Design Verification

Lecture 08 - Multi-Level Logic Verification: BDD 3

1. BDD construction order

   - different from variable order
   - most common way for building BDD is bottom-up, thus need to know which sub-functions to build first, and how to combine the sub-functions
   - motivation: the final BDD for function f may be *smaller* than some BDDs for intermediate BDDs for the function → pick the right order of intermediate BDDs can improve performance

   Example 1

2. Heuristic 1: size-based

   - let BDD = apply(BDD_1, BDD_2, $\odot$)
   - worst case size $|BDD| = |BDD_1| \times |BDD_2|$
   - best case size $|BDD| = \min(|BDD_1|, |BDD_2|)$
     $\longrightarrow$ think back to apply() of 2 BDDs
   - idea: if we start with small BDDs, the growth rate increases more slowly
   - key: if size $|BDD| = |BDD_1| + |BDD_2|$ for all cases, then optimal schedule can be obtained

3. Heuristic 2: support-based
   - let support(BDD) = variables this BDD depends on
   - if $support(BDD_1) \cap support(BDD_2) = \emptyset$, then BDD can simply be constructed by concatenating the two BDDs.
   - if $support(BDD_1) = support(BDD_2)$, then $|BDD| = \min(|BDD_1|, |BDD_2|)$
   - if $support(BDD_1) \subset support(BDD_2)$, then ?
   - key: pick the right sub-BDDs first to combine
4. Heuristic 3: combined size+support
   - first divide sub BDDs into disjoint sets. Each set shares some support
   - within each disjoint set, order them by size and support
     - maximum subset of support
     - least extra support
     - smallest BDD size
5. Heuristic 4: Partial traversal
   - given BDD_1 and BDD_2 for subfunctions f^1 and f^2
   - estimate size of resulting BDD by computing the sizes of co-factored BDDs f^1_p and f^2_p, where p is a cube containing the first k variables
   - let BDD_{1p} be the BDD for f^1_p
   - weight =
     $$\sum_{\forall\ values\ of\ p} f(|BDD_{1p}|, |BDD_{2p}|)$$
     - if either BDD_p is a terminal 0, $f() = 0$
     - if one BDD_p is a terminal 1, $f()$ = size of the other BDD_p
     - else, $f() = |BDD_{1p}| \times |BDD_{2p}|$
6. Results
   - support-based gives smallest final BDD sizes in general
   - support-based also gives smallest maximum-intermediate BDD sizes
   - support-based fastest in run time

7. Handling circuits whose OBDD sizes are exponential
   → example circuit: OBDD for multiplier exponential
   → But general BDD for multiplier is $O(n^3)$, recall that in a general (free) BDD, var order of right path and left path can differ
   → replicate primary inputs to remove (most) fanouts to form new circuit C'
   → construct OBDD for circuit C'
   → smooth replicated input variables

   Example 2: Replicating inputs

8. Once we build BDD for C', even though C' may be satisfiable, the original C may not, because the replicated variables take on different values
   $\longrightarrow$ Thus, need to *smooth* the replicated inputs

9. Input smoothing

   - sequentially smooth away one input at a time
   - let the replicated inputs for x_i be $\{x_{i,0}, ..., x_{i,p}\}$
   - smoothing means all the replicated $x_{i,j}$ have the same value
     → $S_{x_i} f = f_{x_{i,0},...,x_{i,p}} + f_{\overline{x_{i,0}},...,\overline{x_{i,p}}}$
     → essentially cofactoring f w.r.t the cube corresponding to all $x_{i,j}$'s set to 0 and set to 1, and OR the results

- Note that $S_{x_i} f$ may have a bigger BDD than prior to smoothing
  $\rightarrow$ Order to smooth the replicated variables important
  $\rightarrow$ May pick the variables furthest away from BDD root first, since they affect only the portions below them
- a function is completely smoothed when there are no more replicated variables (BDD now only depends on non-replicated variables)
- The BDD for completely smoothed function has better chance to be smaller, thus simpler to check for tautology

Example 3

10. Partitioned OBDDs (POBDD)
    - divide boolean space into k partitions
    - a separate BDD for each paritition
    - sum of all partitioned BDDs exponentially smaller than monolithic BDD
    - partitioned BDDs also canonical
11. Definition of POBDD
    - Given a function f, POBDD $= \{(w_1, f^1), ..., (w_k, f^k)\}$, over k partitions
    - where w_i is a Boolean function (called "window function") representing the partition and
    - $w_1 + w_2 + ... + w_k = 1$ (i.e., the partitions cover the entire Boolean space)

- $f^i = w_i f$
- both w_i and f^i can be represented as ROBDDs, with variable ordering π_i

12. A Simple Example POBDD
    - $f(x_1, ..., x_n) = x_1 f^1(x_2, ..., x_n) + \overline{x_1} f^2(x_2, ..., x_n)$
    - $w_1 = x_1$, $w_2 = \overline{x_1}$. Note $w_1 + w_2 = 1$
    - f^1 and f^2 may have different variable orderings π_1, π_2
    - Traversing POBDD first examines the partitioning variables, then traverses the target OBDD partition
13. POBDD is canonical
    - for a given partition $W = (w_1, ..., w_k)$, OBDD for each f^i is canonical for a specified var order π_i
    - thus, for the specified partition W, the POBDD is canonical
14. POBDD size smaller than monolithic ROBDD
    - Let $f(x_1, ..., x_n) = x_1 f^1(x_2, ..., x_n) + \overline{x_1} f^2(x_2, ..., x_n)$
    - also assume that the smallest ROBDD for f^1 is under π_1 and for f^2 is under π_2
    - Then, $f(x_1, ..., x_n) = x_1 f^1(x_2, ..., x_n) + \overline{x_1} f^2(x_2, ..., x_n)$ under the given partition is smallest
    - Suppose either f^1 under π_2 or f^2 under π_1 is exponential, then a monolithic ROBDD for f would be exponential
15. Manipulations on POBDDs
    - Boolean operations: (AND, OR, NOT, etc. of f and g), if partition of f and g the same (i.e., $W_f \equiv W_g$), then boolean operation is simply applied to each partition
    - co-factor: co-factor each w_i and f_i with co-factoring variable x
      $\rightarrow$ Note: the new co-factored window W changed, but still sum to 1
    - Existential Quantification: $\exists_x f = f_x + f_{\bar{x}}$
      $\rightarrow$ f_x and $f_{\bar{x}}$ may have different window functions, W, if x is the co-factoring variable
      $\rightarrow$ Avoid taking $\exists_x f$ with respect to variables in the window function

- Universal Quantification: $\forall_x f = f_x f_{\bar{x}}$

16. Selection of window functions

- Example: partition on x_1 and $x_2 \rightarrow 4$ partitions: x_1x_2, $x_1\overline{x_2}$, $\overline{x_1}x_2$, $\overline{x_1x_2}$ $\longrightarrow$ this POBDD guaranteed to be smaller than monolithic BDD
- key: select variables that maximize partitioning while minimizing redundancy among different partitions
- let partiion-factor $p_x =$

$$max(\frac{|f_x|}{|f|}, \frac{|f_{\bar{x}}|}{|f|})$$

and redundancy-factor $r_x =$

$$\frac{|f_x| + |f_{\bar{x}}|}{|f|}$$

- Then, cost of partition on variable $x = \alpha p_x + \beta r_x$, where α and β are constants

17. Application to combinational equivalence checking

- create a miter circuit for each PO, and generate POBDD for each miter PO
- for each partition ROBDD, check for tautology. If any fail $\rightarrow$ not equivalent