Sorting Parity Encodings by Reusing Variables

Leroy Chew and Marijn J.H. Heule

Carnegie Mellon University

23rd International Conference on Theory and Applications of Satisfiability Testing July 7, 2020

Problem Statement

Suppose we have a permutation σ on $[n]$. The two xor constraints:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0 \quad \wedge \quad x_{\sigma(1)} \oplus x_{\sigma(2)} \oplus \cdots \oplus x_{\sigma(n)} = 1$$

are a contradiction.

Tseitin variables t_i and s_i represent the reordered parity:

$$\begin{array}{ll} t_1 = x_1 \oplus x_2 & s_1 = x_{\sigma(1)} \oplus x_{\sigma(2)} \\ t_{i-1} = t_{i-2} \oplus x_i & s_{i-1} = s_{\sigma(i-2)} \oplus x_{\sigma(i)} \\ t_{n-3} \oplus x_{n-1} \oplus \bar{x}_n & s_{\sigma(n-3)} \oplus x_{\sigma(n-1)} \oplus x_{\sigma(n)} \end{array} \qquad \text{for } i = 3 \text{ to } n-2$$

We can represent $a = b \oplus c$ in four clauses.

$$\bar{a} \vee b \vee c \qquad a \vee \bar{b} \vee c \qquad a \vee b \vee \bar{c} \qquad \bar{a} \vee \bar{b} \vee \bar{c}$$

An Easy Special Case

- A special case when the permutation is the identity map:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0 \quad \wedge \quad x_1 \oplus x_2 \oplus \cdots \oplus x_n = 1.$$

- These formulas are instances of the `Dubois` family and have short linear resolution proofs.
- However when the reordered parity is done randomly, we do not know if there are short proofs for resolution.

SAT Solvers and Gaussian Elimination

- Appear to show exponential running times on CDCL algorithm.
- However there is at least one method that deals with reordered parity quickly: Gaussian elimination.
- Gaussian elimination requires the detection of xor constraints and then adds them together to create more xor constraints until an inconsistent constraint is found.
- However solvers that support Gaussian elimination cannot produce proofs.

The DRAT proof system

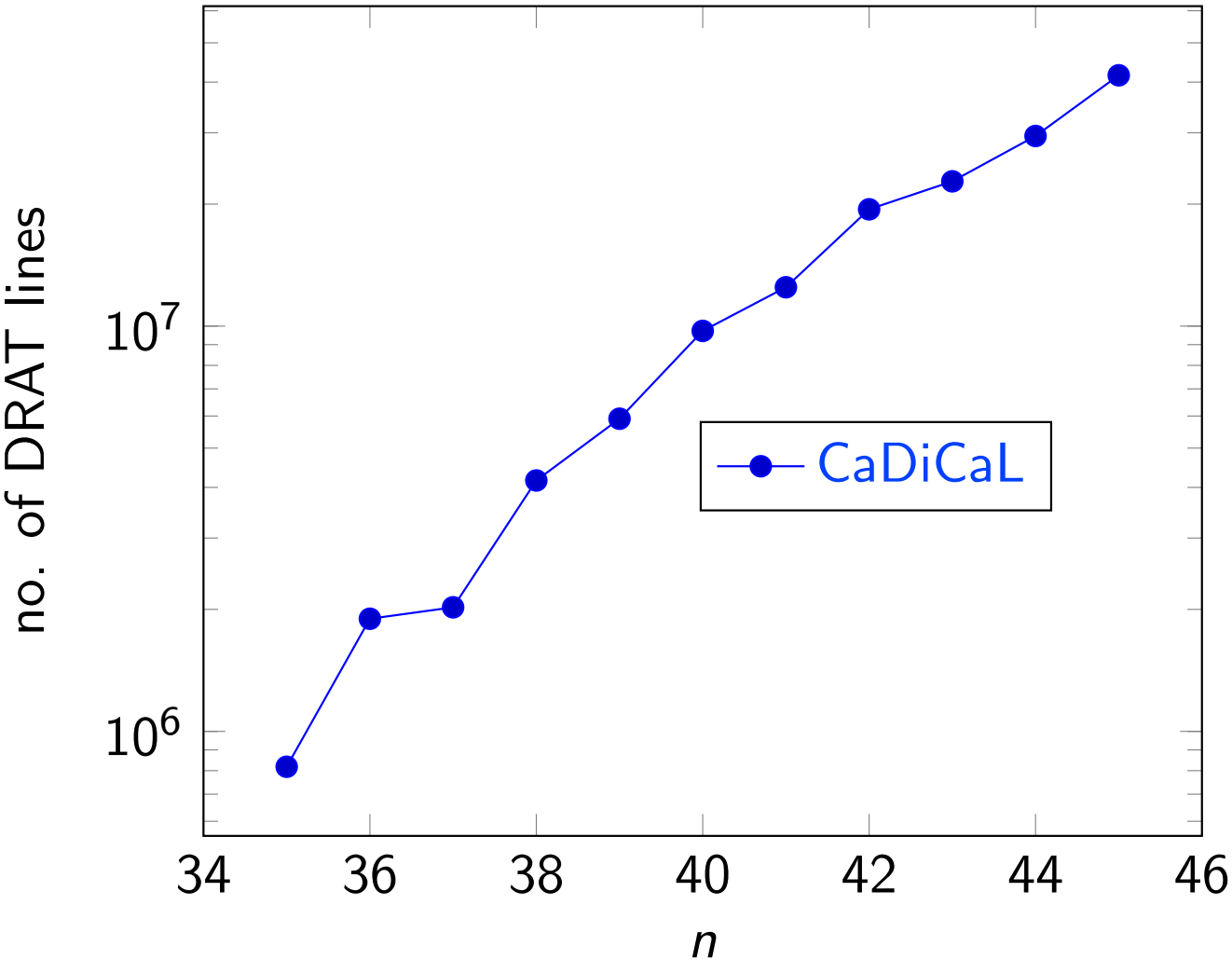
- Powerful proof system, simulates strong systems like Frege, Extended resolution (ER) and Propagation Redundancy (PR).
- Standard proof format in SAT solving.
- Every ER proof is automatically a DRAT proof.
- DRAT$^-$ is without new variables [Buss, Thapen 2019]

DRAT Rules:

- Logical implication rules (ATA, ATE)
- Satisfiability preserving rules (RATA, RATE)

Generated Proofs

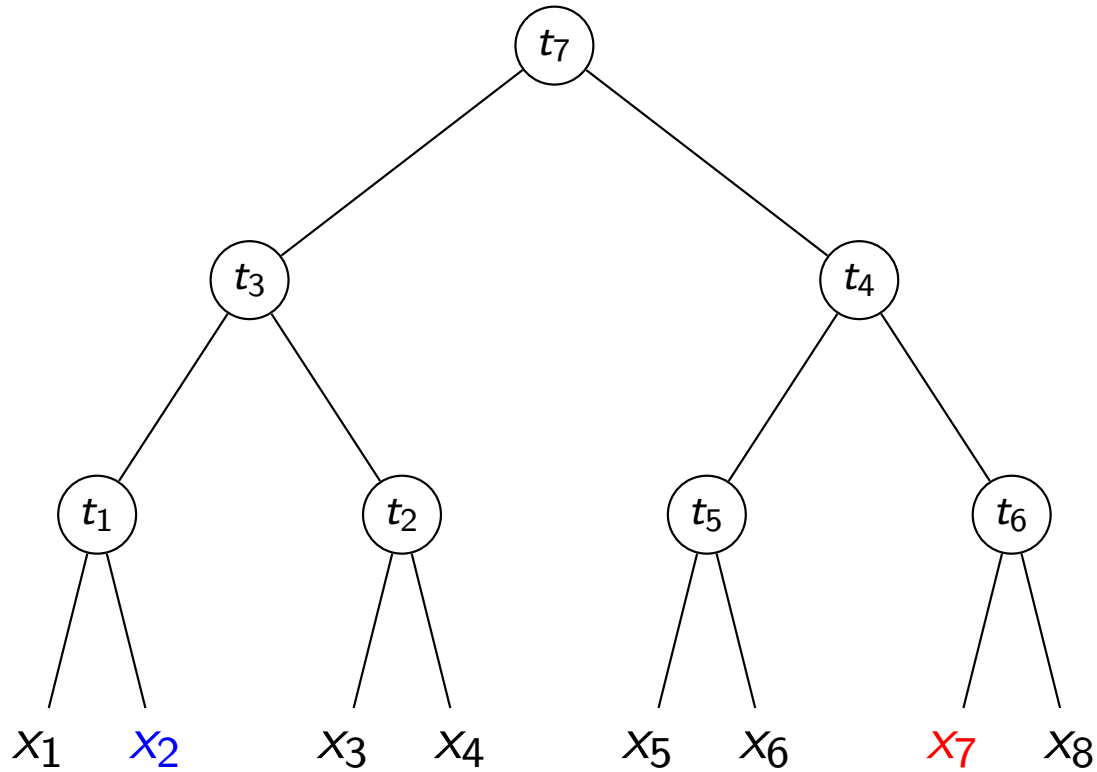
CDCL on reordered parity formulas appear exponential.



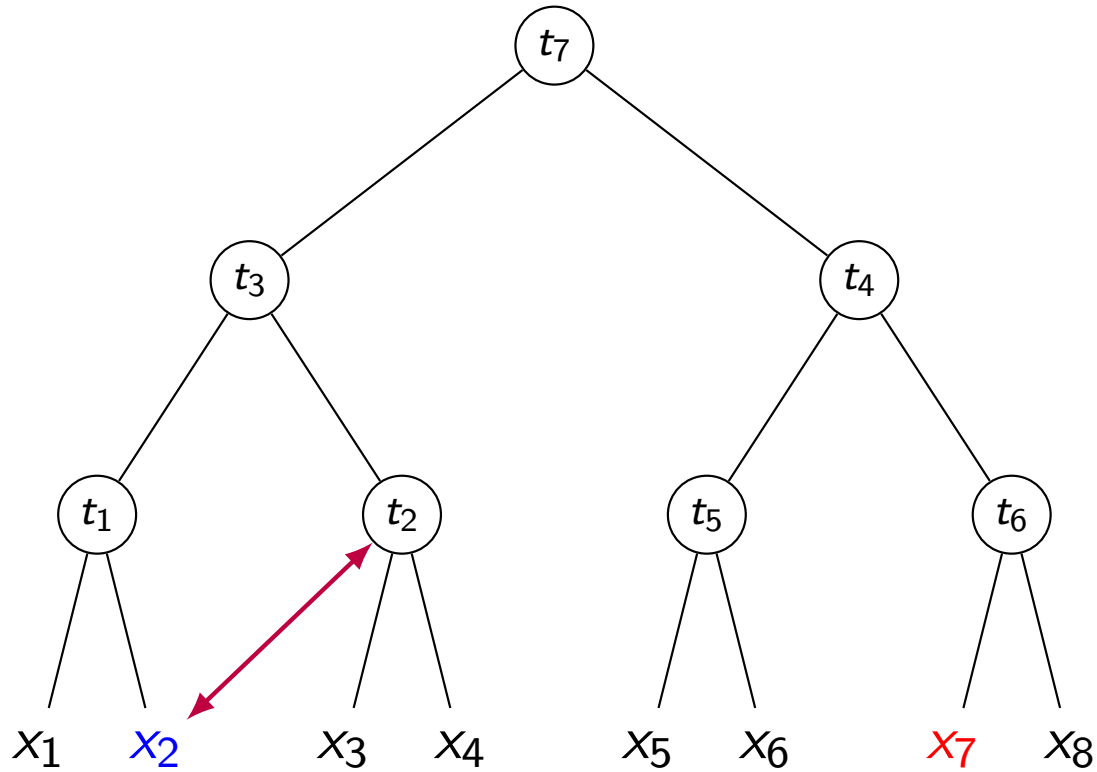
Solving Unordered Parity by Sorting

- We can use swaps to turn $x_{\sigma(1)} \oplus x_{\sigma(2)} \oplus \cdots \oplus x_{\sigma(n)} = 1$ into $x_1 \oplus x_2 \oplus \cdots \oplus x_n = 1$ (in Tseitinised form).
- Swaps must have short DRAT proofs.
- We limit swaps to adjacent variables, those used in the next Tseitin variable.
- The Tseitin variables are in a linear depth tree: hence $O(n^2)$ in the worst case.
- For a log depth tree this is $O(n \log n)$.

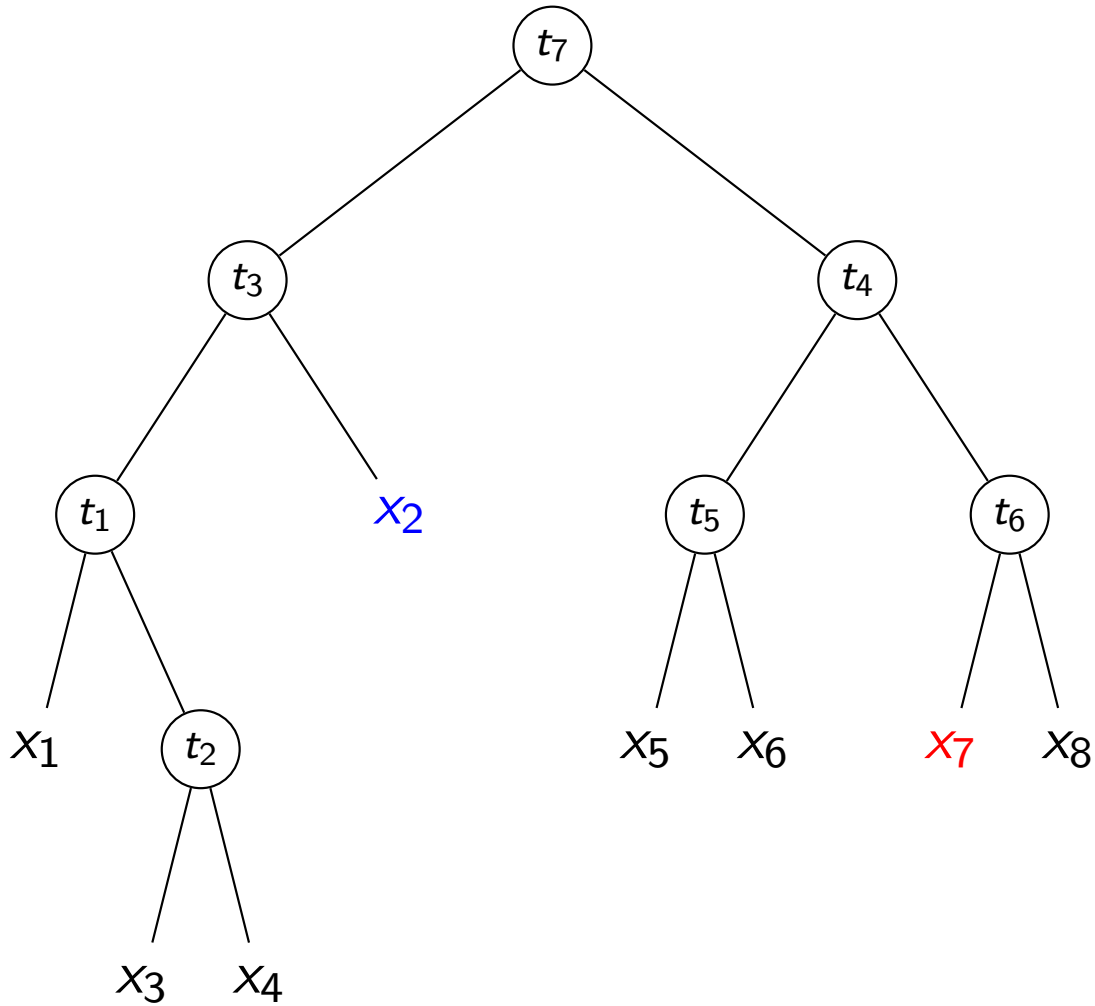
Sorting a Log Depth Tree



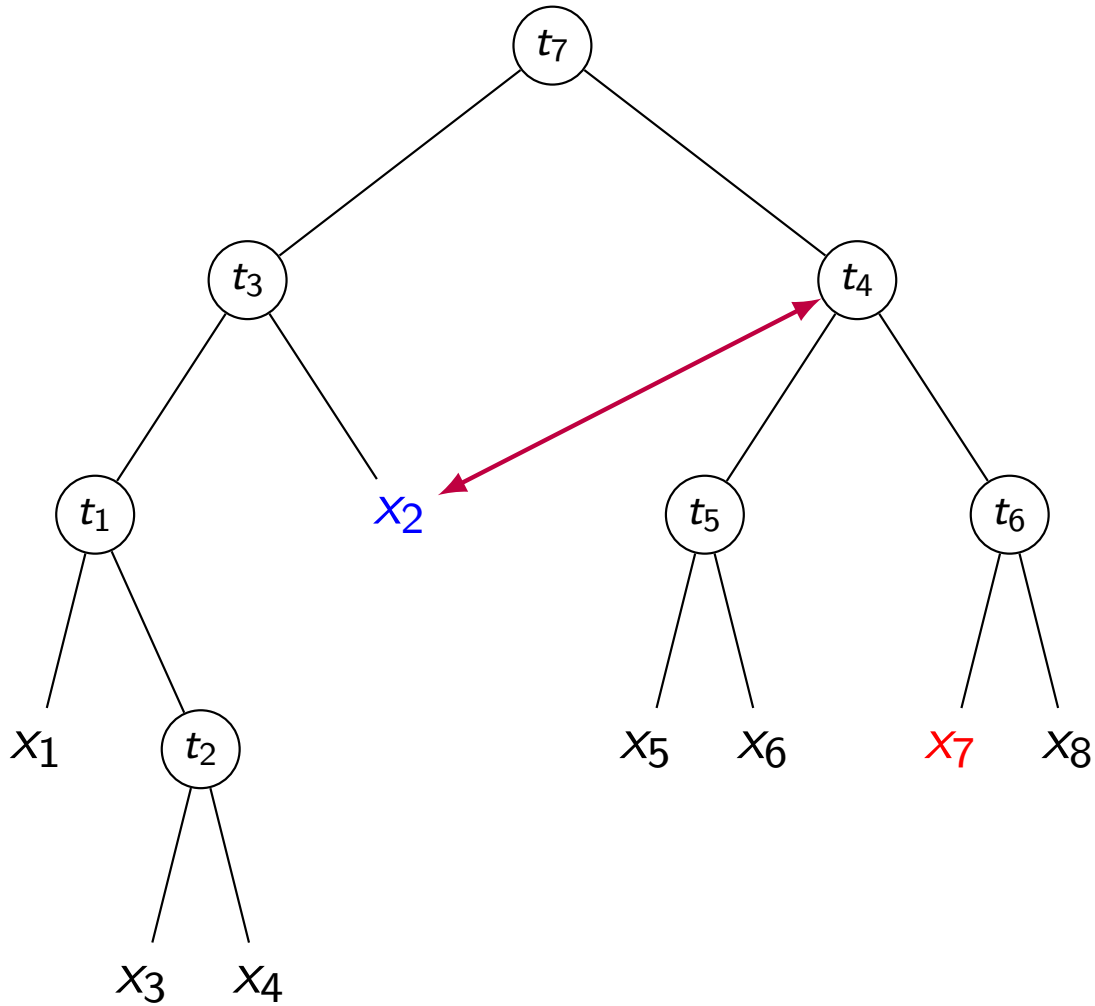
Sorting a Log Depth Tree



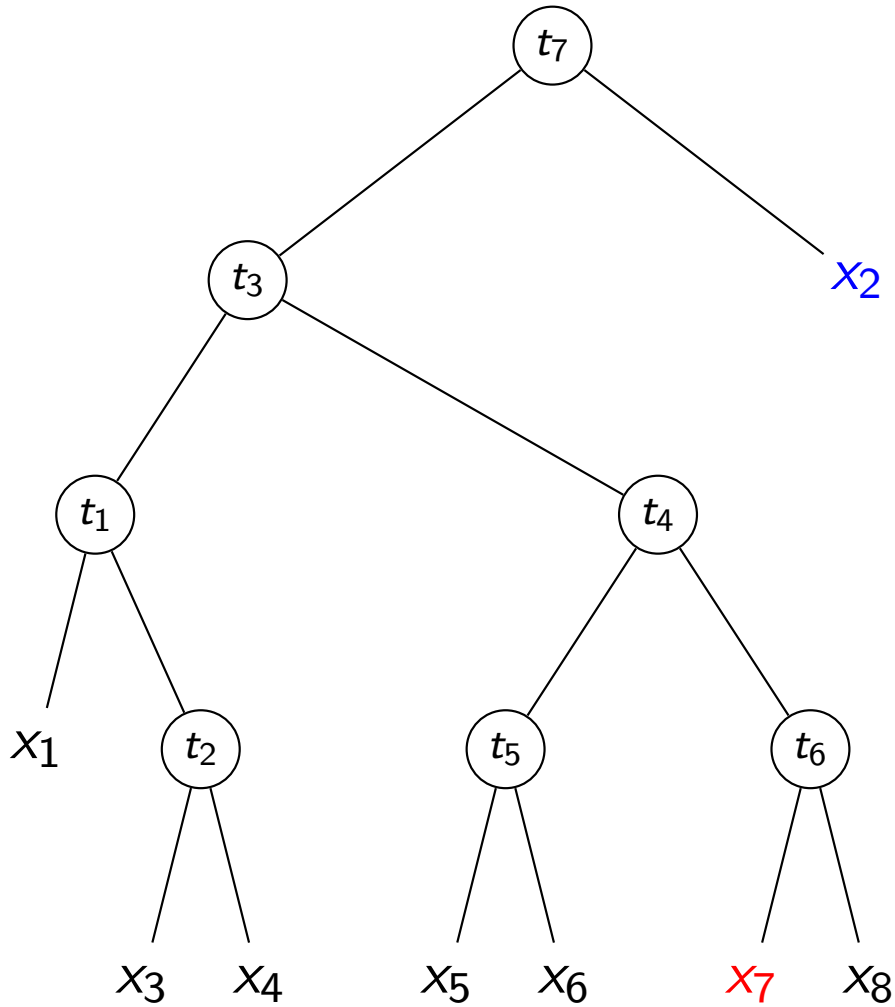
Sorting a Log Depth Tree



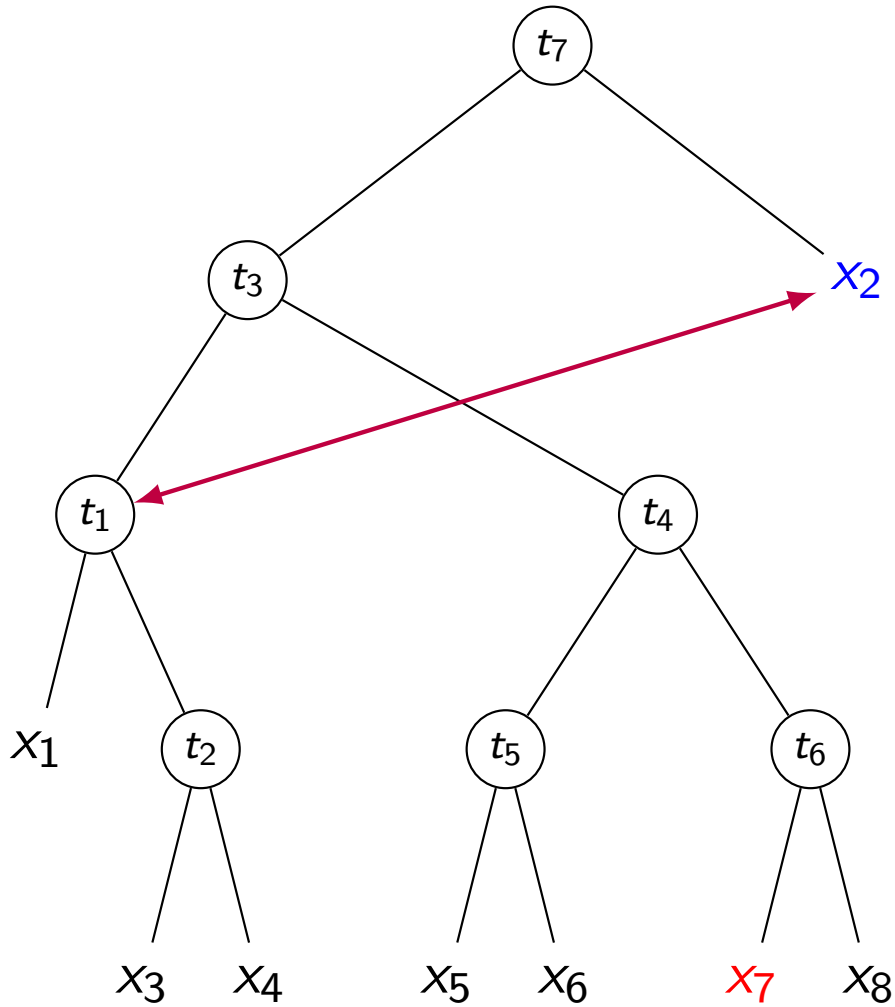
Sorting a Log Depth Tree



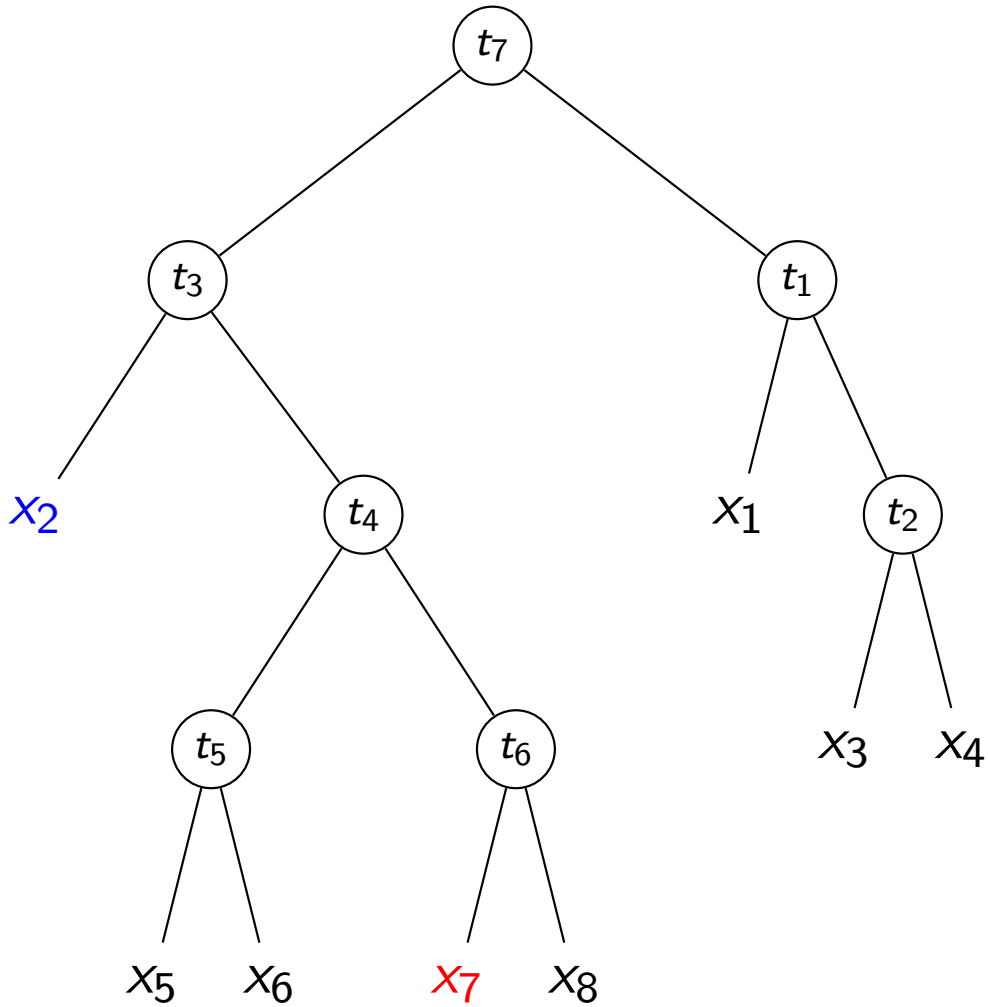
Sorting a Log Depth Tree



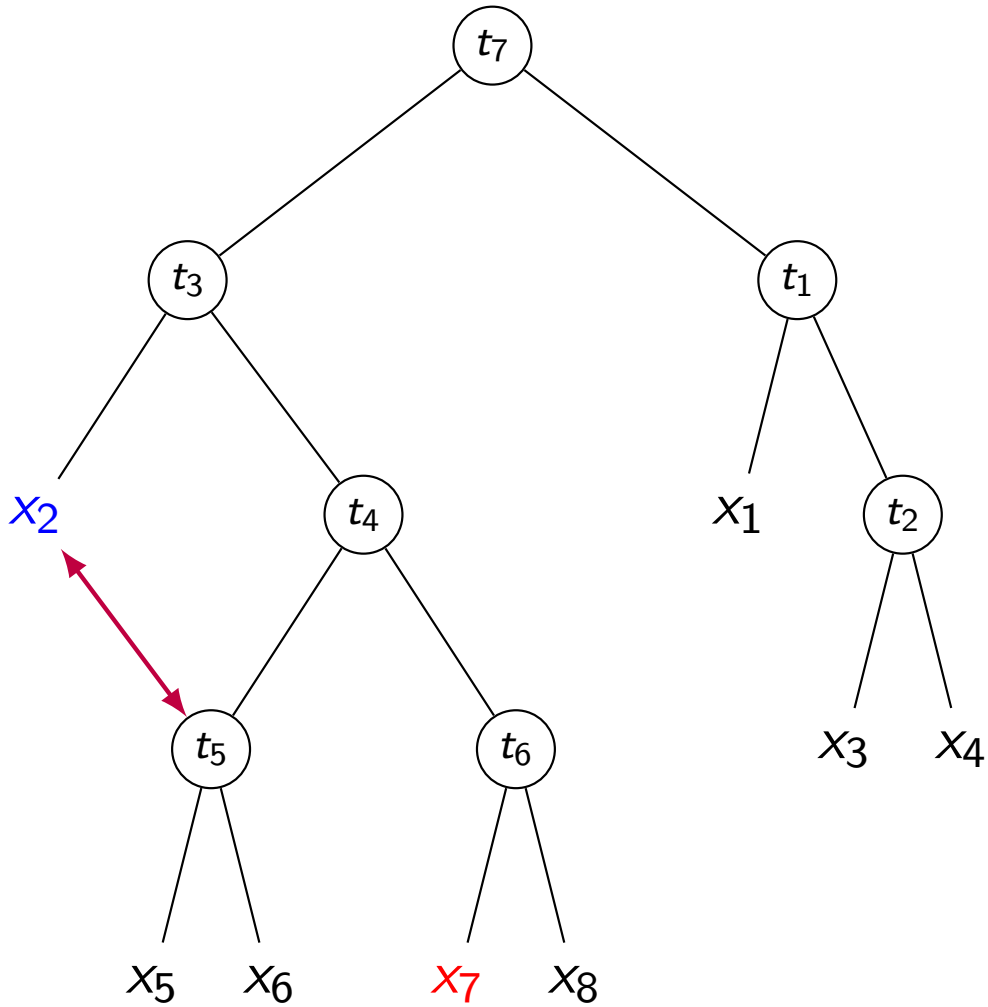
Sorting a Log Depth Tree



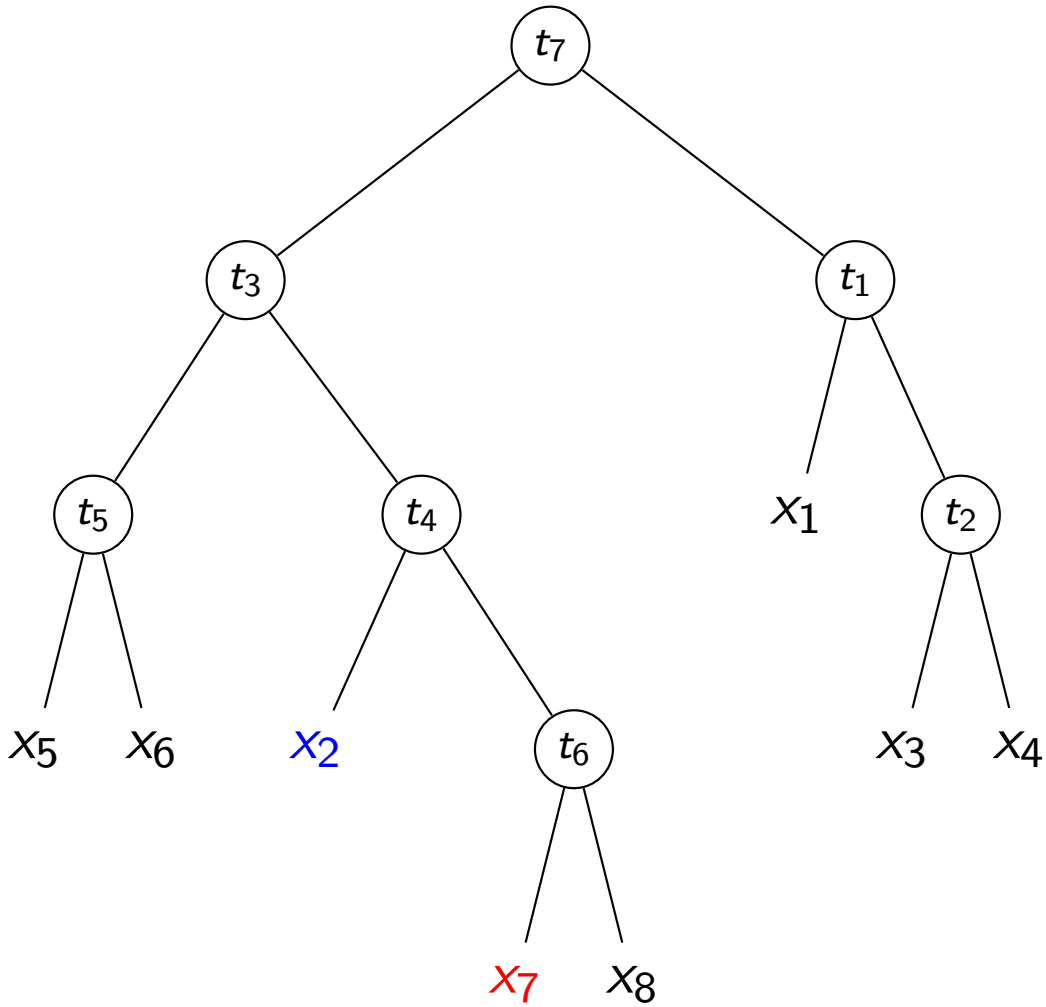
Sorting a Log Depth Tree



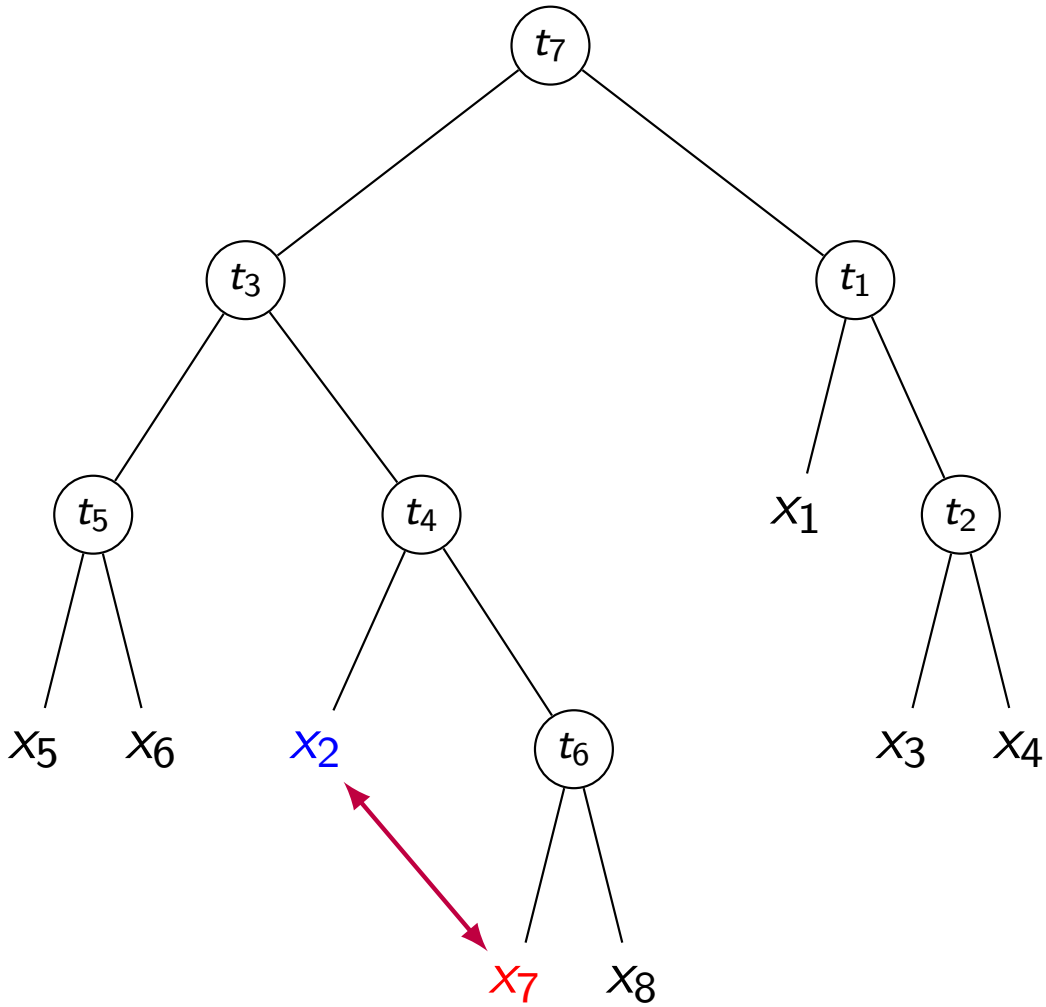
Sorting a Log Depth Tree



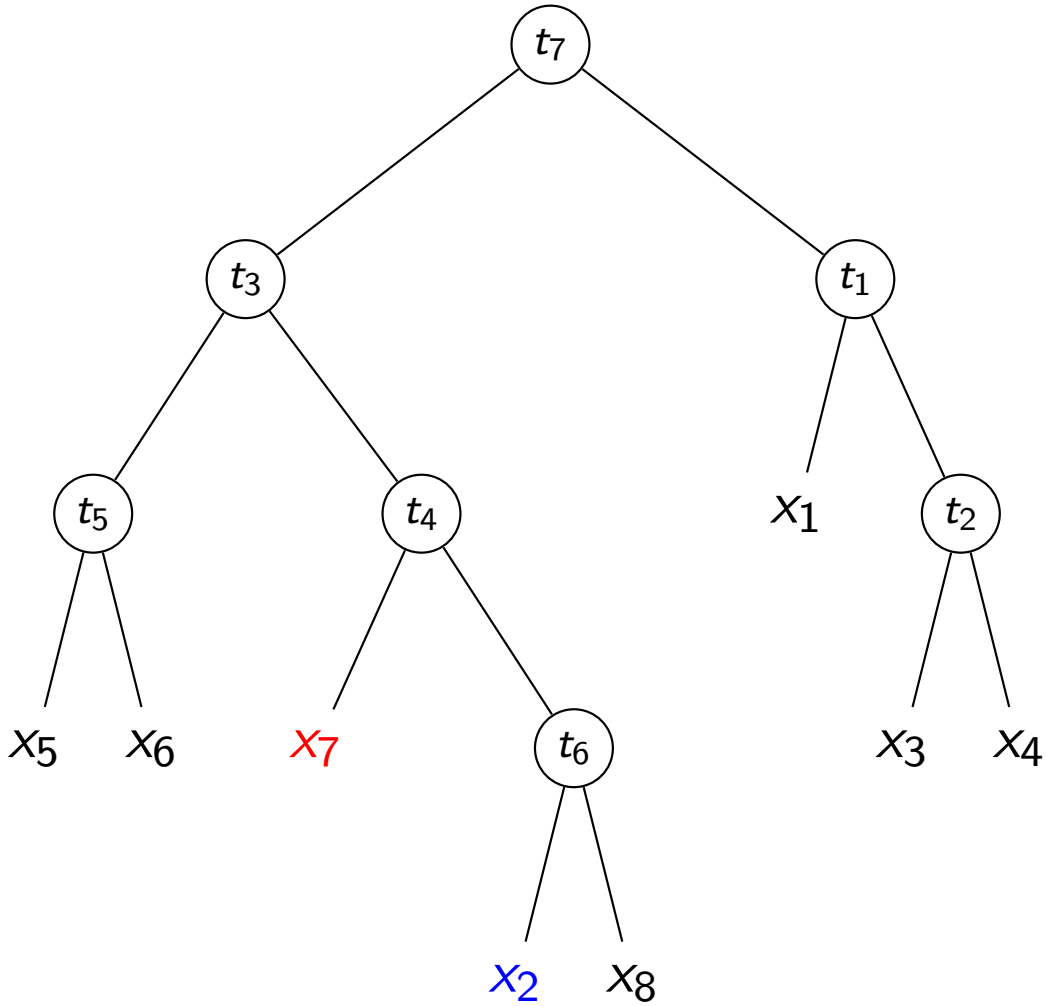
Sorting a Log Depth Tree



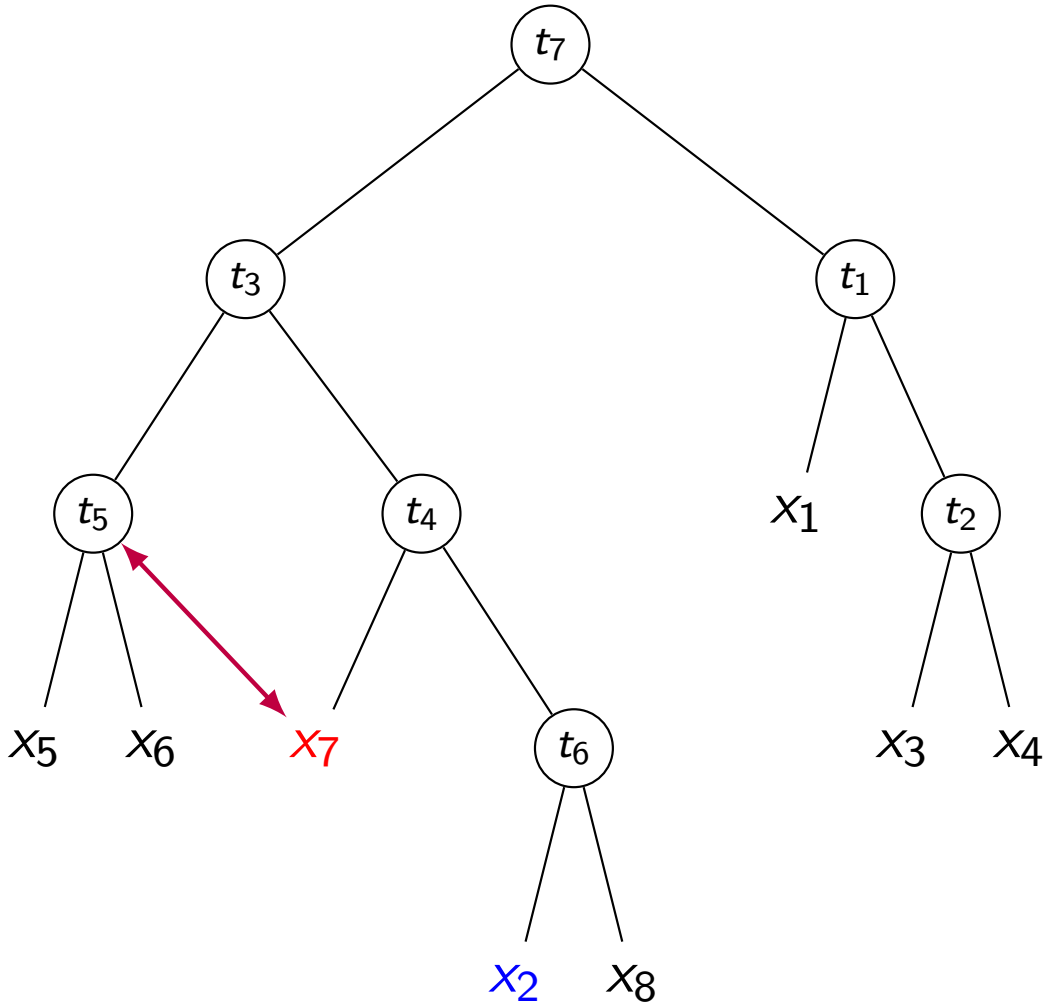
Sorting a Log Depth Tree



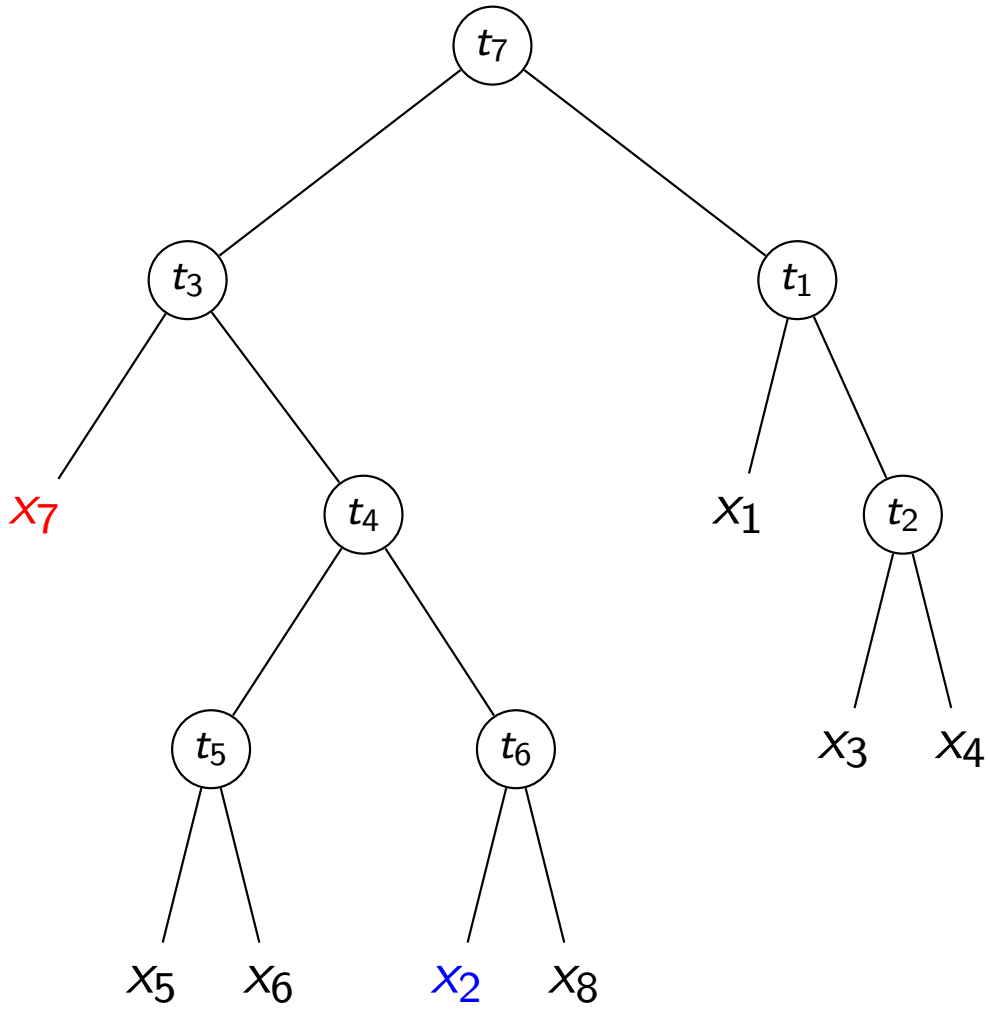
Sorting a Log Depth Tree



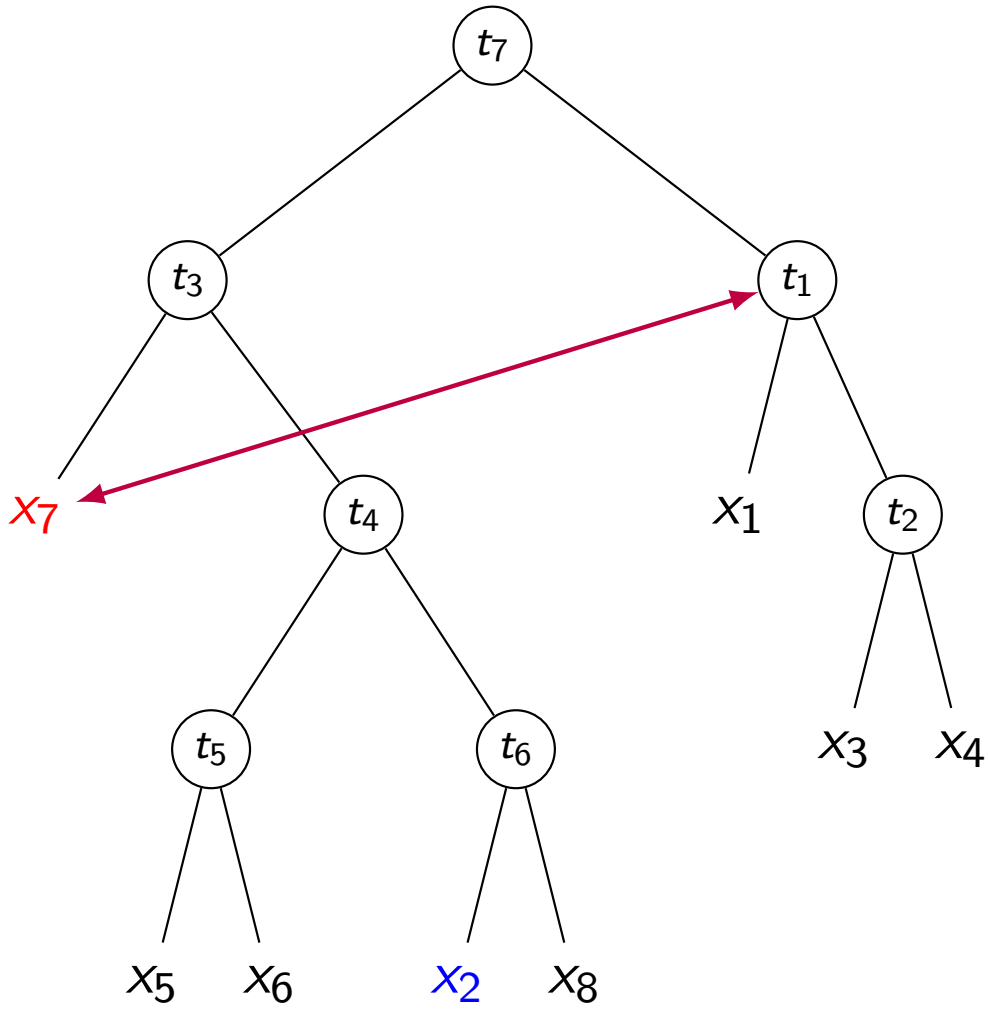
Sorting a Log Depth Tree



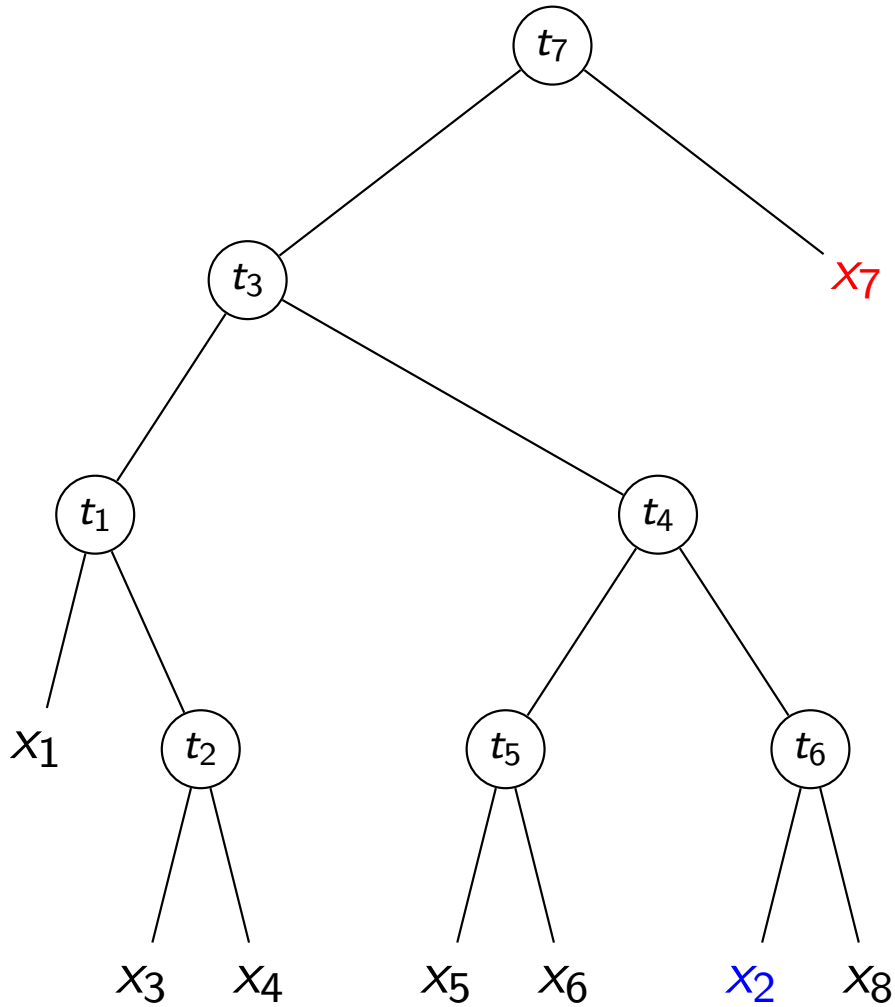
Sorting a Log Depth Tree



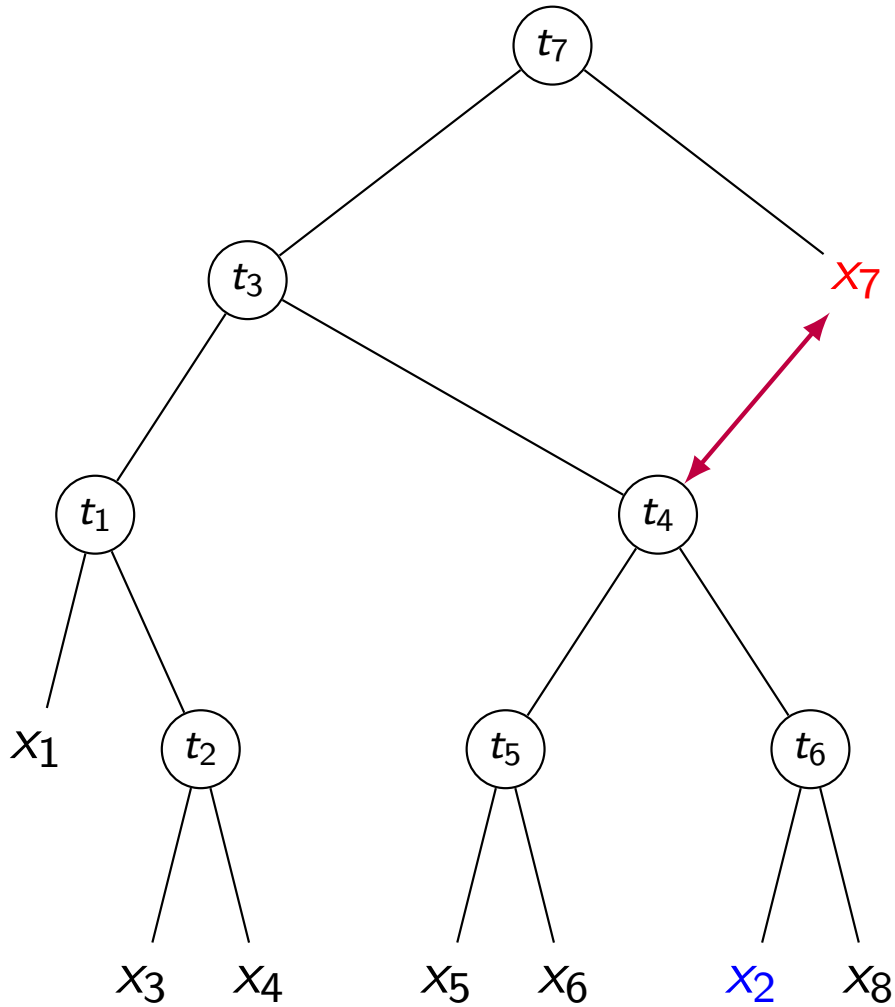
Sorting a Log Depth Tree



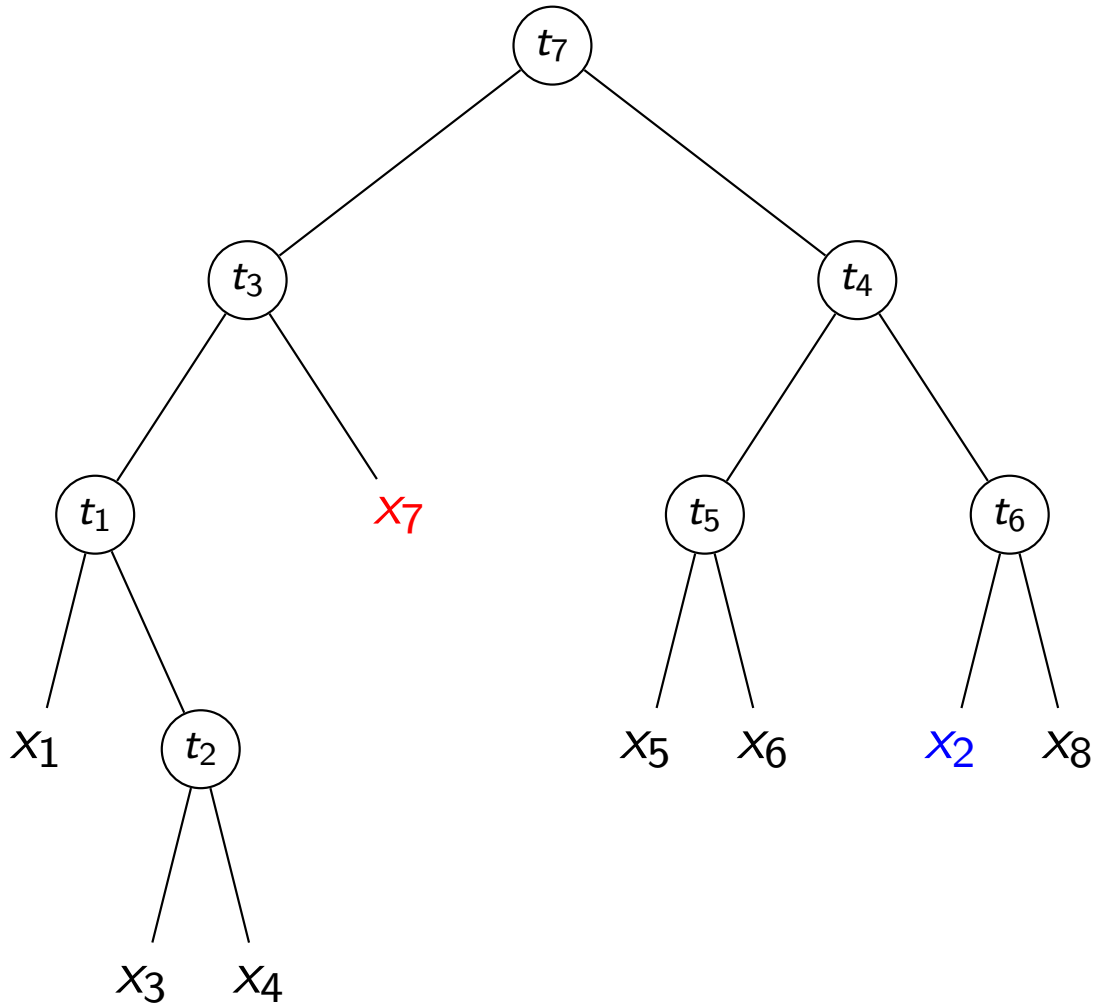
Sorting a Log Depth Tree



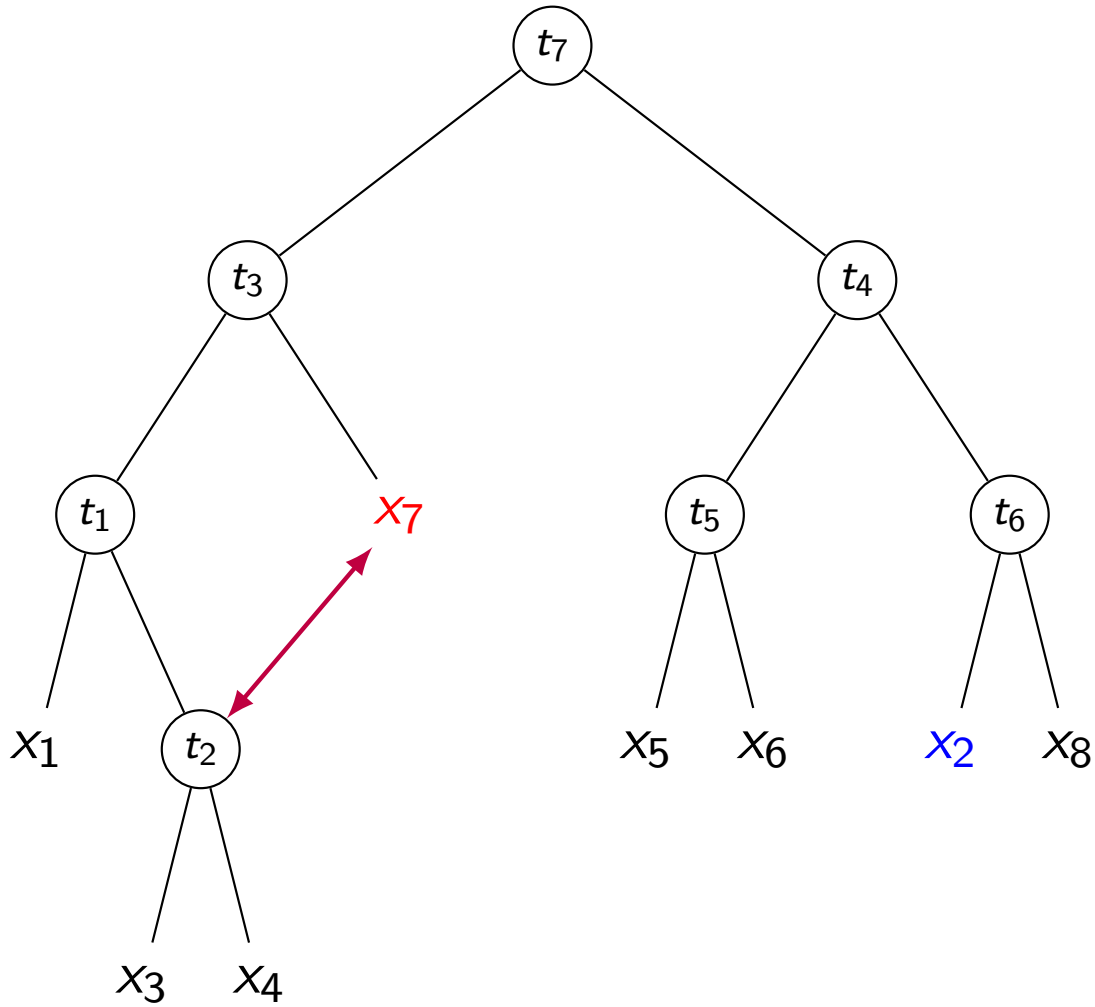
Sorting a Log Depth Tree



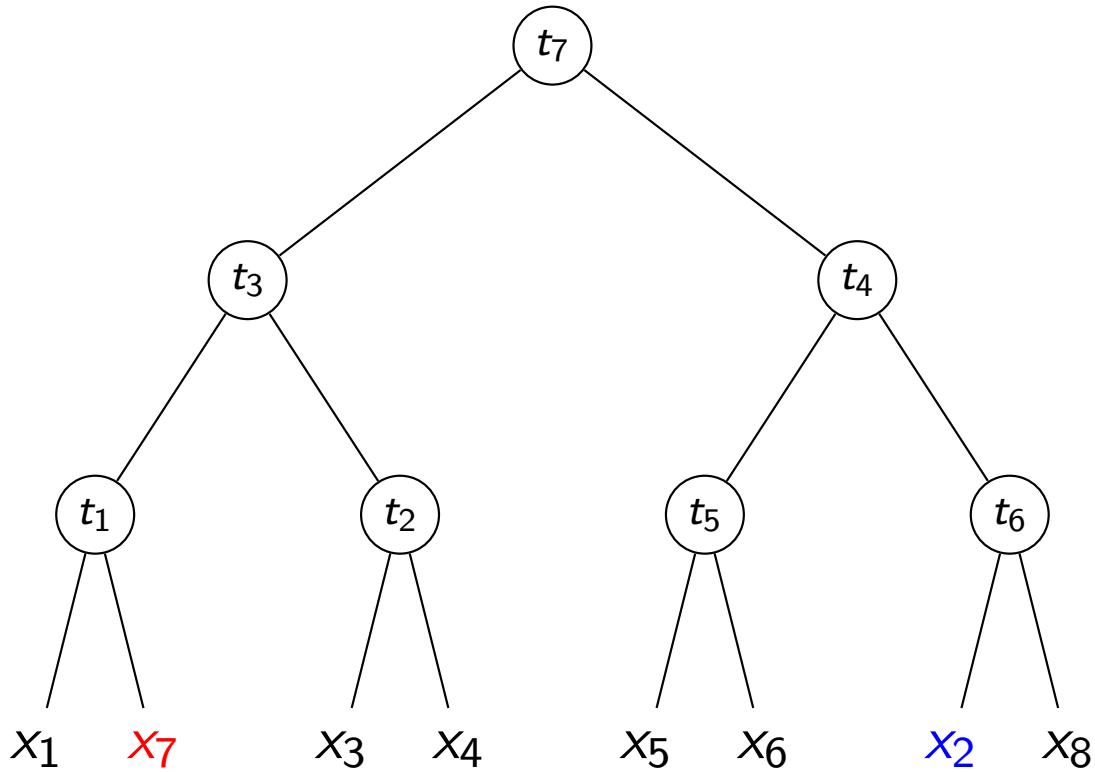
Sorting a Log Depth Tree



Sorting a Log Depth Tree

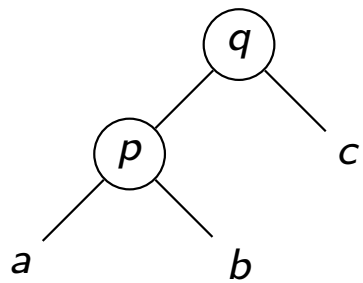


Sorting a Log Depth Tree



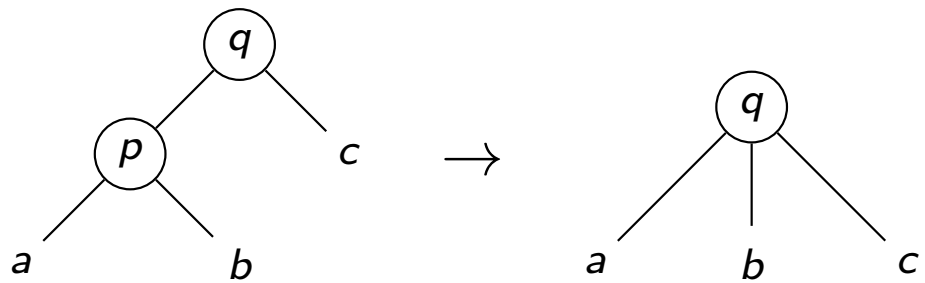
Logging Swaps in DRAT

Suppose we have two Tseitin variables p and q,
$p = a \oplus b$ and $q = p \oplus c$



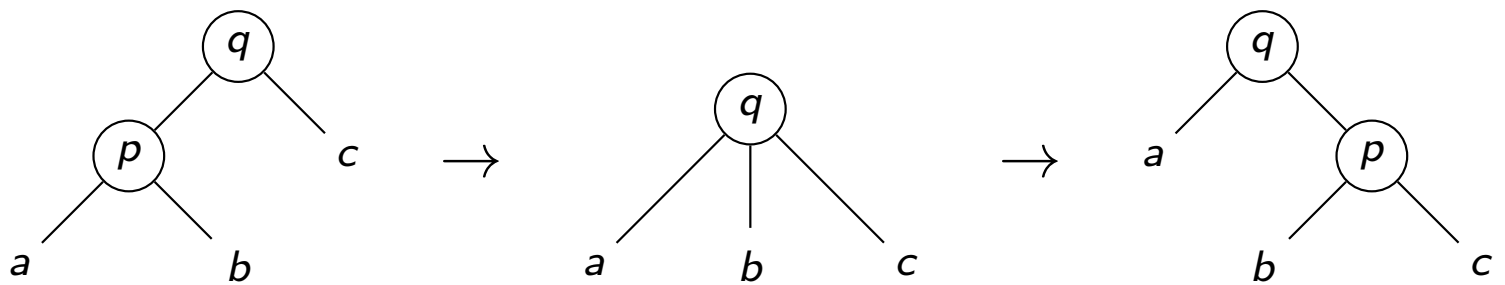
Logging Swaps in DRAT

ATA	RATE
$\bar{q} \vee a \vee b \vee c$	d $\bar{p} \vee a \vee b$
$\bar{q} \vee \bar{a} \vee \bar{b} \vee c$	d $\bar{p} \vee \bar{a} \vee \bar{b}$
$\bar{q} \vee a \vee \bar{b} \vee \bar{c}$	d $p \vee a \vee \bar{b}$
$\bar{q} \vee \bar{a} \vee b \vee \bar{c}$	d $p \vee \bar{a} \vee b$
$q \vee \bar{a} \vee b \vee c$	d $\bar{p} \vee q \vee c$
$q \vee a \vee \bar{b} \vee c$	d $\bar{p} \vee \bar{q} \vee \bar{c}$
$q \vee a \vee b \vee \bar{c}$	d $p \vee q \vee \bar{c}$
$q \vee \bar{a} \vee \bar{b} \vee \bar{c}$	d $p \vee \bar{q} \vee c$

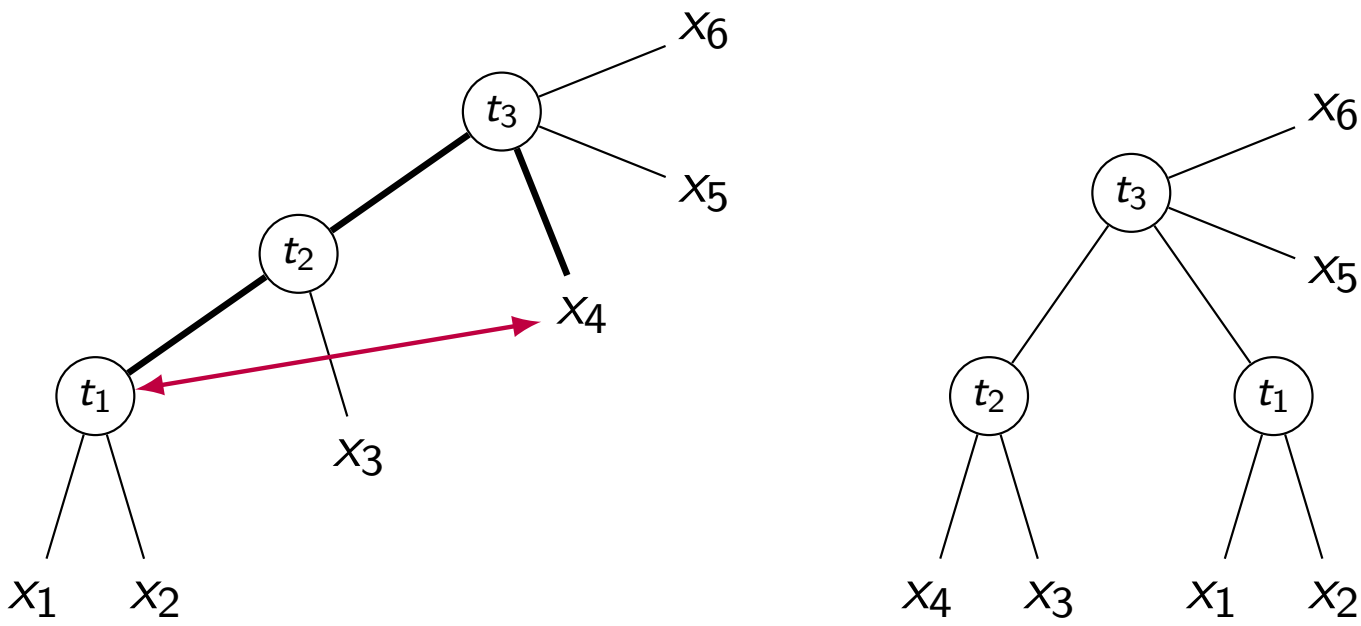


Logging Swaps in DRAT

ATA	RATE	RATA	ATE
$\bar{q} \vee a \vee b \vee c$	d $\bar{p} \vee a \vee b$	$\bar{p} \vee c \vee b$	d $\bar{q} \vee a \vee b \vee c$
$\bar{q} \vee \bar{a} \vee \bar{b} \vee c$	d $\bar{p} \vee \bar{a} \vee \bar{b}$	$\bar{p} \vee \bar{c} \vee \bar{b}$	d $\bar{q} \vee \bar{a} \vee \bar{b} \vee c$
$\bar{q} \vee a \vee \bar{b} \vee \bar{c}$	d $p \vee a \vee \bar{b}$	$p \vee c \vee \bar{b}$	d $\bar{q} \vee a \vee \bar{b} \vee \bar{c}$
$\bar{q} \vee \bar{a} \vee b \vee \bar{c}$	d $p \vee \bar{a} \vee b$	$p \vee \bar{c} \vee b$	d $\bar{q} \vee \bar{a} \vee b \vee \bar{c}$
$q \vee \bar{a} \vee b \vee c$	d $\bar{p} \vee q \vee c$	$\bar{p} \vee q \vee a$	d $q \vee \bar{a} \vee b \vee c$
$q \vee a \vee \bar{b} \vee c$	d $\bar{p} \vee \bar{q} \vee \bar{c}$	$\bar{p} \vee \bar{q} \vee \bar{a}$	d $q \vee a \vee \bar{b} \vee c$
$q \vee a \vee b \vee \bar{c}$	d $p \vee q \vee \bar{c}$	$p \vee q \vee \bar{a}$	d $q \vee a \vee b \vee \bar{c}$
$q \vee \bar{a} \vee \bar{b} \vee \bar{c}$	d $p \vee \bar{q} \vee c$	$p \vee \bar{q} \vee a$	d $q \vee \bar{a} \vee \bar{b} \vee \bar{c}$



Rebalancing



- Internal nodes can be swapped to balance the tree.
- By divide and conquer this is $O(n \log n)$ to make a linear tree log depth.

Complexity

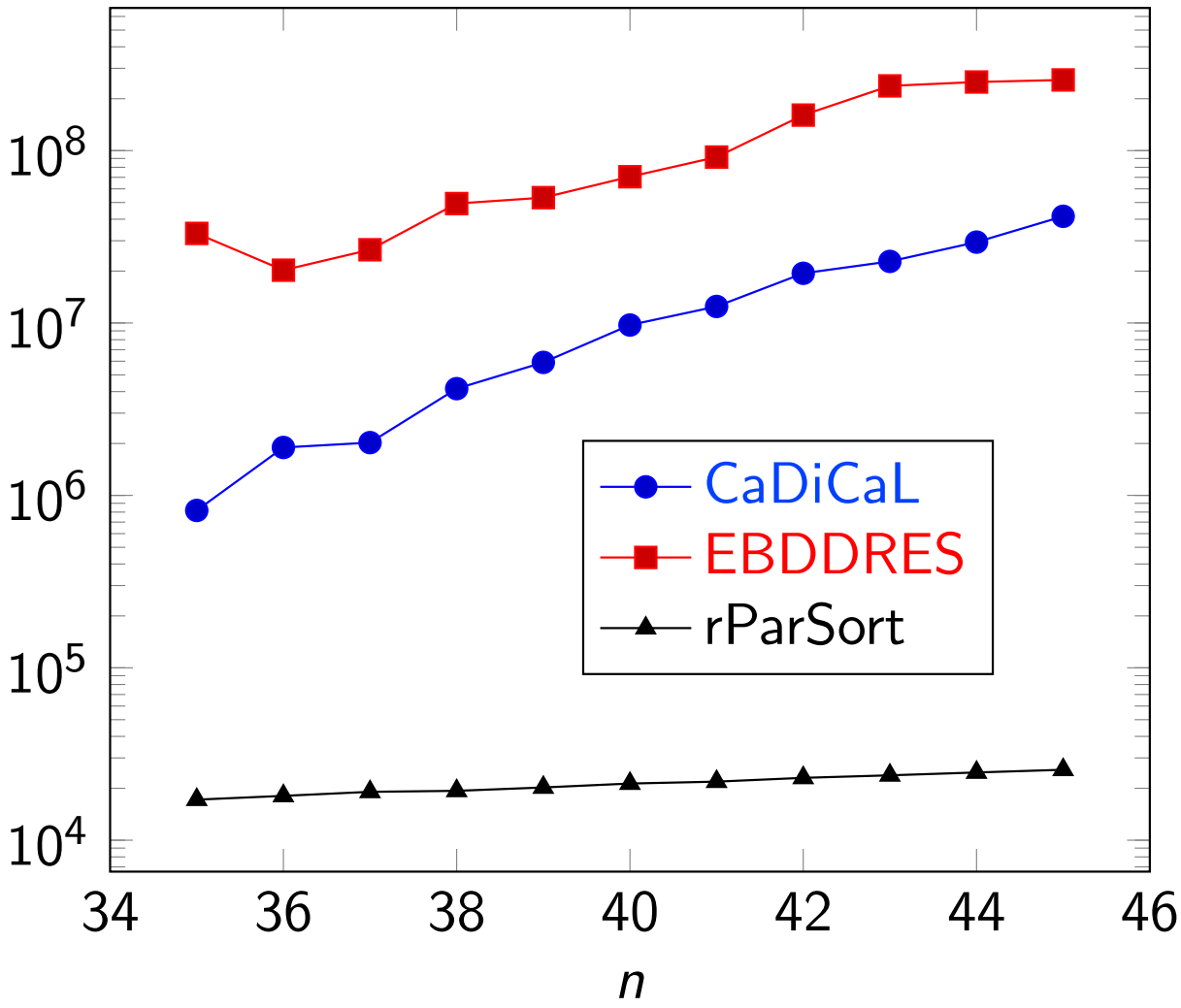
- Rebalancing the tree is $O(n \log n)$ because each level needs up to $n/2$ swaps. And there will be $\log n$ levels.
- Every transposition swap between two leaves, takes at most $4 \log n$.
- There is a maximum of n transpositions.
- The rebalancing needs to be undone which takes $O(n \log n)$
- The formulas are now in the `Dubois` family and can be refuted by an $O(n)$ size resolution proof.

$$O(n \log n) + n \cdot 4 \log n + O(n \log n) + O(n) = O(n \log n)$$

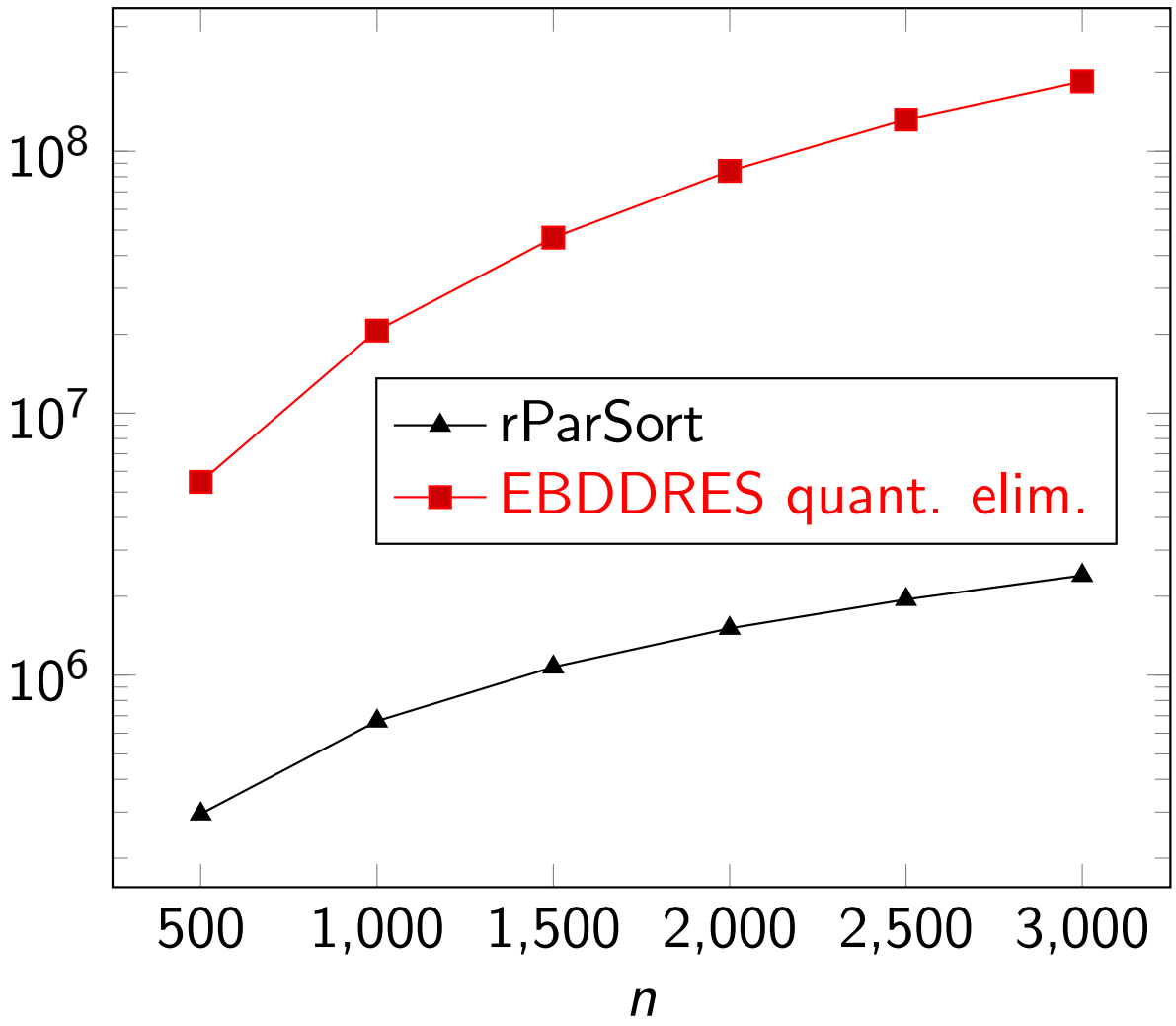
Experimental Setup

- We ran a program rParSort that generated a random instance of reordered parity and also generated a DRAT proof based on the aforementioned $O(n \log n)$ technique.
- rParSort proofs were very compact- 150MB for $n = 1000$.
- We compare the size of our proofs by ones produced by the state-of-the-art SAT solver CaDiCaL
- Another technique EBDDRES, solves the instance using binary decision diagrams and turns the construction into an ER proof.
- CaDiCaL and EBDDRES could manage up to $n = 45$, but started reaching > 5000 seconds time-outs, afterwards.

Comparisons I



Comparisons II (Not in Paper)



Summary

- Reordered parity constraints are hard for CDCL without Gaussian elimination
- Sorting the leaves can give an $O(n \log n)$ short proofs.
- The DRAT proof we have create no new variables, instead re-use existing ones.
- We have a working algorithm generating these short proofs, and these are outperforming all other existing tools.

Further Work

- The current algorithm assumes the structure of the tree, this can be made automatic.
- Reordered parity formulas are special cases of Tseitin graphs [Tseitin 1968]. We plan to generalise our technique for refutations of Tseitin formulas.
- The original problem was because Gaussian elimination does not support proofs. With adaptation we can use this sorting tool to help resolve two xor-constraints and simulate Gaussian elimination.