Compressed Data Structures for Query Processing

ROBDDs and Variable Ordering

Anna Bolotina

University of Salzburg

Advanced Topics in Data Analytics
Winter semester 2026/27

Lecture outline

Before the break

- How can a decision tree check whether a record is both Austrian and Premium?
- Once we know the country, what is left to check?
- When can we reuse part of the graph or skip a test?
- With a fixed variable order, do we always get the same reduced graph?

After the break

- When comparing two bit vectors, why can checking each pair of corresponding bits together save nodes?
- How can we combine two existing diagrams to compute AND?
- In Assignment 2, you will try different variable orders and count the nodes in each graph.

Austria AND Premium as a Boolean function

For a single record, write

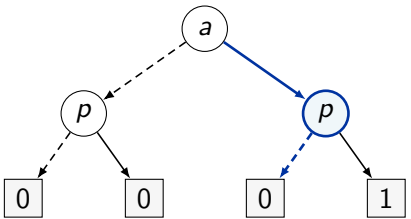
$$a = \text{[country is Austria]}, \qquad p = \text{[plan is Premium]}, \qquad f(a, p) = a \wedge p.$$

Record	a	p	$f(a, p)$
1 (Austria, Premium)	1	1	1
3 (Austria, Basic)	1	0	0
4 (Italy, Premium)	0	1	0

For each record, $f(a, p)$ gives its bit in the query result.

Reading a decision tree

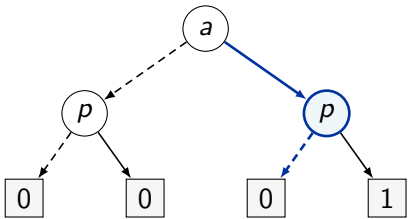
a	p	f
0	0	0
0	1	0
1	0	0
1	1	1



Record 3

Take the 1 branch at a, then the 0 branch at p. The leaf is 0.

Fixing a variable gives a restriction



Low branch ($a = 0$)

$f|_{a=0}(p) = 0$ for either value of p.

High branch ($a = 1$)

$f|_{a=1}(p) = p$.

Shannon expansion

$$f = (\neg a \wedge \underbrace{f|_{a=0}}_{\text{low successor}}) \vee (a \wedge \underbrace{f|_{a=1}}_{\text{high successor}}).$$

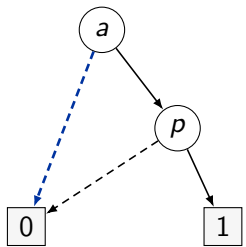
For Austria AND Premium this gives

$$f(a, p) = (\neg a \wedge 0) \vee (a \wedge p) = a \wedge p.$$

One term per branch

When $a = 0$, only the low term can contribute. When $a = 1$, only the high term can contribute.

An ordered binary decision diagram

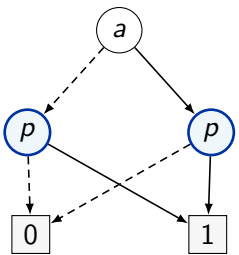


Dashed: low (0). Solid: high (1).

- A decision node tests a variable and has a low and a high successor.
- A terminal supplies the output bit.
- Along every path, tests follow the same order. Here $a \prec p$.

If $a = 0$, the path reaches 0 immediately and skips p.

Reduction rule 1: merge identical subgraphs

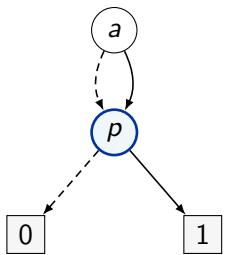


Dashed: low (0). Solid: high (1).

Merge nodes with the same

- variable,
- low successor,
- high successor.

Reduction rule 1: merge identical subgraphs



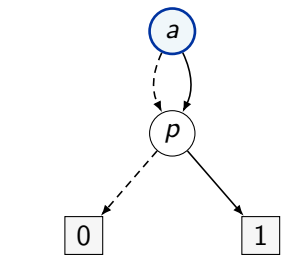
Dashed: low (0). Solid: high (1).

Merge nodes with the same

- variable,
- low successor,
- high successor.

Redirect both incoming edges to the shared p node.

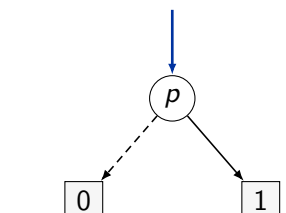
Reduction rule 2: remove a redundant test



Dashed: low (0). Solid: high (1).

If low and high point to the same successor, bypass the decision node.

Reduction rule 2: remove a redundant test

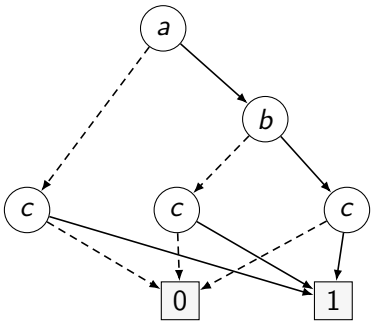


Dashed: low (0). Solid: high (1).

If low and high point to the same successor, bypass the decision node.

Here the root becomes p. The function is independent of a.

Exercise: simplify the ordered graph

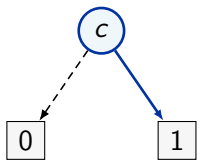


Dashed: low (0). Solid: high (1).

Order $a \prec b \prec c$.

1. Which nodes can share a representative?
2. Which tests can you remove after merging?
3. What does $(a, b, c) = (1, 0, 1)$ return?

Solution: the function is c



Dashed: low (0). Solid: high (1).

1. Merge the three c nodes.
2. Bypass b: both edges lead to c.
3. Bypass a: both edges now lead to c.

For $(1, 0, 1)$, skip a and b, then follow the high edge at c to 1.

Why a fixed order gives a canonical graph

Reduced ordered binary decision diagram (ROBDD)

For a fixed variable order, each Boolean function has a unique ROBDD up to graph isomorphism.

- A constant function is a single terminal.
- For any other function, the root tests the earliest variable the function depends on.
- Its two successors represent the two restrictions. Apply the same argument to each restriction and share identical subgraphs.

R. E. Bryant (1986), Graph-Based Algorithms for Boolean Function Manipulation.

Break

10 minutes

Next: two orders for bit-vector equality.

Equality of two bit vectors

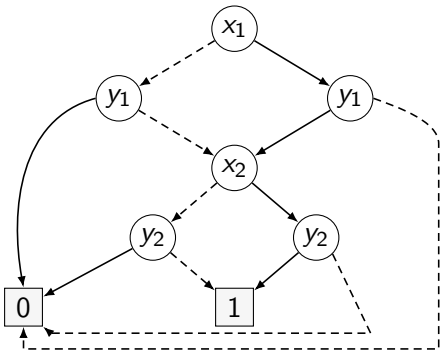
$$\mathrm{EQ}_n(x, y) = \bigwedge_{i=1}^{n} (x_i \leftrightarrow y_i).$$

The result is 1 exactly when all corresponding bits match.

Interleaved	$x_1, y_1, x_2, y_2, \ldots, x_n, y_n$
Grouped	$x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$

Which order will need fewer decision nodes? When can a path first detect a mismatch?

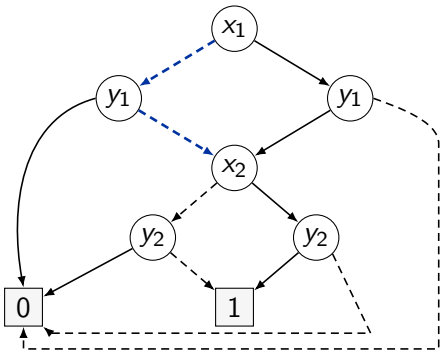
Interleaved order checks the first pair immediately



Dashed: low (0). Solid: high (1).

Order x_1, y_1, x_2, y_2.

Interleaved order checks the first pair immediately

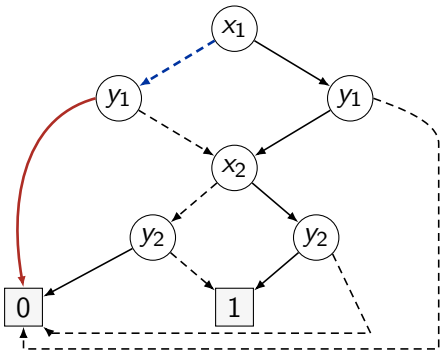


Dashed: low (0). Solid: high (1).

Order x_1, y_1, x_2, y_2.

- If $x_1 = y_1 = 0$, continue at x_2.

Interleaved order checks the first pair immediately

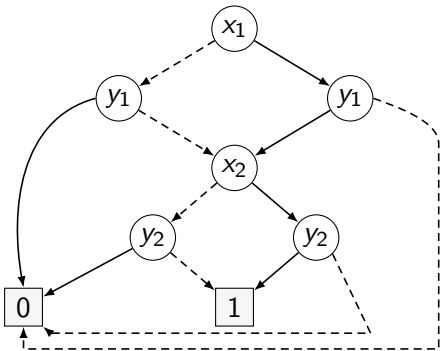


Dashed: low (0). Solid: high (1).

Order x_1, y_1, x_2, y_2.

- If $x_1 = y_1 = 0$, continue at x_2.
- If $x_1 = 0$ and $y_1 = 1$, reach 0 immediately.

Matched pairs share the same continuation

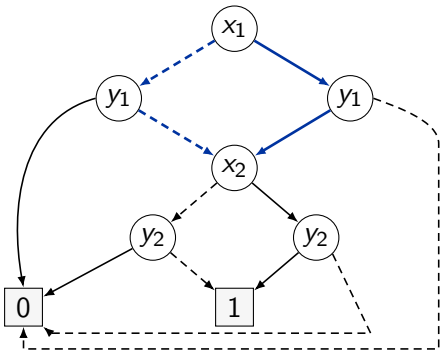


Dashed: low (0). Solid: high (1).

Both $(x_1, y_1) = (0, 0)$ and $(1, 1)$ lead to the same x_2 node.

Variable	x_1	y_1	x_2	y_2
Nodes	1	2	1	2

Matched pairs share the same continuation



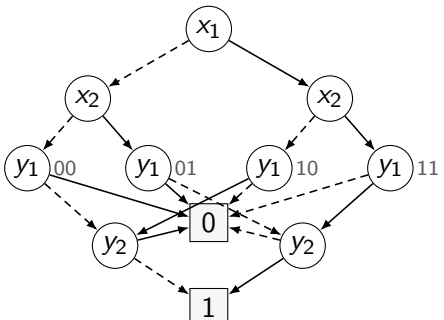
Dashed: low (0). Solid: high (1).

Both $(x_1, y_1) = (0, 0)$ and $(1, 1)$ lead to the same x_2 node.

Variable	x_1	y_1	x_2	y_2
Nodes	1	2	1	2

6 reachable nonterminal nodes.

Grouped order remembers the first vector

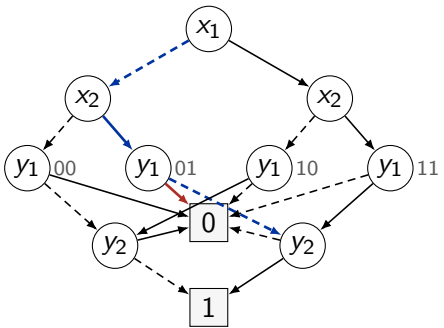


Dashed: low (0). Solid: high (1).

Order x_1, x_2, y_1, y_2.

Each label beside a y_1 node shows the x prefix that reaches it.

Grouped order remembers the first vector



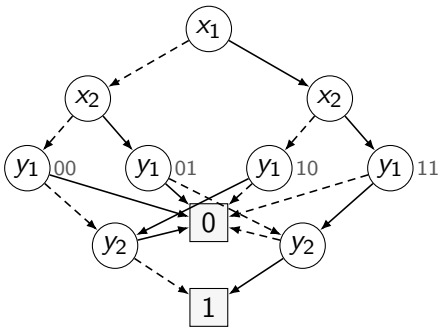
Dashed: low (0). Solid: high (1).

Order x_1, x_2, y_1, y_2.

Each label beside a y_1 node shows the x prefix that reaches it.

For $x = 01$, the remaining test accepts only $y = 01$. A high edge at y_1 takes this path to 0.

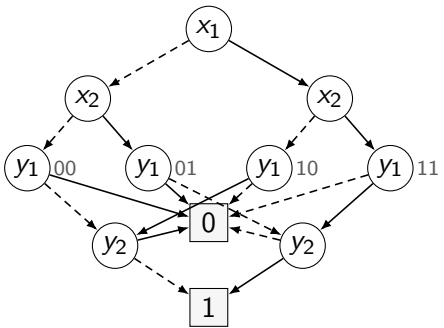
Four distinct tests of the second vector



Dashed: low (0). Solid: high (1).

x read	Remaining test
00	$\neg y_1 \wedge \neg y_2$
01	$\neg y_1 \wedge y_2$
10	$y_1 \wedge \neg y_2$
11	$y_1 \wedge y_2$

Four distinct tests of the second vector



Dashed: low (0). Solid: high (1).

x read	Remaining test
00	$\neg y_1 \wedge \neg y_2$
01	$\neg y_1 \wedge y_2$
10	$y_1 \wedge \neg y_2$
11	$y_1 \wedge y_2$

$1 + 2 + 4 + 2 = \mathbf{9}$ reachable nonterminal nodes.

Interleaving uses three nodes per pair

Before each pair, all earlier matched pairs share one remaining equality test.

- One x_i node starts the pair.
- Two y_i nodes remember whether x_i was 0 or 1.
- A mismatch goes to 0. A match continues at the next pair, or reaches 1 after the last pair.

Interleaved size

$3n$ reachable nonterminal nodes in the ordinary ROBDD.

Grouping leaves 2^n distinct residual functions

After all x bits have been read, each vector $u \in \{0,1\}^n$ leaves the residual test $[y = u]$.

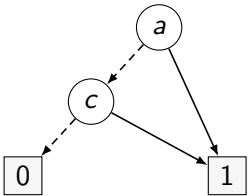
- For two distinct prefixes u, v, the input $y = u$ distinguishes their residual functions.
- Each residual requires a distinct root at y_1. This level alone contains 2^n nodes.

What is being ordered?

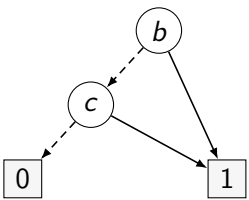
In a BDD, the order specifies the sequence of tests on an input. In the bitmap examples, record order specifies which record occupies each bit position.

Apply combines two diagrams with AND

$f = a \vee c$



$g = b \vee c$



Dashed: low (0). Solid: high (1).

Both diagrams follow $a \prec b \prec c$. What should the diagram for $h = f \wedge g$ compute?

Split on the earliest root variable

The current roots test a and b, so Apply splits on a. The root of g stays at b on both branches.

Branch	f restricted	g restricted	AND
$a = 0$	c	$b \vee c$	$c \wedge (b \vee c)$
$a = 1$	1	$b \vee c$	$b \vee c$

High successor

$1 \wedge g = g$, so reuse the root of g.

The low branch reduces to c

For $a = 0$, the current pair is $(c, b \vee c)$. Split on b.

Branch	First operand	Second operand	AND
$b = 0$	c	c	c
$b = 1$	c	1	c



The low branch reduces to c

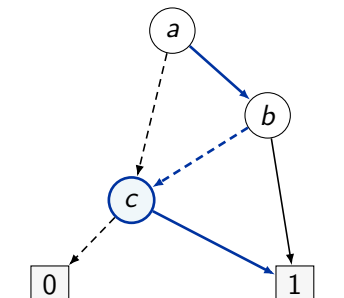
For $a = 0$, the current pair is $(c, b \vee c)$. Split on b.

Branch	First operand	Second operand	AND
$b = 0$	c	c	c
$b = 1$	c	1	c



Both successors are the same c subgraph.
Bypass the b node.

The result shares the c subgraph



Dashed: low (0). Solid: high (1).

For $(a, b, c) = (1, 0, 1)$, follow high, low, high to 1. Both operands are true.

Memoization

During Apply, cache the result of each processed pair of operand nodes and reuse it when that pair recurs.

Assignment 2: Variable Ordering Explorer

How does changing the variable order affect the size of an ROBDD for EQ_n?

- Compare interleaved, grouped and seeded random orders.
- Predict how the size curves change as n increases.
- Use parity of the same $2n$ bits as a control. Predict what changes when you permute its variables.

Parity on the same inputs ($n = 2$)

Parity is 1 for an odd number of ones.

x_1	x_2	y_1	y_2	Number of ones	Parity
0	1	1	1	3	1
0	1	1	0	2	0

Before running the experiment

Write a hypothesis and explain which residual functions can share nodes.

Assignment handout · Setup instructions and worked example

Measuring size under a fixed order

Size metric

Count the distinct reachable nonterminal nodes from the result root.

1. Use the same library and representation throughout.
2. Set the order explicitly and disable automatic reordering.
3. Verify and record the effective order for every run.
4. Check all 16 assignments of EQ_2 before measuring larger inputs.

The assignment uses `dd.autoref` 0.6.0. Complemented references let it share nodes that are separate in the diagrams shown in class.

What to include in the report

- Hypotheses, functions and input sizes.
- Effective orders, seeds, library and version.
- Correctness check and a labelled plot or table of node counts.
- Explanation of the results through subgraph sharing.

Report due: 24 November at 23:59.
Presentations: 1 December.

Next session

Restricting and projecting Boolean relations with BDDs.
Complemented edges and multi-terminal diagrams for matrices.