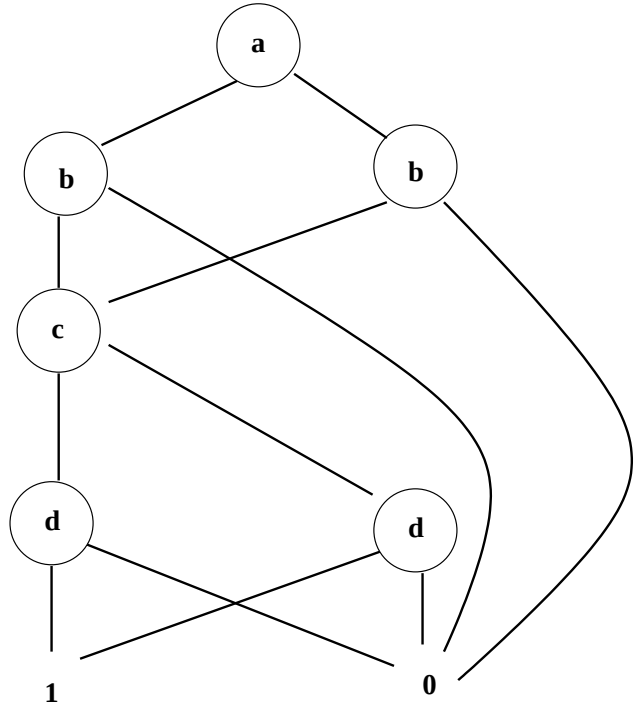
There are 8 subtrees with roots labelled by d, but only 3 of them are distinct

Canonical Binary Decision Diagrams

- 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 $var(u) < var(v)$
- Reduced. For any nonterminal vertex u, $low(u)$ leads to a vertex which is different from $high(u)$.

Canonical Binary Decision Diagrams

Canonical two-bit comparator example



$$f(a, b, c, d) = (a \leftrightarrow c) \wedge (b \leftrightarrow d)$$

Hash Tables

- They are data structures in which keys (e.g a person's name or a BDD node), are mapped into values (e.g a person's social security number)
- Collision-free: keys are uniquely mapped into values (and vice-versa)

A Hash Table Library for Storing BDDs

- formalise BDDs 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
- formalise a predicate `reduced?` which is true only if the BDD stored at some key is reduced
- 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