Reduced Ordered Binary Decision Diagrams as compacted tree-structures: Enumeration and Sampling

Antoine Genitrini[†]
with Julien Clément[*]

workshop CLA

October 13, 2020

[†]LIP6 @ Sorbonne University
[*]GREYC @ Normandie University

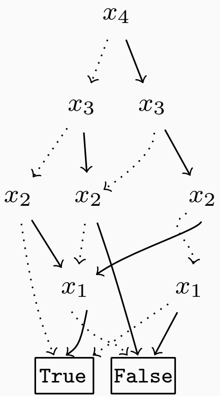
Binary Decision Diagram

Let f be a Boolean function in k variables.

A *Binary Decision Diagram* is a compact representation of f allowing to evaluate it efficiently.

It is based on some divide-and-conquer principle.

[Wegener00]: Branching Programs and Binary Decision Diagrams
[Knuth11]: The Art of Computer Programming (vol.4)

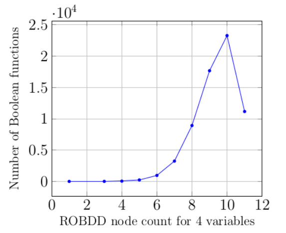


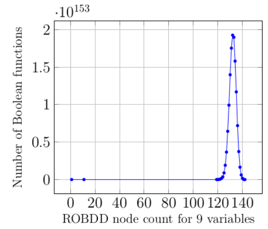
We are interested in Reduced Ordered Binary Decision Diagrams, denoted ROBDDs from now.
We take a point of view of a combinatorialist.

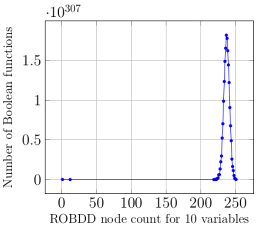
Motivations: J. Newton & D. Verna (2018)

A Theoretical and Numerical Analysis of the Worst-Case Size of Reduced Ordered Binary Decision Diagrams

Article *in* ACM Transactions on Computational Logic · January 2018







$$2^{2^4} = 2^{16} = 65,536$$

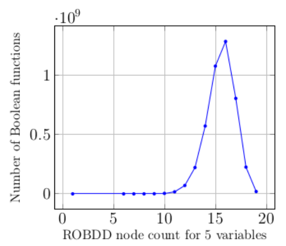
$$2^{2^5} = 2^{32} = 4,294,967,296$$

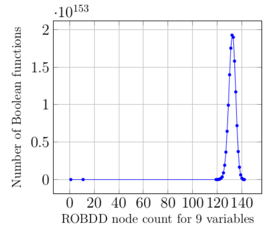
$$2^{2^9} = 2^{512} \approx 1.34 \cdot 10^{154}$$

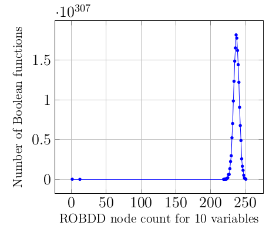
$$2^{2^{10}} = 2^{1024} \approx 1.80 \cdot 10^{308}$$

No. Variables (n)	No. Samples	No. Unique Sizes	Compute Time hh:mm:ss	Seconds per ROBDD
5	500,003	15	10:26:41	0.075
6	400,003	18	17:51:42	0.161
7	486,892	16	73:02:01	0.54
8	56,343	17	35:22:15	2.26
9	94,999	26	292:38:58	11.09
10	17,975	35	304:34:35	61.0

Motivations: J. Newton & D. Verna (2018)







Specific functions with small ROBDDs (in k variables):

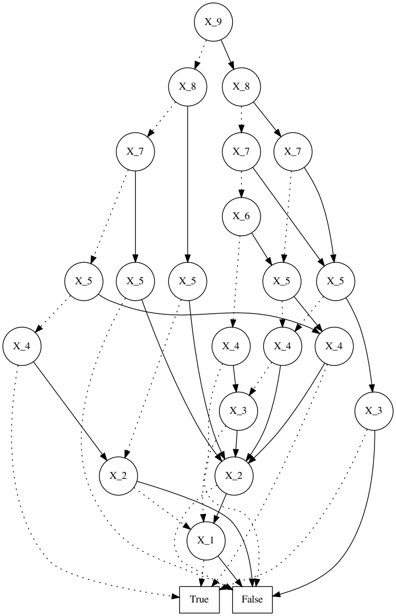
- addition in k bits,
- read-once fcts,
- symmetric fcts: $O(k^2)$.

ℓ threshold functions: $5 \leq k \leq 10$; $1 \leq \ell \leq k$

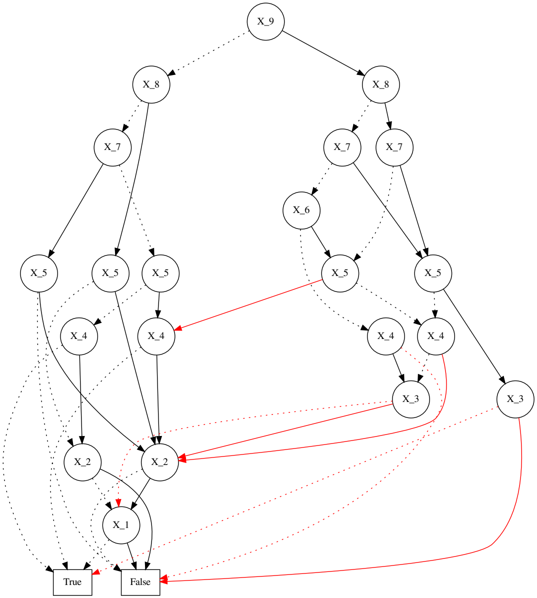
7	10	**11**	10	7					
8	12	**14**	14	12	8				
9	14	17	**18**	17	14	9			
10	16	20	**22**	22	20	16	10		
11	18	23	26	**27**	26	23	18	11	
12	20	26	30	**32**	32	30	26	20	12

(There are also interesting functions with large ROBDDs – size exponential in k – like the multiplier, the hidden weighted bit fct, . . .)

Why does the enumeration be difficult?

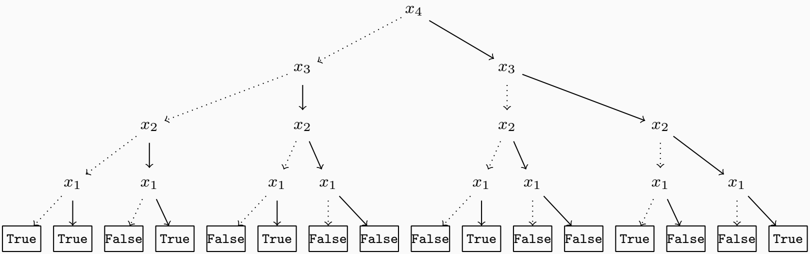


Why does the enumeration be difficult?



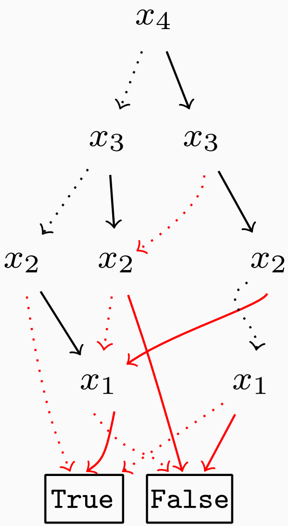
Outline of the talk

- Combinatorial preliminaries



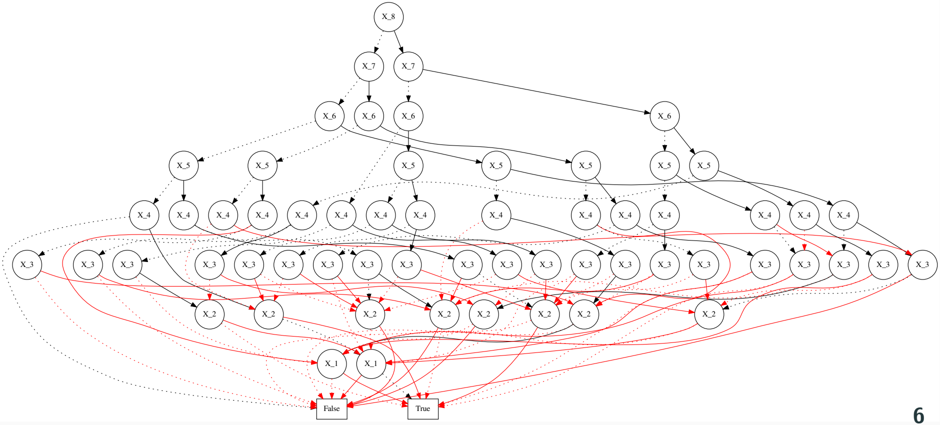
Outline of the talk

- Combinatorial preliminaries
- Recursive counting



Outline of the talk

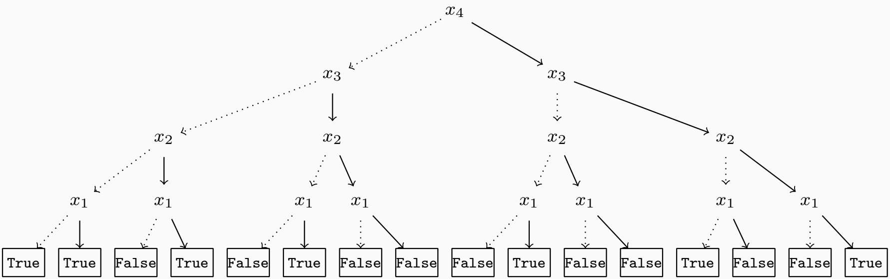
- Combinatorial preliminaries
- Recursive counting
- Unranking an ROBDD (key-ideas)



Combinatorial preliminaries

Boolean functions: decision tree representation

A *decision tree* is a data structure representing a Boolean function.



In the talk we suppose that the tree is plane.

At node x:

the left subtree of a node x is traversed when x is assigned to `False`.

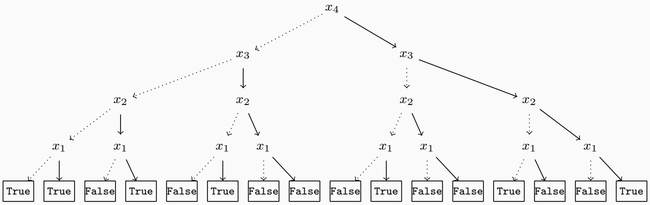
the right subtree is traversed when x is assigned to `True`.

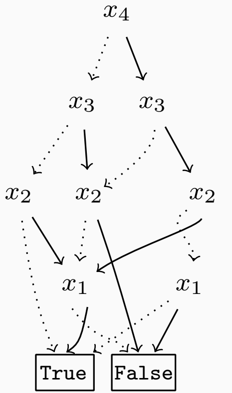
In the rest of the talk, we use the following order $x_k, \ldots, x_1$.

Boolean functions: ROBDD representation

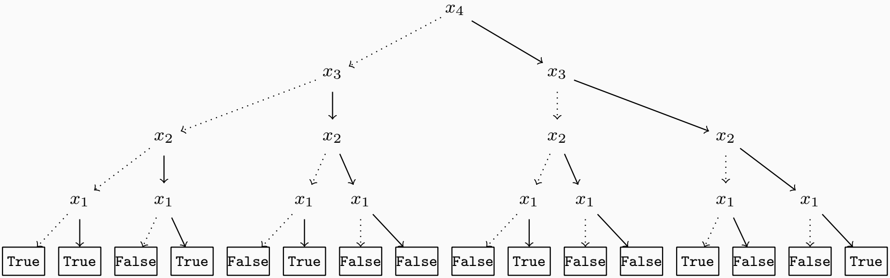
A Reduced Ordered Binary Decision Diagram is a compacted data structure, based on the decision tree of a function and obtained with the following rules of compaction:

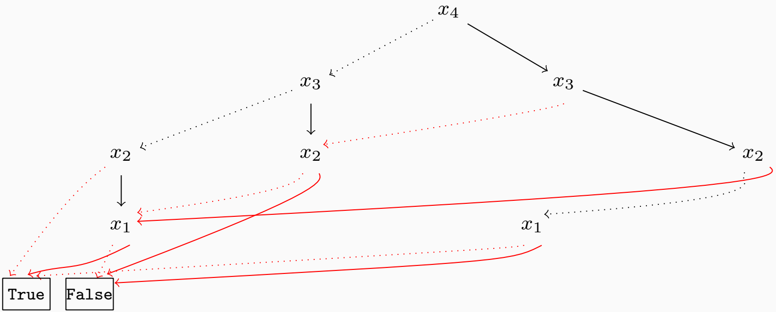
- Eliminate any node with two identical children;
- Merge any identical subtrees.



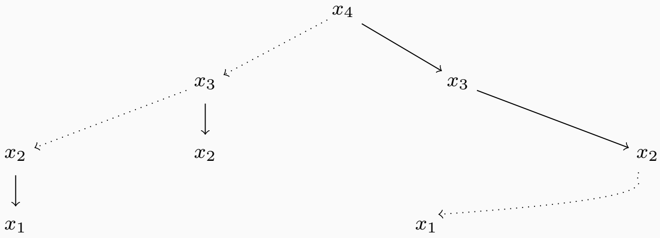


Compaction through a postorder traversal



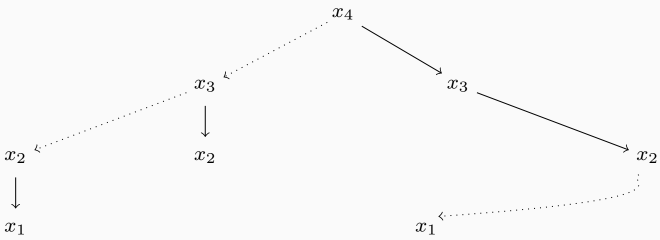


Spine of an ROBDD



The spine of an ROBDD is the *spanning tree* obtained by a postorder traversal of the plane ROBDD, omitting the sinks.

Spine of an ROBDD



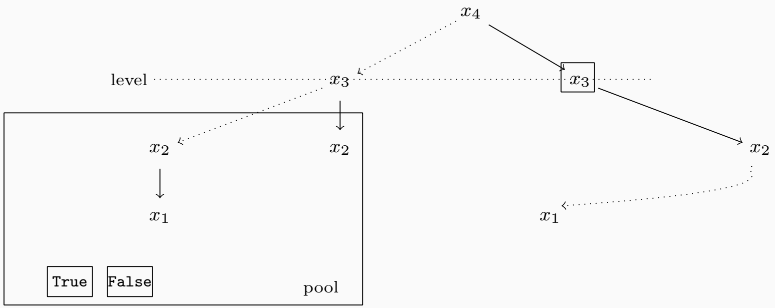
The spine of an ROBDD is the *spanning tree* obtained by a postorder traversal of the plane ROBDD, omitting the sinks.

Goal

- Definition of an equivalence relation to partition the set of ROBDDs according to their spines;
- Count for each spine how many ROBDDs have this spine.

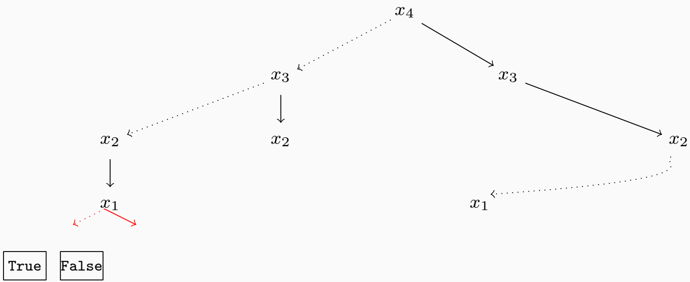
Recursive counting

Node: level rank, pool and weight



Let T be a spine. The weight $w_T(\nu)$ (for $\nu \in Q'$) is the number of possibilities for completing $\delta'(\nu, \cdot)$
and yielding (at the end) an ROBDD with spine T.
(the weight is a multiplicative parameter)

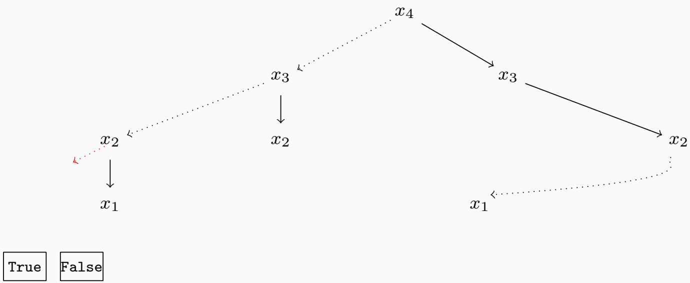
Weight of a spine



The weight is *a multiplicative parameter* for the complete spine.

- $2 \cdot 1$

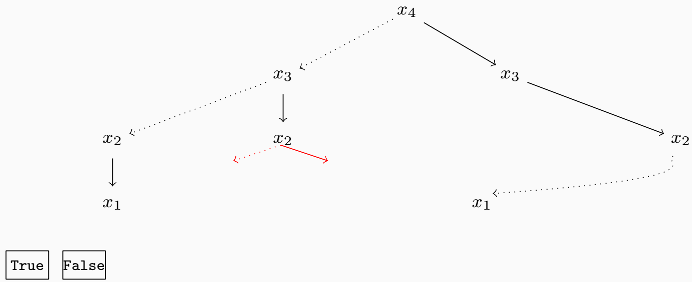
Weight of a spine



The weight is *a multiplicative parameter* for the complete spine.

- $2 \cdot 1$
- 2

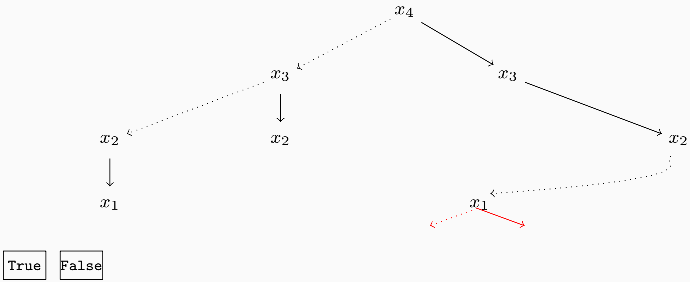
Weight of a spine



The weight is *a multiplicative parameter* for the complete spine.

- $2 \cdot 1$
- 2
- $3 \cdot 2 - 1$

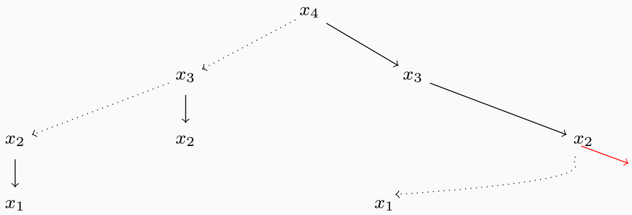
Weight of a spine



The weight is *a multiplicative parameter* for the complete spine.

- $2 \cdot 1$
- 2
- $3 \cdot 2 - 1$
- $2 \cdot 1 - 1$

Weight of a spine

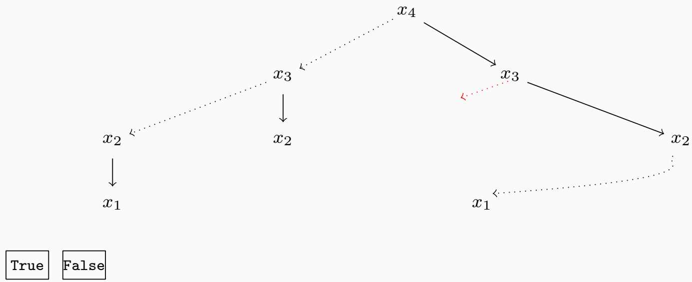


True False

The weight is *a multiplicative parameter* for the complete spine.

- 2
- 2
- $3 \cdot 2 - 1$
- $2 \cdot 1 - 1$
- $4 - 1$

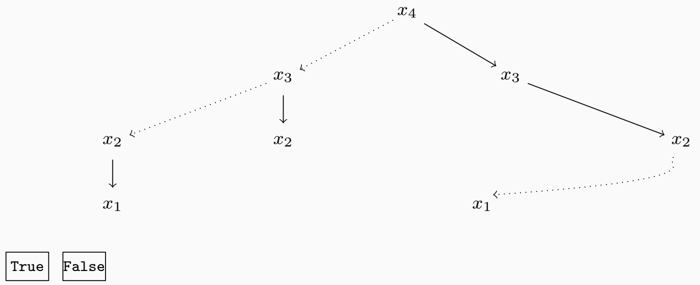
Weight of a spine



The weight is *a multiplicative parameter* for the complete spine.

- 2
- 2
- $3 \cdot 2 - 1$
- $2 \cdot 1 - 1$
- $4 - 1$
- 5

Weight of a spine



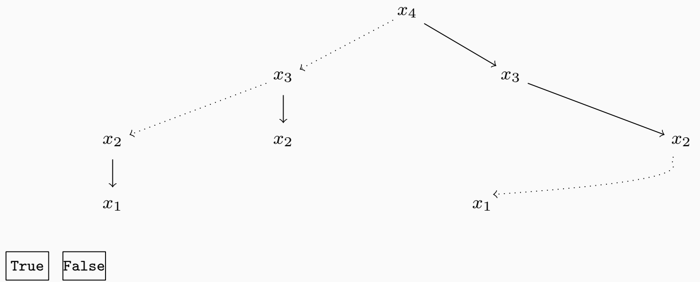
The weight is *a multiplicative parameter* for the complete spine.

- 2
- 2
- $3 \cdot 2 - 1$
- $2 \cdot 1 - 1$
- $4 - 1$
- 5

Total weight: $2 \cdot 2 \cdot 5 \cdot 1 \cdot 3 \cdot 5 = 300$.

$\Longrightarrow$ 300 ROBDDs built with this spine

Node weight computation



$$w_T(\nu) = \begin{cases} 1 & \text{if } \delta'(\nu,0) \neq \text{nil} \ \& \ \delta'(\nu,1) \neq \text{nil} \\ |p_T(\nu)|\big(|p_T(\nu)| - 1\big) - s_T(\nu) & \text{if } \delta'(\nu,0) = \delta'(\nu,1) = \text{nil} \\ |p_T(\nu)| & \text{if } \delta'(\nu,0) = \text{nil} \ \& \ \delta'(\nu,1) \neq \text{nil} \\ |p_T(\nu) + \text{profile}(T')| & \text{if } \delta'(\nu,0) \neq \text{nil} \ \& \ \delta'(\nu,1) = \text{nil} \end{cases}$$

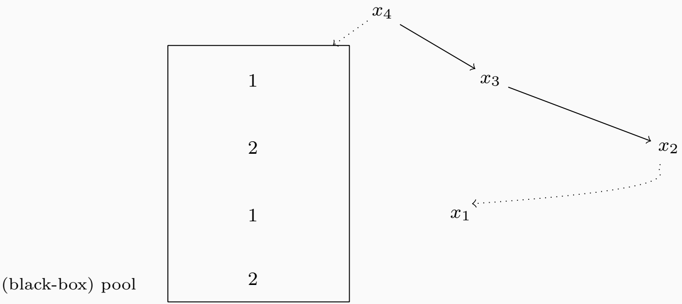
where $p_T(\nu)$ is the pool profile of node ν, s_T is the level rank and $T' = T_{\nu_0}$ is the subtree rooted at $\nu_0 = \delta'(\nu, 0)$.

Pool-ROBDDs

- First equivalence relation: ROBDDs to spines

Pool-ROBDDs

- First equivalence relation: ROBDDs to spines
- Second equivalence relation: spines to pool-spines



Time complexity

The time complexity (in the number of arithmetic operations) for partitioning the Boolean functions in k variables according to their ROBDD size is

$$O\left(\frac{1}{k}2^{3k^2/2+k}\right).$$

This value must be put in front of the number of Boolean functions: 2^{2^k}.

In practice, we compute the partition for $k = 8$ in about 2 hours on a personal computer with a python implementation
and for $k = 9$, in 3 days using a fast computer with a C++ implementation.

cf. `https://github.com/agenitrini/BDDgen`

Time complexity

The time complexity (in the number of arithmetic operations) for partitioning the Boolean functions in k variables according to their ROBDD size is

$$O\left(\frac{1}{k}2^{3k^2/2+k}\right).$$

Proof ideas:

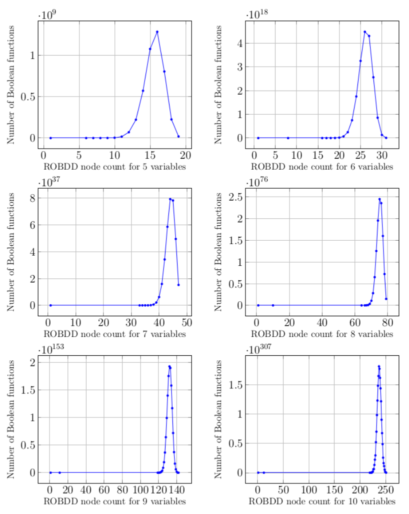
- We have no constructive spine builder without filtering bin. trees.

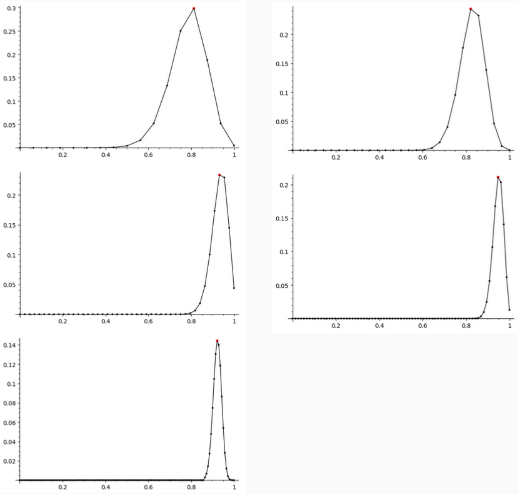
Thus we proceed differently:

- The largest size of an ROBDD in k var. is $m_k = O\left(\frac{2^k}{k}\right)$.
- Let P be the number of profiles up to k var.:
  - for index ℓ there are between 1 and 2^ℓ nodes: $P = O(2^{k^2/2})$
  - for an entry profile and an exit profile, we must build the intermediate profile: $O(P^3)$

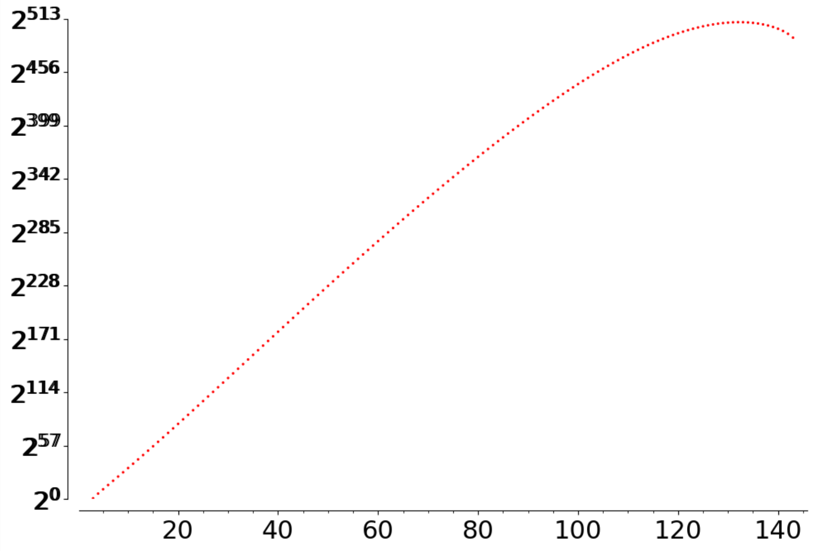
$\Longrightarrow$ time complexity: $O(m_k \cdot P^3)$.

Feedback: J. Newton & D. Verna (2018)





With a logarithmic scale for functions in 9 variables



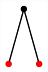
Unranking an ROBDD (key-ideas)

The *Unranking* method for binary trees [NW75,MM03]

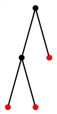
1. Defining a total order over trees (of the same size).
2. Constructing the tree only by using its rank.

$< 0, 0 >$

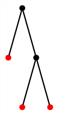
$< 1, 0 >$



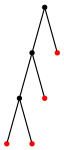
$< 2, 0 >$



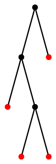
$< 2, 1 >$



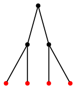
$< 3, 0 >$



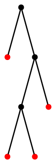
$< 3, 1 >$



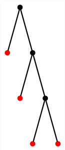
$< 3, 2 >$



$< 3, 3 >$



$< 3, 4 >$



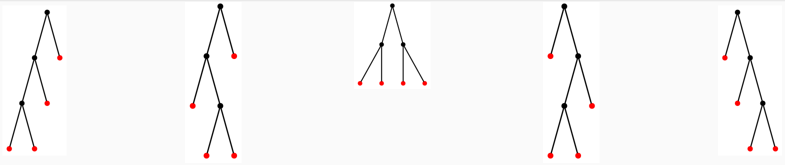
The *Unranking* method for binary trees [NW75,MM03]

1. Defining a total order over trees (of the same size).
2. Constructing the tree only by using its rank.

$< 0, 0 >$ $< 1, 0 >$ $< 2, 0 >$ $< 2, 1 >$

Size decomposition for binary trees

$$B_{n+1} = B_n \cdot B_0 + B_{n-1} \cdot B_1 + B_{n-2} \cdot B_2 + B_{n-3} \cdot B_3 + \dots$$



Unranking a binary tree

Number of binary trees according to their size:

n	0	1	2	3	4	5	6	7	8	9	10	...
$\mathtt{Cat}_n$	1	1	2	5	14	42	132	429	1430	4862	16796	...

Among the 429 trees of size 7,
we want to build the tree with rank 250.

$$429 = 132 \cdot 1 + 42 \cdot 1 + 14 \cdot 2 + 5 \cdot 5 + 2 \cdot 14 + 1 \cdot 42 + 1 \cdot 132.$$

Unranking a binary tree

Number of binary trees according to their size:

n	0	1	2	3	4	5	6	7	8	9	10	...
$\mathtt{Cat}_n$	1	1	2	5	14	42	132	429	1430	4862	16796	...

Among the 429 trees of size 7,
we want to build the tree with rank 250.

$$429 = \underbrace{\underbrace{\underbrace{\underbrace{\underbrace{132 \cdot 1}_{132} + 42 \cdot 1}_{174} + 14 \cdot 2}_{202} + 5 \cdot 5}_{227} + 2 \cdot 14}_{255} + 1 \cdot 42 + 1 \cdot 132.$$

Thus the tree admits a size decomposition $(2, 4)$ and the ranks for the subtrees are $(250 - 227)//14 = 1$ and $(250 - 227)\%14 = 9$.

Unranking a binary tree

Number of binary trees according to their size:

n	0	1	2	3	4	5	6	7	8	9	10	...
$\mathtt{Cat}_n$	1	1	2	5	14	42	132	429	1430	4862	16796	...

Among the 429 trees of size 7,
we want to build the tree with rank 250.

$429 =$ … $132.$

The unranking method does not adapt directly,
to the sampling of ROBDDs:
the recursive calls are not independent.

202

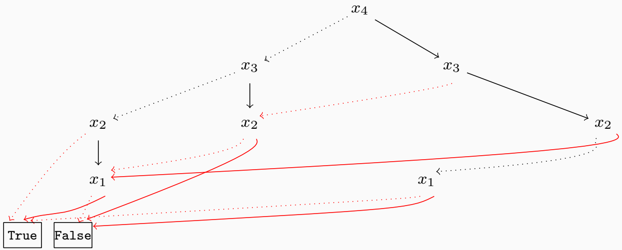
227

255

Thus the tree admits a size decomposition $(2, 4)$ and the ranks for the subtrees are $(250 - 227)//14 = 1$ and $(250 - 227)\%14 = 9$.

Unranking an ROBDD: main ideas

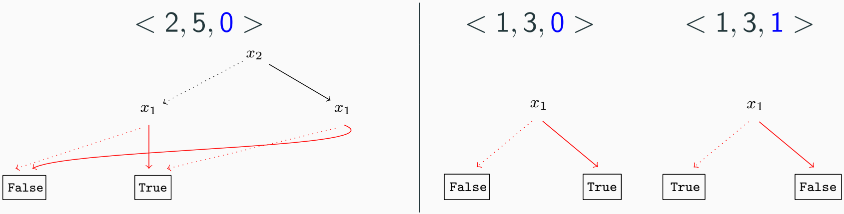
Recall: While traversing an ROBDD, the pointers are all going to already visited nodes.



- for substructure containing at least one node,
  the recursive calls are evaluated through the postorder traversal
- for a given node, all the substructures it can point to must be dynamically ranked
- pay attention not to define twice the same substructure

Unranking an ROBDD: example

There are 2 ROBDDs with 2 variables of size 5.



The rank of this size-5 ROBDD is 0.

During the construction, the rank of the leftmost child is 0 but the rank of the rightmost child is also 0.

But both children are distinct.

During the call for the rightmost child, there is only one available substructure of size 1.

Conclusion and future work

- Our combinatorial approach adapts very well.
  - subclasses of fct: ex. only with essential variables
  - other classes of BDDs (with other compaction rules)
    - OBDDs
    - Quasi-reduced BDDs
    - Zero-supressed BDDs
- We can use an analogous strategy than [E-PFW20] to enumerate ROBDDs through enriched walks.
- In the future:
  - Combinatorial characterization of the class of spines ?
  - A better equivalence relation for a better counting/sampling method?