Ordered Choice Diagrams for Symbolic Analysis

Rajeev Alur
University of Pennsylvania, alur@cis.upenn.edu

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation


This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/104
For more information, please contact repository@pobox.upenn.edu.
Ordered Choice Diagrams for Symbolic Analysis

Abstract
We propose ordered choice diagrams (OCD) for symbolic representation of boolean functions. An OCD is a nondeterministic variant of an ordered binary decision diagram (BDD) with appropriate reduction rules. The introduction of nondeterminism destroys canonicity, but affords significant succinctness. While OCDs have efficient algorithms for union, intersection, existential quantifier elimination, and emptiness test, the equivalence problems for OCDs is coNP-complete. We show that symbolic model checking can still be performed efficiently by replacing equivalence test with a stronger equality test. We report on a prototype implementation and preliminary results that indicate the advantage of OCD-based representation in reachability analysis of distributed protocols.

Comments

This technical report is available at ScholarlyCommons: https://repository.upenn.edu/cis_reports/104
Ordered Choice Diagrams for Symbolic Analysis

Rajeev Alur
University of Pennsylvania
alur@cis.upenn.edu
http://www.cis.upenn.edu/ alur/

May 19, 1998

Abstract

We propose ordered choice diagrams (OCD) for symbolic representation of boolean functions. An OCD is a nondeterministic variant of an ordered binary decision diagram (BDD) with appropriate reduction rules. The introduction of nondeterminism destroys canonicity, but affords significant succinctness. While OCDS have efficient algorithms for union, intersection, existential quantifier elimination, and emptiness test, the equivalence problem for OCDS is coNP-complete. We show that symbolic model checking can still be performed efficiently by replacing equivalence test with a stronger equality test. We report on a prototype implementation and preliminary results that indicate the advantage of OCD-based representation in reachability analysis of distributed protocols.
1 Introduction

Binary decision diagrams (BDD) have been widely used as a representation of boolean formulas in many CAD applications [Bry86, Bry95]. In particular, their application to symbolic model checking of temporal properties, introduced in [BCD+92] and implemented in many tools [McM93, HHK96, ?], has proved to be an effective technique for battling the state-space explosion problem. In spite of the steady progress in improving the performance of BDD-packages, BDDs can offer compaction of state-space representation only in a heuristic way, and memory requirements continue to be a bottleneck in applicability of model checking, demanding further research. In this paper, we propose a variant of BDD, called ordered choice diagram (OCD), as a possible improvement in this regard.

Ordered choice diagrams represent boolean functions over boolean (or enumerated) variables as labeled directed acyclic graphs. The internal vertices of the graph are labeled with variables, and the terminal vertices are labeled with truth values 0 and 1. The edges out of an internal vertex are labeled with 0 or 1, and the paths in the graph encode assignments of truth values to the variables. As in ordered binary decision diagrams, it is required that there is a global (total) ordering of the variables and the sequence of variables appearing along every path is consistent with this total order. In an ordered choice diagram, unlike in a BDD, an internal vertex can have multiple outgoing edges labeled with the same truth value. Given an initial vertex and an assignment of truth values to all the variables, there can be multiple paths leading to terminal vertices, and the function associated with the initial vertex evaluates to 1 if there is some path leading to the terminal vertex 1. Thus, while a BDD is like a deterministic automaton, an OCD is like a nondeterministic one.

A key property of BDD representation of a boolean function is its canonicity. This is achieved by enforcing certain reduction rules. Definition of an OCD also enforces certain reduction rules. These reductions are analogous to the reductions afforded by similarity relations for nondeterministic automata [KS90], and are computable in polynomial time. However, they do not ensure canonicity. A boolean function can have multiple, but only a bounded number of, OCD representations. Not surprisingly, the problem of determining whether two OCDs represent the same function is computationally hard, namely, coNP-complete. What we gain by losing canonicity is succinctness. We show that OCD representation of a boolean function, for instance, the hidden-weighted-bit function, can be exponentially succinct compared to the BDD representation.

While canonicity, and resulting efficient test for equivalence, is central to verification of combinational circuits, its importance in model checking, particularly in invariant verification, is debatable. For symbolic model checking using fixpoint computation, we need a representation of state-sets that supports (1) operations such as union, intersection, and quantifier elimination, (2) test for emptiness, and (3) test for equivalence. In invariant verification, emptiness test is needed to detect violation of the invariant, and equivalence (or inclusion) test is needed to detect that all reachable states have been encountered. In practice, violations are more prevalent and more useful. At an extreme, if we only attempt to falsify an invariant, we can do away with the equivalence test.

For application of OCDs to model checking, we show that operations such as union, intersection, and quantifier elimination, can be implemented effectively with the same cost as for BDDs (complementation of OCDs is computationally hard, but is not needed in model checking). Test for emptiness is constant-time, and we replace test for equivalence by a sufficient and efficient test for equality. This means that the algorithm may not terminate in the iteration in which the fixpoint is attained. However, it is eventually guaranteed to terminate because there are only bounded number of distinct equivalent OCDs.

We also report on a prototype implementation of OCDs for symbolic reachability analysis.
Figure 1: A sample ordered choice diagram

Preliminary experimental results concerning asynchronous distributed protocols indicate that our approach can provide a saving in space requirements. This is particularly evident in the case of point-to-point process networks in which there is no obvious way of ordering variables of different processes.

Related work

Many variants of BDDs, such as BMD and ZBDD, have been introduced in the literature aimed at providing succinct representation of functions that do not have compact BDD representation (see [Bry95] for a survey). Canonicity has been a central concern in most of this research.

For symbolic model checking of asynchronous protocols, [DHY94] suggests to maintain the transition relation in a disjunctive partitioned form. This can be viewed as a special case of OCD. Similarly, maintaining boolean formulas as a sum of product is also a special case of OCD.

While OCD representation is inspired by classical nondeterministic finite automata, it differs in two important ways: it is acyclic and it is always reduced (see Figure 4 for resulting impact on complexity). In [Waa96], a nondeterministic variant of BDDs, called POBDD, is defined. While syntactically similar to an OCD, a POBDD has completely different semantics obtained by counting the number of paths leading to the two terminal vertices.

It is known that introducing auxiliary variables can sometimes reduce the BDD size. Nondeterminism can imitate auxiliary variables without explicitly introducing them. Furthermore, while introduction of auxiliary variables is ad hoc, the use of nondeterminism in OCDS is systematic.

2 Ordered Choice Diagrams

2.1 Definition

A sample OCD is shown in Figure 1. The vertex \( v_0 \) represents the function \( a(bd + cd) \) for the variable ordering \( a < b < c < d \). The vertex \( v_0 \) has two edges for the case \( a = 1 \) corresponding to the two disjuncts \( bd \) and \( cd \).

Instead of defining individual ordered decision diagrams, we define an OCD-space to be a global data structure containing all functions of interest. Let \( X \) be a (finite) set of boolean variables, and let \( < \) be a total order over \( X \).
\textbf{Definition 1} An OCD-space over \((X, \prec)\) consists of the following components:

- A (finite) set \(V\) of OCD-vertices.
- A (finite) set \(B\) of OCD-indices; an element of \(B\) is either the constant \(0\), or the constant \(1\), or a nonempty subset of \(V\). The latter type of indices are called nonterminal indices.
- A labeling function \(L\) that maps every vertex \(v\) to a variable \(L(v) \in X\).
- A left-child function \(C_l\) that maps every vertex \(v\) to an index \(C_l(v) \in B\) such that if \(C_l(v)\) is nonterminal and contains a vertex \(w\) then \(L(v) < L(w)\).
- A right-child function \(C_r\) that maps every vertex \(v\) to an index \(C_r(v) \in B\) such that if \(C_r(v)\) is nonterminal and contains a vertex \(w\) then \(L(v) < L(w)\).

Each vertex and each index represents a boolean function over the set \(X\). Formally, we define a mapping \(P\) from \(B \cup V\) to boolean formulas over \(X\).

\textbf{Definition 2} The function \(P\) associates a boolean function over \(X\) with every element of \(B \cup V\):

- \(P(0) = \text{false} \), and \(P(1) = \text{true} \).
- For a vertex \(v\), \(P(v) = (\neg L(v) \land P(C_l(v)) \lor (L(v) \land P(C_r(v)))\).
- For a nonterminal index \(b = \{v_1 \ldots v_k\}\), \(P(b) = P(v_1) \lor \cdots \lor P(v_k)\).

Because of the ordering constraint in the definitions of the left-child and right-child functions, the mapping \(P\) is well-defined.

In binary decision diagrams, reduction is obtained by identifying isomorphic subtrees and by deleting vertices with identical children. In ordered decision diagrams, a third form of reduction is possible due to functional redundancy. To illustrate this reduction, consider the graph of Figure 2 (as a convention, we omit the edges leading to the terminal vertex \(0\)). The vertex \(v_1\) represents the function \(bc\), while the vertex \(v_2\) represents the function \(bc + bd\). The edge from \(v_0\) to \(v_1\) is redundant, that is, the function represented by \(v_0\) is \(ab(c + d)\) irrespective of the presence of this edge. In this case, the fact that \(P(v_1)\) implies \(P(v_2)\) can be detected by a simple bottom-up analysis. In our definition of reduced diagrams, we require that such redundancies are absent. In particular, we consider the set \(\{v_1, v_2\}\) to be an illegal index. To define reduction in ordered choice diagrams, we start by defining a binary relation over the vertices and indices.

\textbf{Definition 3} The binary relation \(\rightarrow\) over \(B \cup V\) is the least relation satisfying all the following clauses:

FIGURE 2: Redundancy in ordered choice diagrams

![Diagram](image-url)
1. for every $\alpha \in V \cup B$, $0 \to \alpha \to \alpha \to 1$.
2. for nonterminal indices $b$ and $c$, $b \to c$ if for all $v \in b$, $v \to c$.
3. for a vertex $v$ and a nonterminal index $b$, $b \to v$ if $v \to w$ for some $w \in b$, and $b \to v$ if $w \to v$ for all $w \in b$.
4. for two vertices $v$ and $w$, $v \to w$ if (1) $L(v) = L(w)$ and $C_l(v) \to C_l(w)$ and $C_r(v) \to C_r(w)$; or (2) $L(v) < L(w)$ and $C_l(v) \to w$ and $C_r(v) \to w$. ■

The next proposition asserts that whenever two elements are related by this relation, their boolean functions are related by logical implication.

**Proposition 1** If $\alpha \to \beta$ then $\neg P(\alpha) \lor P(\beta)$ is a valid formula. ■

Consequently, if $v \to w$ then from an index $b$ containing both $v$ and $w$, we can delete $v$ without altering the boolean function $P(b)$. We use this property to define reduction.

**Definition 4** The OCD-space is said to be reduced if
1. there are no two distinct vertices $v$ and $w$ in $V$ such that $v \to w$ and $w \to v$,
2. for a nonterminal index $b$, there are no two distinct vertices $v$ and $w$ in $b$ with $v \to w$,
3. for a vertex $v$, $C_l(v) \neq C_r(v)$. ■

Henceforth, unless otherwise stated, we consider only reduced OCD-space. We represent a boolean function by an index in the OCD-space. It is clear that every boolean function over $X$ is represented by some index. Unlike binary decision diagrams, the OCD-representation is not canonical. That is, two distinct indices may represent the same boolean function. This is because the binary relation $\rightarrow$ over $V \cup B$ is a sufficient, but not necessary, condition for the logical implication of the associated functions. For instance, consider the two vertices $v = (a, 0, 1)$, and $w = (a, 1, 0)$. The index $\{v, w\}$ represents the same function as the index 1.

As discussed earlier, ordered choice diagrams can be viewed as labeled graphs. Specifically, let us consider the graph $G$ that has a node for each vertex in $V$, and two special nodes 0 and 1. The vertex $v$ is labeled with the variable $L(v)$. If $C_l(v) = 0$ then there is an edge from $v$ to 0 labeled with 0; if $C_l(v) = 1$ then there is an edge from $v$ to 1 labeled with 0; else for every $w \in C_l(v)$, there is an edge from $v$ to $w$ labeled with 0. Similarly, $C_r(v)$ determines edges from $v$ labeled with 1. Once we view the OCD-space as a labeled graph, we can consider various equivalence relations over the vertices. In particular, let us consider the simulation relations. A binary relation $\preceq$ over the vertices $V$ is a simulation relation iff whenever $v \preceq w$, we have (i) $L(v) = L(w)$, (ii) either $C_l(v) = C_l(w)$, or both $C_l(v)$ and $C_l(w)$ are nonterminal indices such that for every $v' \in C_l(v)$ there exists $w' \in C_l(w)$ with $v' \preceq w'$, and (iii) either $C_r(v) = C_r(w)$, or both $C_r(v)$ and $C_r(w)$ are nonterminal indices such that for every $v' \in C_r(v)$ there exists $w' \in C_r(w)$ with $v' \preceq w'$. The vertex $w$ simulates the vertex $v$ if there exists a simulation $\preceq$ such that $v \preceq w$. Two vertices are $v$ and $w$ are similar if $v$ simulates $w$ and $w$ simulates $v$.

**Proposition 2** If the vertex $w$ simulates the vertex $v$ in an OCD-space, then $v \to w$. ■

As a corollary, we obtain that no two distinct vertices of a reduced OCD-space are similar. It follows that no two distinct vertices of a reduced space are bisimilar to each other, which implies that no two distinct vertices are isomorphic. We note that our rules of reduction may collapse even similar states, and thus, afford greater reduction compared to the similarity partition studied for nondeterministic transition systems [KS90].
2.2 Succinctness compared to binary decision diagrams

For an element $\alpha \in V \cup B$, let $\text{reach}(\alpha)$ be the set of elements reachable from $\alpha$. Traversal originating at a vertex or an index corresponds to visiting all the elements reachable from it. The size $\text{size}(\alpha)$ of an element $\alpha$ is then defined to be the sum of the cardinality of $\text{reach}(\alpha)$ and the sum of the cardinalities of all the nonterminal indices $b \in \text{reach}(\alpha)$. Thus, the size is the number of vertices and edges in the reachable subgraph.

Consider a vertex $v$ such that every nonterminal index in $\text{reach}(v)$ contains only one vertex. Then, the graph containing the vertices in $\text{reach}(v)$ represents a binary decision diagram. This is because the reduction rules for OCD coincide with those for BDD when we restrict attention to singleton vertices. Consequently, a binary decision diagram is a special case of an ordered choice diagram. Hence, for a given variable ordering, every boolean function has some OCD representation whose size is at most the size of the BDD representation of that function. However, a boolean function can have multiple OCD representations, and some of them can have size larger than the BDD representation. The following two examples illustrate that the nondeterminism allowed in an OCD can lead to an exponentially succinctness compared to BDD.

Let $X$ contain $2n$ variables with the order $x_1, x_2, \ldots x_{2n}$. Consider the boolean predicate:

$$f = (x_1 \leftrightarrow x_{n+1}) \lor (x_2 \leftrightarrow x_{n+2}) \lor \cdots \lor (x_n \leftrightarrow x_{2n})$$

The BDD representation of $f$ (for the chosen ordering) is exponential in $n$. However, the OCD representation is linear in $n$. For each $1 \leq i \leq n$, the disjunct $x_i \leftrightarrow x_{n+i}$ can be represented by a vertex $v_i$ of constant size. The index $\{v_1, \ldots v_n\}$ represents the function $f$. For instance, the OCD representation of $(x_1 \leftrightarrow x_3) \lor (x_2 \leftrightarrow x_4)$ is shown in Figure 3.

Another example where the OCD representation leads to an exponentially succinct representation is the hidden-weight-bit function. Let $X$ contain $n + 1$ variables $x_1, \ldots x_{n+1}$, and consider the predicate

$$g = x_{n+1} \leftrightarrow x_i \text{ where } i = \Sigma_{1 \leq j \leq n} x_j$$

It is known that irrespective of the chosen order over $X$, the BDD representation of $g$ is exponential in $n$ [Bry91]. Observe that $g = \lor_{1 \leq i \leq n} g_i$, where $g_i$ is the predicate $(x_{n+1} \leftrightarrow x_i) \land (i = \Sigma_{1 \leq j \leq n} x_j)$. It is easy to verify that each $g_i$ has a BDD representation of size $O(n^2)$, and hence, $g$ has a OCD representation of size $O(n^3)$.
2.3 Decision problems

Membership test

A valuation $s$ for $X$ is an assignment of boolean values to the variables in $X$. The OCD membership problem is to determine, given an input valuation $s$ and an index $b$, whether $s$ satisfies the boolean function $P(b)$. This problem can be solved by traversing the OCD-space starting at $b$. The traversal can be performed by appropriately modifying standard graph search algorithms (e.g. depth-first search).

**Theorem 1** Given an index $b$ and a valuation $s$ for $X$, checking whether $s$ satisfies $P(b)$ requires time $O(\text{size}(b) + |X|)$.

Satisfiability

An element $\alpha$ is said to be *satisfiable* if some valuation $s$ for $X$ satisfies the function $P(\alpha)$. The reduced OCD-space has the property that every vertex represents a satisfiable function, and thus, the index 0 is the only unsatisfiable index:

**Proposition 3** For an index $b$, the boolean function $P(b)$ is satisfiable iff $b \neq 0$.

It follows that checking satisfiability can be performed in constant time.

Equivalence

An element $\alpha$ is said to be *valid* if every valuation $s$ for $X$ satisfies the function $P(\alpha)$.

**Theorem 2** The problem of deciding whether an index is valid is coNP-complete.

**Proof.** Given an index $b$, we can guess a valuation $s$ for $X$, and check if $s$ satisfies $b$. This shows that validity is in co-NP. For lower bound, consider a boolean formula $\varphi$ over $X$ in disjunctive normal form where each disjunct has 3 conjuncts. Checking validity of $\varphi$ is coNP-hard. We can build the OCD representation of $\varphi$ in polynomial time: each disjunct has 3 conjuncts, and hence, can be represented by a vertex that has 3 reachable vertices. The index that contains one vertex per conjunct represents $\varphi$. Thus, checking validity of $\varphi$ reduces to checking validity of an OCD of size linear in the size of $\varphi$.

Two elements $\alpha$ and $\beta$ are said to be equivalent if the boolean functions $P(\alpha)$ and $P(\beta)$ are equivalent. The complexity of checking equivalence of two elements is again coNP-complete.

**Theorem 3** Given two indices $b$ and $c$, the problem of deciding whether they are equivalent is coNP-complete.

The complexity of various decision problems is summarized in Figure 4. Observe that while choice diagrams are like nondeterministic automata in spirit, problems such as equivalence are computationally harder for nondeterministic automata. This difference arises from the acyclic nature of OCDs.

3 Algorithms for OCD manipulation

In this section, we describe how to represent and construct ordered choice diagrams. Our data structures and algorithms are modifications of the corresponding data structures and algorithms for manipulating binary decision diagrams [BRB90].
### Decision problem | BDD | OCD | Nondeterministic automata
--- | --- | --- | ---
Membership | LOGSPACE | NLOGSPACE | NLOGSPACE
Satisfiability | $O(1)$ | $O(1)$ | NLOGSPACE
Validity/Equivalence | $O(1)$ | coNP | PSPACE

Figure 4: Summary of complexity of decision problems

#### 3.1 Data structures

The vertices of the OCD-space are stored in the hash-table $V$, and the indices are stored in the hash-table $B$. An element of $V$ is a triple consisting of a variable name, two pointers to the elements in $B$ corresponding to the maps $C_l$ and $C_r$. An element of $B$ is either the special constant 0, the special constant 1, or a finite set of pointers to the entries in $B$. Initially, the set $V$ is empty and the set $B$ contains the two special constants. The procedures for insertion into these two data structures ensure that the OCD-space is always in a reduced form. The data structures support the following basic routines.

**Dominates**: Given two elements $\alpha$ and $\beta$ belonging to $V \cup B$, $\text{Dominates}(\alpha, \beta)$ returns true if $\alpha \rightarrow \beta$ holds, and false otherwise. The evaluation of $\text{Dominates}$ is recursive according to the rules defining the relation $\rightarrow$.

**CreateVertex**: Given a variable $x$ and two pointers $b$ and $c$ to $B$, the function $\text{CreateVertex}$ creates (if necessary) a new vertex in the table $V$ with label $x$, left-index $b$, and right-index $c$. Before insertion, $\text{CreateVertex}$ ensures that $b$ and $c$ are distinct, and the table $V$ does not already contain the tuple $(x, b, c)$.

**CreateIndex**: Given a set $\{v_1, \ldots, v_k\}$ of vertices, the function $\text{CreateIndex}$ creates a new index in the table $B$. First, it checks, for every pair $v_i$ and $v_j$ of vertices, whether $\text{Dominates}(v_i, v_j)$ holds, and if so, discards $v_i$ from the set. Before insertion, it checks whether the index is already in the table $B$.

**Merge**: Given two nonterminal indices $b$ and $c$, $\text{Merge}$ returns the union $b \cup c$ after deleting any $v \in b$ such that $\text{Dominates}(v, w)$ for some $w \in c$ and deleting any $w \in c$ such that $\text{Dominates}(w, v)$ for some $v \in b$.

To ensure that $\text{Dominates}$ does not perform unnecessary work, we use a table $\text{Done}$ to store results of all computations. The function $\text{Dominates}$ upon invocation with input $v$ and $w$, first checks if $\text{Dominates}(v, w)$ was computed earlier by consulting $\text{Done}$, and starts the recursive evaluation only if needed. Upon termination, it records its result in $\text{Done}$.

If every update to the hash-tables $V$ and $B$ is performed by the functions $\text{CreateVertex}$ and $\text{CreateIndex}$, then the OCD-space is always reduced. Of the three clauses in Definition 4, the second one is explicitly checked by $\text{CreateIndex}$, while the third one is explicitly checked by $\text{CreateVertex}$. If the OCD-vertices are created by the function $\text{CreateVertex}$, and the OCD-indices are created by the function $\text{CreateIndex}$, then the resulting OCD-space is reduced.

#### 3.2 Operations

Representation of boolean functions are obtained using primitives to build atomic diagrams and positive boolean operations. Unlike BDDs, complementing an OCD is computationally difficult, and hence, we do not support complementation.
**Atomic diagrams**

For a variable \( x \), the function \( \text{Atom}^+(x) \) returns the index representing the predicate \( x \). The function \( \text{Atom}^+(x) \) corresponds to first creating a vertex via \( \text{CreateVertex}(x, 1, 0) \), and then a singleton index containing this vertex. Analogously, for a variable \( x \), the function \( \text{Atom}^-(x) \) returns the index representing the predicate \( \neg x \), and is implemented via \( \text{CreateVertex}(x, 0, 1) \).

**Conjunction**

Given two indices \( b \) and \( c \), the function \( \text{Conj} \) computes the the conjunction of the predicates represented by \( b \) and \( c \). The function \( \text{Conj} \) is implemented using an auxiliary function \( \text{VertexConj} \) that computes the conjunction of two vertices.

The function \( \text{Conj} \) first checks if one of the input indices is a terminal index. In such a case, the result can be computed immediately. Suppose both \( b \) and \( c \) are nonterminal vertices. Observe that

\[
P(v) \land P(w) = \bigvee_{v \in b, w \in c} (P(v) \land P(w)).
\]

For every pair \( v \in b \) and \( w \in c \), the function \( \text{VertexConj}(v, w) \) is invoked to compute the index \( d_{vw} \) such that \( P(v) \land P(w) = P(d_{vw}) \). The next step is to merge all the sets corresponding to the indices \( d_{vw} \). The function \( \text{Conj} \), then, returns the index corresponding to the merged set.

The function \( \text{VertexConj} \) stores its results in the global table \( \text{Done} \). When invoked with input vertices \( v \) and \( w \), it first checks if \( \text{VertexConj}(v, w) \) was computed earlier. If not, then suppose \( v = (x, b, c) \) and \( w = (x', b', c') \). Depending upon the ordering of the variables \( x \) and \( x' \), there are three cases to be considered. If \( x = x' \), then

\[
P(v) \land P(w) = (x \land P(b) \land P(b')) \lor (\neg x \land P(c) \land P(c')).
\]

In this case, an index \( d \) corresponding to \( P(b) \land P(b') \) is created via \( \text{Conj}(b, b') \), an index \( d' \) corresponding to \( P(c) \land P(c') \) is created via \( \text{Conj}(c, c') \), and then, \( \text{VertexConj} \) returns the vertex \( (x, d, d') \). If \( x < x' \), then \( P(w) \) does not depend on \( x \). In this case, an index \( d \) corresponding to \( P(b) \land P(w) \) is created via \( \text{Conj}(b, \{w\}) \), an index \( d' \) corresponding to \( P(c) \land P(w) \) is created via \( \text{Conj}(c, \{w\}) \), and then, \( \text{VertexConj} \) returns the vertex \( (x, d, d') \). The remaining case \( x > x' \) is handled similarly. At the end of its computation, \( \text{VertexConj} \) stores its result in the table \( \text{Done} \) to be used in later invocations with the same inputs.

To analyze the complexity, consider two indices \( b \) and \( c \). Suppose \( \text{reach}(b) \) has \( n_1 \) and elements and \( \text{reach}(c) \) has \( n_2 \) elements. Suppose the sum of the cardinalities of the indices in \( \text{reach}(b) \) is \( m_1 \), and the sum of the cardinalities of the indices in \( \text{reach}(c) \) is \( m_2 \). In other words, the graph corresponding to \( b \) has \( n_1 \) vertices and \( m_1 \) edges, and the graph corresponding to \( c \) has \( n_2 \) vertices and \( m_2 \) edges. Then, the complexity of \( \text{Conj}(b, c) \) is \((m_1 + n_1) \cdot (m_2 + n_2))\). This is because for every pair \( \alpha \in \text{reach}(b) \) and \( \beta \in \text{reach}(c) \), the conjunction of \( \alpha \) and \( \beta \) is computed at most once. If \( d \) is the index for the conjunction, the reachable graph of \( d \) can have \( n_1 \cdot n_2 \) vertices and \( m_1 \cdot m_2 \) edges (this is like the product construction for automata).

**Theorem 4** For two OCD-indices \( b \) and \( c \), the function \( \text{Conj}(b, c) \) returns an index \( d \) such that \( P(d) = P(b) \land P(c) \) in time \( O(\text{size}(b) \cdot \text{size}(c)) \).

**Disjunction**

Disjunction is an easy operation for OCDs. Given two indices \( b \) and \( c \), the function \( \text{Disj}(b, c) \) computes an index \( d \) such that \( P(d) = P(b) \lor P(c) \). The function first checks if one of the indices
is a terminal index, and if both are nonterminal indices, then computes the union using \textit{Merge}.
Observe that disjunction operation does not create any new vertices. The time complexity of \textit{Disj}(b,c) is quadratic: $O(\text{size}(b) \cdot \text{size}(c))$. This is because \textit{Merge} needs to check for every $v \in b$ and $w \in c$, if $v \rightarrow w$ or $w \rightarrow v$ holds.

\textbf{Existential quantification}

Given an index \( b \) and a set \( Y \) of variables, the function \textit{Exists}(\( b,Y \)) returns an index \( c \) such that \( P(c) = \exists Y. P(b) \). The algorithm for \textit{Exists} is again an adaptation of the corresponding \textit{BDD} algorithm. The complexity of \textit{Exists}(\( b,Y \)) is $O(\text{size}(b))$.

\section{Application: Invariant Verification}

In this section, we consider the problem of verifying whether all reachable states of a finite-state model satisfy an invariant.

\textbf{Definition 5} \textit{A symbolic model} \( M \) \textit{consists of}

- A finite set \( X \) of boolean variables. The valuations for \( X \) are called \textit{states} of \( M \).
- An initial predicate \( q^I \) over \( X \). States satisfying \( q^I \) are called \textit{initial states}.
- A transition predicate \( q^T \) over \( X \cup X' \), where the set \( X' \) contains, for every variable \( x \in X \), its primed counterpart \( x' \). A state \( t \) is said to be a \textit{successor} of a state \( s \) if \( q^T \) evaluates to true when each variable \( x \) is assigned the value \( s(x) \) and each \( x' \) is assigned the value \( t(x) \).

For a sequence \( s_0s_1 \cdots s_n \) of states of \( M \) such that \( s_0 \) is initial, and \( s_{i+1} \) is a successor of \( s_i \) for all \( 0 \leq i < n \), the state \( s_n \) is said to be a reachable state of \( M \). In the invariant verification problem, we are given a predicate \( q^S \) that describes the set of \textit{safe} states, and we wish to determine if all reachable states satisfy \( q^S \), that is, if \( q^S \) is an invariant of \( M \). For symbolic verification, define an operation \textit{Post} as follows [BCD+92]. Given a predicate \( q \) over \( X \), \textit{Post}(\( q \)) is the predicate \( [\exists X. (q \land q^T)][X' := X] \) obtained by first taking the conjunction of \( q \) with the transition predicate \( q^T \), then eliminating the variables in \( X \) using existential quantification, and then renaming each primed variable \( x' \) to \( x \). Verify that \textit{Post}(\( q \)) describes the set of successors of states satisfying \( q \). Then, the predicate \( q^S \) is an invariant of \( M \) iff \( \textit{Post}^*(q^I) \land \neg q^S \) is unsatisfiable.

The outline of the algorithm for symbolic verification using ordered choice diagrams is shown in Figure 5. We first choose an ordering \(< \) over the set \( X \cup X' \). The ordering satisfies the property that if \( x < y \) for two variables \( x \) and \( y \) in \( X \), then \( x' < y' \). The next step is to build the \textit{OCD} representations of the initial predicate \( q^I \), the transition predicate \( q^T \), and the predicate \( \neg q^S \) describing the set of “unsafe” states. The set of reachable states is computed iteratively. In each iteration the algorithm first checks if the intersection of the current reachable set and the unsafe set is nonempty. If so, \( q^S \) is not an invariant of the input model. Note that the test for satisfiability is implemented in constant time. Otherwise, all current states are safe, and the successors of the current reachable set are computed using \textit{Post}. If \( P(b) = P(c) \) then we can conclude that \( P(b) \) contains all the reachable states, and hence, \( q^S \) is an invariant of \( M \). However, testing equivalence of two indices is computationally hard. Hence, the algorithm settles for a sufficient, but not necessary, test of checking whether \( b \) and \( c \) are identical indices. Since it may happen that \( P(b) \) and \( P(c) \) are equivalent while \( b \) and \( c \) are distinct, the algorithm executes further iterations. However, the number of distinct equivalent indices in an \textit{OCD}-space is finite, the algorithm is guaranteed to terminate.
Input: a symbolic model $M = (X, q^I, q^T)$ and a predicate $q^S$
Output: answer to the invariant verification question $(M, q^S)$

Choose a variable ordering $<$ over $X \cup X'$
Construct OCD-indices $b^I, b^T,$ and $b^U$ for $q^I, q^T,$ and $-q^S,$ resp.
$c := b^I$
repeat
\begin{align*}
 b &:= c \\
 \textbf{if} \ Conj(b, b^T) \neq 0 \textbf{ then return } \text{No} \\
 c &:= Disj(b, Post(b)) \\
 \textbf{until} \ b = c \\
 \textbf{return} \ \text{YES}.
\end{align*}

Figure 5: Algorithm for symbolic invariant verification

5 Implementation

In this section, we report a prototype implementation of OCDs in Lucid Common Lisp. The objective is to determine if the OCD-based representation leads to space-savings during symbolic invariant verification.

Our implementation uses multi-valued choice diagrams. That is, instead of boolean variables, we allow variables over finite, enumerated, domains. Consequently, instead of the left- and right-child functions $C_l$ and $C_r$, a vertex labeled with variable $x$ has a child $C_v$ for every possible value $v$ of $x$. All the algorithms reported earlier generalize to this case in a straightforward manner.

Recall that to take disjunction of two indices $b$ and $c$, it suffices to merge the two lists. However, initial experiments indicated that this approach leads to an unacceptable proliferation due to the extensive use of existential quantification in computation of $Post$. Consequently, we use the following routines to modify generation of OCD-indices.

$VertexDisj$: Given two vertices $v$ and $w$, $VertexDisj$ computes a new vertex $u$ such that $P(u) = P(v) \lor P(w)$. The algorithm for $VertexDisj$ is similar to the algorithm for the conjunction of OCDs, and if the two input vertices $v$ and $w$ represent BDDs, then the output of $VertexDisj$ is, in fact, a BDD.

$Optimize$: Given a nonterminal index $b = \{v_1, \ldots, v_k\}$, the function $Optimize$ creates, for each $1 \leq i < k$, another vertex $u_i$ representing the disjunction of $v_i$ and $v_{i+1}$ using $VertexDisj$. If $\text{size}(u_i)$ exceeds $\text{size}(\{v_i, v_{i+1}\})$ for some $i$, it replaces $v_i$ and $v_{i+1}$ in $b$ with $u_i$, and calls itself recursively on the resulting index. Otherwise, it returns the input index $b$.

Intuitively, whenever replacing a pair of adjacent vertices by a vertex representing their disjunction leads to reduction in size, we do it.

Experimental results

We have tested our implementation on some small examples of distributed protocols among asynchronous processes. Such protocols are usually modeled using the interleaving assumption by which one of the enabled transitions is executed at every step. In such systems, the transition relation is naturally partitioned in a disjunctive manner, and consequently, such protocols are a natural candidate for OCD-based representation. For each example, we determined the size of the transition
relation using OCD as well as BDD representation, and the sizes of the state-sets generated during the fixpoint computation using the two representations.

Our first example is the classical two-process mutual exclusion protocol due to Peterson. As seen from Figure 5, the OCD representation of the transition relation is a few nodes smaller. However, during the computation of successive approximation of the reachable set, the two approaches behave identically.

The second example concerns a simplified variant of the leader election protocol for \( n \) processes connected in a two-way ring. Each process \( i \) has a variable \( x_i \) that can take \( k \) values. Initially, the values of \( x_i \) variables are arbitrary. In each step, some process \( i \) examines the value of one of its neighbors processes \( i+1 \) or \( i-1 \) (module \( n \)), and updates \( x_i \) to the maximum of \( x_i \) and the examined value. The protocol ensures that eventually all the variables will have the same value, namely, the maximum of initial values. The choice of variable ordering in this example is any cyclic ordering consistent with the ring structure. The reported result is for \( n = 4 \) and \( k = 10 \). Again, the OCD-representation affords reduction only in the size of the transition relation. The two approaches agree on the number of iterations and the peak size of reachable set. However, the initial growth is significantly slower for the OCD-based approach (e.g., the successor-set of the initial region has 172 nodes in OCD representation and 511 nodes in BDD representation).

Finally, let us modify the example so that all processes are connected with each other. Then, in a single step, two processes \( i \) and \( j \) such that their current values are different, exchange their values, and both \( x_i \) and \( x_j \) are updated to the maximum of the two. The table shows that OCD representation reduces the peak size without affecting the number of iterations. In particular, the BDD representation after first step is ten times larger than the OCD representation. This can be supported by a theoretical analysis also. For instance, the state-set \( \sigma_1 \) containing states reachable in one step corresponds to the predicate \( V_{i\neq j} x_i = x_j \). There is no preferred variable-ordering for such a symmetric function. Assuming \( k > n \), it can be shown that while the OCD representation of \( V_{i\neq j} x_i = x_j \) is polynomial (bounded by \( kn^2 \)), the BDD representation is exponential \( (\Sigma_{m=0}^{n} k!/(m!(k-m)!)) \).

In the current implementation, BDD sizes were computed in a round-about way, and hence, we cannot report any meaningful comparison of running time requirements of the two approaches (this will be done in the full paper).

### 6 Conclusions

We have proposed to introduce nondeterminism in BDD representation. As a result, the canonicity is lost, but can lead to significant succinctness. We have shown how to do symbolic model checking using OCD representation. Initial experimental results indicate that this may be a fruitful approach for distributed protocols modeled by interleaving of process actions. Much work remains to be
done. In particular, in the full paper, we will report more detailed experimental results comparing BDD-based and OCD-based analysis. An important research direction concerns finding improved heuristics to determine when two vertices in an index can be replaced by a vertex representing their disjunction.

**Acknowledgements.** The author thanks Tom Henzinger, Shaz Qadeer, Sriram Rajamani, and Rajeev Ranjan for fruitful discussions.

**References**


