Exercise session
Binary Decision diagrams

Introduction

We are interested in electronic circuits with state (latches, memories) given as a vector of Booleans $x_1, \ldots, x_n$ initially equal to $i_1, \ldots, i_n$. From a state $x_1, \ldots, x_n$ the circuit may transition to a state $x'_1, \ldots, x'_n$ if and only if a formula $f_T(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$ is true. All memories are updated simultaneously at every clock tick.

The problem of verification is to ensure that the designed circuit behaves as intended. It is possible to *test* the circuit on a large number of inputs (*test vectors*), but this is costly since it means bugs are detected at a late stage.[1] It is also possible to *simulate* the circuit on a large number of test vectors, but it is in general impossible to cover all possible inputs. It is possible, even though all tests pass, that there is a bug — testing is not *exhaustive*. In contrast, in verification we wish to ensure that the circuit will never go into an undesirable state, for any input. In other words we wish to show that undesirable states are *unreachable*, by an automated method.

This exercise sheet aims at describing one of the main data structures used for the verification of Boolean circuits: *binary decision diagrams* (BDD) [3, ch. 7.1.4] [2, ch. 5], made fashionable by Bryant in 1986 [1].

As the system has only finitely many possible states, one could want to manipulate explicit sets of states (*explicit-state model-checking*), but these sets tend to have a large size ($\sim 2^n$), too large to fit in memory and too large for efficient algorithmic operations. BDD are used to cleverly represent such states; in the worst case they will also have prohibitively large size, but often they will be much more compact.

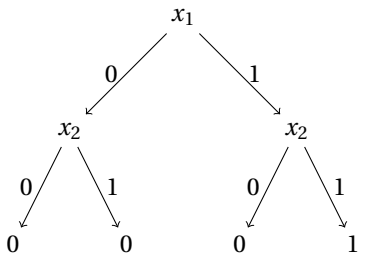
1.1 Basics

BDDs are a compact way to represents functions from B^n to B, where $B = \{0, 1\}$.

To each function $f : B^n \to B$ and word $w \in \{0,1\}^*$ of length $0 \leq |w| \leq n$ the following tree, defined by recurrence on $n - |w|$:

- If $|w| = n$, the tree is a leaf labeled by $f(w)$.
- If $|w| < n$, the tree is a node labeled by $|w|$, with two sons, the trees associated to f and the words $w0$ and $w1$ respectively, the respective edges being 0 and 1. If the arguments of the f function are $x_1, \ldots, x_n$, we can label the internal node by $x_{|w|}$.

Thus, if f is the function from B^2 to B defined by $f(x_1, x_2) = x_1 \wedge x_2$, the associated tree is:



Question 1. Which tree is associated to $(\neg a \wedge \neg b) \vee (a \wedge b)$?

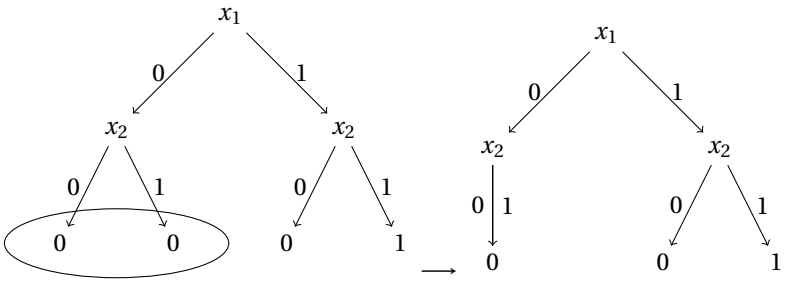
Question 2. What are the number of leaves and internal nodes, as a function of n?

[1] One often quoted cost is $1,000,000 for producing and testing a prototype circuit.

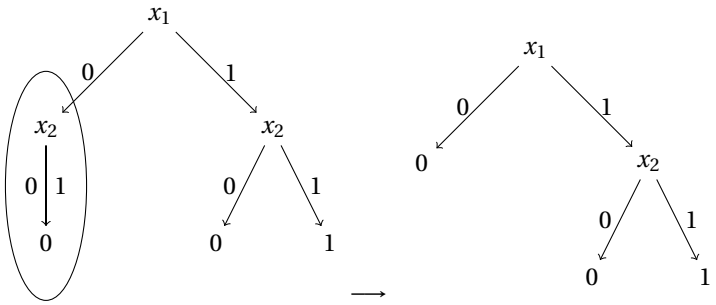
Now we shall allow subtrees to be shared: several edges may point to the same subtree. Thus, we no longer have trees, strictly speaking, but connected DAGs (directed acyclic graphs). A node is therefore either a leaf, either an internal node with two children, labeled with a variable, and with edges labeled 0 or 1. Such DAGs are called *binary decision diagrams*.

In order to *reduce* the diagram with apply the following operations:

1. Fusion of two identical subtrees: several edges will now point to the root. For instance, one can fuse the two boxed leaves:



2. Removal of a node with identical children: a node whose 0 and 1 outgoing edges point to the same node is removed.



A diagram is said to be *reduced* if these operations are not possible anymore. One way to obtain the reduced binary decision diagram (ROBDD) from a function from B^n to B is thus to construct the associated tree, then reduce it. (This is not the right way to do it algorithmically, since the tree may be very large even though the BDD is compact.)

Question 3. Give the ROBDD for $(x_1, x_2, x_3) \mapsto x_1 \wedge x_2 \wedge x_3$.

Question 4. Show the existence and unicity of the ROBDD associated to a given function $f : B^n \to B$.

Question 5. Given 2 BDDs A and B representing $f_A, f_B : B^n \to B$, how to represent $f_A \wedge f_B$? Give a simple algorithm, under the assumption that the order of variables stays the same.

Similarly, we obtain an algorithm for the "or" operation.

1.2 Reachability

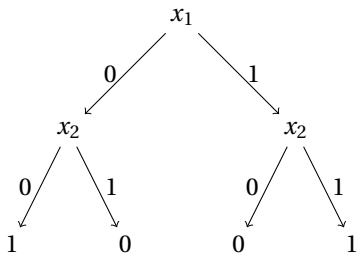
We identify to each set of vectors $(x_1, \ldots, x_n) \in B^n$, its characteristic function $B^n \to B$ and its associated ROBDD.

Question 6. Express the intersection and the union of two sets of vectors in terms of characteristic functions.

The *transition relation* of the system is the set of pairs $\big((x_1, \ldots, x_n), (x'_1, \ldots, x'_n)\big)$ such that it is possible to move from $(x_1, \ldots, x_n)$ to $(x'_1, \ldots, x'_n)$ in exactly one computational step. A transition relation can thus be represented by a ROBDD over variables $x_1, \ldots, x_n, x'_1, \ldots, x'_n$ (not necessarily in that order). post(X) denotes the set of states reachable in exactly one computational step from X. Otherwise said:

$$\text{post}(X) = \{(x'_1, \ldots, x'_n) \mid \exists x_1, \ldots, x_n \in X \wedge T(x_1, \ldots, x_n, x'_1, \ldots, x'_n) = 1\} \tag{1.1}$$

Question 7. Consider the ROBDD:



It represents a set $A \subseteq B^2$. What are the ROBDDs representing the formulas $\exists x_1\ (x_1, x_2) \in A$ and $\exists x_2\ (x_1, x_2) \in A$? (Try obtaining them directly from the ROBDD instead of getting them from A.)

Question 8. Let us now generalize this operation. Given a ROBDD R over variables $v_1, \ldots, v_m, w_1 \ldots, w_n$ (not necessarily in that order), how can we obtain a ROBDD R' over $v_1, \ldots, v_m$ representing $\{(v_1, \ldots, v_m) \in B^m \mid \exists (w_1, \ldots, w_n) \in B^n (v_1, \ldots, v_m, w_1 \ldots, w_n) \in R\}$?

Question 9. Given a ROBDD T over variables $x_1, \ldots, x_n, x'_1, \ldots, x'_n$ and a ROBDD X, how can we compute a ROBDD X' over variables $x'_1, \ldots, x'_n$ representing post(X)?

Question 10. How can we compute the set of states that the system can reach? We shall temporarily admit that we know how to test equality between functions represented by BDDs (see question 20).

Question 11. How do we check if an error state is reachable? (We assume we are given a formula f_E representing the error states.)

Question 12. Propose an algorithm that would reach the same result but by computing an ascending sequence of length linear in n, but on ROBDDs with $2n$ variables.

Hint: consider the operation $A \circ B$ over ROBDDs, defined as : $x_1, \ldots, x_n, x''_1, \ldots, x''_n$ is in $A \circ B$, with A over $x'_1, \ldots, x'_n, x''_1, \ldots, x''_n$ and B over $x_1, \ldots, x_n, x'_1, \ldots, x'_n$, by $\exists x'_1, \ldots, x'_n\ A \wedge B$.

Note: it is possible to test reachability using only polynomial memory space (in fact, the problem is PSPACE-complete). Unfortunately the polynomial-space methods are impractical.

1.3 Size

We shall now see that the choice of the variable ordering may have considerable consequences on the size used by ROBDDs (*space complexity*).

Let us denote by $a \oplus b$ the "exclusive or", that is $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$. Otherwise said, "exclusive" or is the addition, and "and" the multiplication, in the field of two elements.

Question 13. Construct a ROBDD for $(a, b, c) \mapsto a \oplus b \oplus c$.

Question 14. More generally, how many nodes are needed in the ROBDD for $(x_1, \ldots, x_n) \mapsto x_1 \oplus \cdots \oplus x_n$?

Question 15. Give a ROBDD for $(a, b, c, d, e, f) \mapsto (a \oplus b) \wedge (c \oplus d) \wedge (e \oplus f)$.

Question 16. Generalize to a ROBDD representing $(x_1, y_1, \ldots, x_n, y_n) \mapsto (x_1 \oplus y_1) \wedge \cdots \wedge (x_n \oplus y_n)$.

Question 17. Describe the ROBDD for $(a, c, e, b, d, f) \mapsto (a \oplus b) \wedge (c \oplus d) \wedge (e \oplus f)$. How many nodes does it have?

Question 18. Generalize to a ROBDD representing $(x_1, \ldots, x_n, y_1, \ldots, y_n) \mapsto (x_1 \oplus y_1) \wedge \cdots \wedge (x_n \oplus y_n)$.

Note that for the same function, the size of the ROBDD varies considerably with variable ordering!

1.4 Advanced algorithm

The algorithm presented at Question 5 does not guarantee the minimality of diagrams (there may be several isomorphic sub-diagrams), and may recompute several times the same thing. To avoid this, we use *hash tables*.

Question 19. How to avoid ever getting two isomorphic but different diagrams?

Question 20. Let us suppose that we have a system ensuring that there are never two different yet isomorphic ROBDDs in the computer memory. How do we check the equivalence of the functions represented by two ROBDDs?

Question 21. Modify the naive algorithm so that he does not recompute several times $f_A \wedge f_B$ as he get through the subtrees A and B.

1.5 For the more curious

Remark that with the above definitions, there is no sharing between the diagrams representing f and $\neg f$. In order to allow such sharing, one can use *signed ROBDDs*. In this case, each node contains, in addition to the variable and two children, an indicator saying whether the truth value of the "1" subtree should be inverted. In addition, there is a Boolean at the root saying whether the truth value of the whole function should be inverted.

Question 22. Show that such signed ROBDDs provide a unique representation for every Boolean function.

Using a low-level programming language, one can further optimize representation by using the lowest-order bit of a pointer to store the indicator Boolean, since pointers are even.

Bibliography

[1] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, August 1986.

[2] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[3] Donald Ervin Knuth. *The Art of Computer Programming, 4a: Combinatorial algorithms, part I*. Addison-Wesley, 2011.