Quantified Boolean Formulas, and Binary Decision Diagrams

Lecture Scribe: Qiaobin Fu (qiaobinf@bu.edu)

April 27, 2015

1 Quantifier Boolean Formulas

One of the simplest and most widely described methods for coping with QBF is quantifier elimination. Quantifier elimination in a QBF consists in applying the following rules repeatedly:

- Replace every sub-WFF $\exists x.\psi(x)$ by $\psi(0) \lor \psi(1)$
- Replace every sub-WFF $\forall x.\psi(x)$ by $\psi(0) \land \psi(1)$

where “0” stands for $\bot$ (falsity or F) and “1” stands for $\top$ (truth or T). Once all quantifiers are eliminated, satisfiability of the original QBF is reduced to the satisfiability of a propositional WFF. The main drawback of quantifier elimination is the exponential blow-up in the size of the WFF (i.e., $|\phi| = n \sim |\phi'| = O(2^n)$), unless we can combine it with other rules and ways of organizing their representation that will limit the exponential growth.

Efficient ways of organizing the representation of propositional WFFs as well as quantified propositional WFFs are

- Binary Decision Diagram (BDD)
- Ordered Binary Decision Diagram (OBDD)
- Binary Expression Diagram (BED)
- Reduced Boolean Circuits (RBC)
- ...

2 Reachability in Transition System

Consider a transition system $M \triangleq (S, A, I, F)$, where

- $S$ is the set of states
- $A$ is the transition relation
• $I$ is the set of initial states
• $F$ is the set of final states

A typical question in model checking is whether a state in $F$ can be reached from a state in $I$ in $k$ transition steps. Let $R_k(t, t')$ denote that state $t'$ is reachable from state $t$ in exactly $k$ steps. To express that a final state $t' = t_k$ is reachable from an initial state $t = t_0$, we can write

$$R_k(t_0, t_k) \overset{\Delta}{=} \exists t_1 \ldots \exists t_{k-1} I(t_0) \land F(t_k) \land (\forall i=0^{k-1} A(t_i, t_{i+1}))$$

For example, for reachability in exactly $k = 4$ steps, we deal with the WFF

$$R_4(t_0, t_4) \overset{\Delta}{=} \exists t_1 \exists t_2 \exists t_3 I(t_0) \land F(t_4) \land A(t_0, t_1) \land A(t_1, t_2) \land A(t_2, t_3) \land A(t_3, t_4)$$

By eliminating the quantifiers “$\exists t_1 \exists t_2 \exists t_3$”, we need to “unwind” or “unroll”

$$A(t_0, t_1) \land A(t_1, t_2) \land A(t_2, t_3) \land A(t_3, t_4)$$

by considering all possible selections of $\{t_1, t_2, t_3\}$. If $|S| = M$, then there are $\binom{M}{3}$ selections, i.e.,

$$\frac{M(M-1)(M-2)}{3!}$$

a potentially very large number. More generally, $R_k$ uses $k$ conjunctions of the transition relation $A$. An alternative definition of $R_k$ is written as follows:

$$R_k(t_0, t_k) \overset{\Delta}{=} \exists t_1 \ldots \exists t_{k-1} I(t_0) \land F(t_k) \land \forall u \forall v (\bigwedge_{i=0}^{k-1} (u = t_i \land v = t_{i+1}) \rightarrow A(u, v))$$

3 Binary Decision Diagrams

3.1 Canonical Form

Suppose we have some set $S$ of objects, with an equivalence relation. A canonical form [1] is given by designating some objects of $S$ to be “in canonical form”, such that every object under consideration is equivalent to exactly one object in canonical form. In other words, the canonical forms in $S$ represent the equivalence classes, once and only once. To test whether two objects are equivalent, it then suffices to test their canonical forms for equality.

There are three possible canonical representation of WFF’s of propositional logic:

• The CNF’s of propositional WFF’s
• The DNF’s of propositional WFF’s
• The truth-table representation of propositional WFF’s

However, neither CNF nor DNF is a canonical representation of propositional WFF’s. Counterexamples are in Handout 26 [2]. As a matter of fact, truth table is a canonical representation of propositional WFF’s. However, its canonicity comes with a heavy price, i.e., the size of the truth table of a propositional WFF with size $n$ would be $O(2^n)$. 

Quantified Boolean Formulas, and Binary Decision Diagrams 2
Table 1: Logical Operations

| ¬ | ¬ |
| ∧ | ⋅ |
| ∨ | + |
| → | → |

Figure 1: Example of BDT representation

3.2 Boolean Functions

The definition of boolean functions is in the second paragraph of Sect 6.1 [LCS, page 358]. In short, a boolean function \( f \) of \( n \) arguments is a function from \( \{0, 1\}^n \) to \( \{0, 1\} \). The logical operations in propositional formulas (left column) and truth tables (right column) are shown in table 1.

3.3 Binary Decision Tree (BDT)

The BDT in Figure 1 logically represents the Boolean Function \( f(x, y) \triangleq x \cdot \overline{y} \). The circle nodes are internal nodes, which are called “decision points”, and the square nodes are called “leaf nodes”. Notice that, a BDT has three main characteristics:

- Every level of internal nodes is labelled with some propositional variable.
- Every internal node has two branches–one dashed (e.g., \( x \mapsto 0 \), \( y \mapsto 0 \)); one plain (e.g., \( x \mapsto 1 \), \( y \mapsto 1 \)).
- Every leaf node is 0 or 1.

From this example, we can see the size of the above BDT is 4, i.e., it has 4 leaf nodes. Notice that, the size of the BDT is not affected by the ordering of the variables, which is illustrated in Figure 2.
3.4 Transform BDT to BDD

Actually, BDTs are a special case of BDDs. In a BDD, we allow the following reduction rules:

C.1 Sharing of equal leaf nodes, which is illustrated in Figure 3.

C.2 Removing redundant decision points, which is illustrated in Figure 4.

C.3 Merge duplicate internal nodes (i.e., identical sub-WFF), which is illustrated in Figure 5, 6.
Figure 5: The original BDD

(a) Merge 2 and 3
(b) Merge 1 and 2, 3
(c) Remove one redundant x

Figure 6: Merge duplicate internal nodes
3.5 Reduced Ordered Binary Decision Diagrams (ROBDD’s)

Notice that, if we apply repeatedly reduction rules \{C1,C2,C3\} to $\text{BDT}(\phi)$ [2], we obtain a ROBDD w.r.t. to a specific ordering of the variables. Some facts are illustrated on pages 13 – 16 in Handout 26. An important fact is that ROBDD’s are canonical.

3.6 Operations on BDD representation

Given BDDs representation of $\phi$ and $\psi$ (i.e., $B_\phi$ and $B_\psi$, respectively), how to get efficiently $B_{\neg \phi}$, $B_{\phi \land \psi}$, and $B_{\phi \lor \psi}$?

- For $B_{\neg \phi}$, it just needs to switch the leaf nodes.
- For $B_{\phi \land \psi}$, it just needs to take every leaf node 1 in $B_\phi$ and replace it by $B_\psi$.
- For $B_{\phi \lor \psi}$, it just needs to take every leaf node 0 in $B_\phi$ and replace it by $B_\psi$.

References:
