Efficient OBDD-Based Boolean Manipulation in CAD Beyond Current Limits

Jochen Bern Christoph Meinel Anna Slobodová*

FB IV – Informatik,
Universität Trier,
D–54 286 Trier, Germany

Abstract— We present the concept of TBDD's which considerably enlarges the class of Boolean functions that can be efficiently manipulated in terms of OBDD's. It extends the idea of using domain transformations, which is well-known in many areas of mathematics, physics, and technical sciences, to the context of OBDD–based Boolean function manipulation in CAD: Instead of working with the OBDD-representation of a function f, TBDD's allow working with an OBDD-representation of a suited cube transformed version of f.

Besides of giving some theoretical insights into the new concept, we investigate in some detail cube transformations which are based on complete types. We

- **show that such TBDD–representations can be derived similarly as OBDD–representations,**
- **give evidence of the practical importance of such TBDD's by presenting very small-size TBDD-representations of the *hidden weighted bit* functions HWB_n which were proved to have only very large OBDD-representations, and**
- **report some promising experimental results with some ISCAS benchmark circuits including the multiplier circuit C6288.**

I. Introduction

One of the fundamental problems in computer-aided circuit design and other areas of practical and theoretical computer science is the task of representing and manipulating Boolean functions. Although, in principle, any valid representation is allowed, some representations may be preferred because they are more efficient in memory, or more efficient to manipulate, or more indicative of the complexity of the final implementations. Hence, the search for good trade-offs between these competing objectives – space-efficient representation of the considered functions and time-efficient manipulation algorithms – is a central theme of research [e.g. Bry86, Kar89, BCMD90, JBAF92, GM93, BGMS94].

Developing the concept of TBDD's, we show that the use of domain transformation, well-known in many areas of mathematics, physics, and technical sciences (e.g., Fourier transformation, Laplace transformation, Z-transformation), can be applied successfully in the context of Boolean function manipulation in CAD, too. The idea behind is to work with the OBDD-representation of a suited cube transformed version of a function instead of working with the OBDD–representation of the original function itself.

There are two main advantages of such TBDD–representations. First, TBDD's allow Boolean function manipulation in terms of the well–known and well–comprehended data structure of OBDD's with all its nice features (see e.g. [Bry92]). Second, TBDD's can lead to function representations which are often much more compressed than OBDD–representations. Hence, the main disadvantage of Boolean function manipulation in terms of OBDD's – which consists in the often very extensive space requirements of OBDD's – is considerably reduced.

Besides of giving some theoretical insights into the new concept, we investigate cube transformations which are based on complete types [BGMS94]. In detail, we show that TBDD–representations can be derived similarly as OBDD–representations. Then we give evidence of the importance of the TBDD concept by presenting constant–size TBDD-representations of the *hidden weighted bit* functions HWB_n, $n \in \mathbb{N}$, which were proved to have only exponential size OBDD-representations. Finally, we report some promising experimental results with some ISCAS benchmark circuits including the multiplier circuit C6288.

II. Preliminaries

As usual let $\mathbb{B}_n = \{f : \{0,1\}^n \to \{0,1\}\}$ denote the Boolean algebra of all single output functions over the cube $\{0,1\}^n$.

*Granted by DFG Me 1077/2-1

32nd ACM/IEEE Design Automation Conference ®

Definition. *A cube transformation τ is a bijective mapping $\tau : \{0,1\}^n \to \{0,1\}^n$.*

A cube transformation τ induces a mapping $\phi_\tau : \mathbb{B}_n \to \mathbb{B}_n$ onto the Boolean algebra with $\phi_\tau(f)(a) = f \circ \tau(a)$ for each $a = (a_1, \ldots, a_n) \in \{0,1\}^n$.

Fact 1. *If τ is a cube transformation, then ϕ_τ defines an automorphism on $\mathbb{B}_n$.*
I.e., we have

1. *$f = g$ if and only if $\phi_\tau(f) = \phi_\tau(g)$, for each $f, g \in \mathbb{B}_n$.*
2. *Let $*$ be any binary operation on $\mathbb{B}_n$. If $f = f_1 * f_2$ for any $f_1, f_2 \in \mathbb{B}_n$ then $\phi_\tau(f) = \phi_\tau(f_1) * \phi_\tau(f_2)$.* □

To make notations as simple as possible, we shortly write $\tau(f)$ instead of $\phi_\tau(f)$.

We use cube transformations in order to obtain functions that can be represented much more succinct than the original one. The function representations we work with are the well-known OBDD's. OBDD's, introduced by Bryant [Bry86] as data structure for Boolean function manipulation, have obtained great importance. Due to their nice algorithmic properties, they provide nowadays the state-of-the-art data structure in many areas of computed-aided circuit design. (For a survey, we refer to [Bry92].) We will consider OBDD's that test variables in the natural order.

Definition. *A* binary decision diagram *(BDD) over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic graph with one source and at most two sinks labelled 0 and 1. Each non-sink node v is labelled with a Boolean variable from X_n and has two outgoing edges, one labelled with 0 and the other with 1. The* then–son *of v is reached via the* 1*–edge, the* else–son *is reached via the* 0*–edge. (In pictural representations, we do not indicate the edge labels if the* 0*-edge is drawn left of the* 1*-edge.) The* computation path *for an input $a = (a_1, \ldots, a_n)$ starts at the source. At an inner node with label x_i, the outgoing edge with label a_i is chosen. $size(P)$ denotes the number of non-sink nodes of P. A BDD P* represents *a Boolean function $f \in \mathbb{B}_n$ if the computation path for each input a leads to the sink labelled $f(a)$. f is sometimes denoted by f_P. A BDD is called* ordered binary decision diagram (OBDD) *if, on each path, the variables are tested consistently with the natural order of variables, i.e., $x_1 < x_2 < \ldots, < x_n$.*

Fact 2 [Bry86].

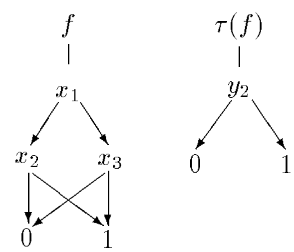
1. *Each Boolean function f over X_n can be represented by means of an OBDD, i.e., OBDD's provide a universal representation scheme.*
2. *The reduced OBDD for f is uniquely determined, i.e., it provides a canonical representation.*
3. *Let f_1, f_2 be Boolean functions represented by the OBDDs P_1, P_2, respectively. For every binary operation $*$, the reduced OBDD P for $f = f_1 * f_2$ can be constructed in time $O(size(P_1) \cdot size(P_2))$.* □

III. The TBDD–Concept

Combining appropriate cube transformations with the OBDD's, we simultaneously are able to exploit the nice features of OBDD's as summarized in Fact 2 and to increase the succinctness of the representation. The resulting data structure is called TBDD.

Definition. *Let $f \in \mathbb{B}_n$ be a Boolean function, and let τ be a cube transformation $\tau : \{0,1\}^n \to \{0,1\}^n$. A* τTBDD–representation *of f (shortly called a τTBDD or, simply, TBDD) is an OBDD-representation P of $\tau(f)$. (I.e. $f_P(a) = \tau(f)(a) = f \circ \tau(a)$ for each $a \in \{0,1\}^n$.)*

x	$f(x)$	$\tau(x)$	$\tau(f)(x)$
$(0,0,0)$	0	$(0,0,0)$	0
$(0,0,1)$	0	$(0,0,1)$	0
$(0,1,0)$	1	$(0,1,0)$	1
$(0,1,1)$	1	$(0,1,1)$	1
$(1,0,0)$	0	$(1,0,0)$	0
$(1,0,1)$	1	$(1,1,0)$	0
$(1,1,0)$	0	$(1,0,1)$	1
$(1,1,1)$	1	$(1,1,1)$	1



(a) (b)

Figure 1. A cube transformation τ on $\{0,1\}^3$ together with an OBDD–representation (a) and a τTBDD–representation (b) of $f = \overline{x_1}x_2 + x_1x_3$.

The properties of OBDD's (Fact 2) together with the fact that cube transformations induce automorphisms of $\mathbb{B}_n$ (Fact 1) provide the following properties of TBDD's.

Theorem 3. *Let $f, f_1, f_2 \in \mathbb{B}_n$ be Boolean functions, and let τ be a cube transformation onto $\{0,1\}^n$.*

1. **Universality of TBDD–representation.**
Each Boolean function $f \in \mathbb{B}_n$ can be represented by means of a (reduced) τTBDD.
2. **Canonicity of TBDD–representation.**
Each Boolean function $f \in \mathbb{B}_n$ has exactly one reduced τTBDD–representation, i.e., TBDD's provide a canonical representation scheme.
3. **Efficient synthesis of TBDD's.**
Let T_1, T_2 be τTBDD–representations of f_1, f_2. Then, for any binary operation $*$, the (reduced) τTBDD-representation T of $f = f_1 * f_2$ corresponds to $T_1 * T_2$ and, therefore, can be constructed in time $O(size(T_1) \cdot size(T_2))$.

4. **Efficient equivalence test for TBDD's.**
Let T_1, T_2 be τTBDD's for f_1, f_2. The equivalence of the functions f_1 and f_2 corresponds to the equivalence of $\tau(f_1)$ and $\tau(f_2)$, which in turn corresponds to the functional equivalence of the OBDD's T_1 and T_2. Therefore, it can be tested in linear time.

The universality of TBDDs, i.e., their ability to represent any Boolean function, is an important property. However, it remains still insufficient if the representations are too large. The power of the cube transformation approach comes – at least theoretically – to a full expression by the statement that each function over X_n can be transformed into a function (defined in the same number of variables) whose OBDD–representation is of size n.

Proposition 4. *For any Boolean function $f \in \mathbb{B}_n$, there exists a TBDD-representation of size n.*

Proof. Let $k = \sharp on(f)$, and consider the inputs $a \in \{0,1\}^n$ as binary representations of the numbers $0, \ldots, 2^n - 1$. Let τ be the bijection that maps the strings representing the numbers $0, \ldots, k-1$ to the inputs of $on(f)$. Then $\tau(f)$ can be represented by an OBDD that merely has to test whether the input is smaller than k (in this case $\tau(f)$ computes 1) or not (in this case $\tau(f)$ computes 0). Since such a test can be performed by an OBDD of size n we are done. □

However, we should not carelessly overestimate the practical consequences of this result. It can be difficult to find, and even to store and manipulate an optimal cube transformation for a given f. Nevertheless, our experiments show a great advantage of TBDD's even if we work with non-optimal cube transformations.

We conclude this section by giving a rough sketch how to make use of the TBDD–concept in practical applications. In order to do this, let us remember how OBDD's are used in one of their favorite applications, namely in combinational circuit verification. Starting with a net-list description of two (single output) circuits C and C', we (try to) show that C and C' compute the same function (i.e. $f_C = f_{C'}$) by comparing their OBDD–representations P_C and $P_{C'}$. The canonicity of the OBDD–representation implies $f_C = f_{C'}$ iff $P_C = P_{C'}$. The OBDD–representation of a circuit C (more exactly of the function f_C computed by C) is constructed by "symbolic simulation" of C: Starting with the (trivial) OBDD–representations of the input variables $x_1, \ldots, x_n$, one successively constructs OBDD's for each gate g of C from the OBDD's of the predecessor gates of g by applying the operation associated with g. Unfortunately, for many even relatively simple circuits, it is practically impossible to construct OBDD–representations because of their huge sizes.

Due to Fact 1 and Theorem 3, we can work with the transformed functions in a similar way as with the original ones: In order to prove that two circuits C and C' are functionally equivalent, it suffices to show that the reduced TBDD–representations T_C and $T_{C'}$ of f_C and $f_{C'}$ (i.e., the OBDD–representations of $\tau(f_C)$ and $\tau(f_{C'})$) are functionally equivalent (Fact 1.1). Since the mapping on $\mathbb{B}_n$ induced by a cube transformation τ is an automorphism (Fact 1.2), due to Theorem 3, the desired TBDD–representations (i.e., the OBDD's of the transformed functions) can be computed exactly in the same way as described above: One generates τTBDD's for the variables $x_1, \ldots, x_n$ and derives (now within an OBDD–environment) the τTBDD's of f_C and $f_{C'}$ by symbolic simulation of C and C', respectively. Indeed, with exception of the first step this can be done by means of any available OBDD package.

IV. Type–based TBDD's

To give an example of the practicability and importance of the TBDD–concept, we now consider a class of cube transformations that can be handled quite easily – the cube transformations defined via complete types.

A. Cube Transformations Defined by Complete Types

Definition. *A* complete type *σ over X_n is defined like a BDD with two exceptions. First, it has only one sink. Second, on each source-to-sink path, each variable of X_n occurs* exactly *once.*

We note that linear orderings provide special, and very simple structured complete types. A more interesting example is shown in Figure 2. For $n = 7$, it shows a type σ_n.

With the help of complete types, we may define cube transformations: Let σ be a complete type. Then each assignment $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ of X_n defines a uniquely determined source-to-sink path $p_\sigma(a)$. $\sigma_a(i)$ denotes the index of the variable tested on $p_\sigma(a)$ in the i-th position. The cube transformation $\tau_\sigma : \{0,1\}^n \to \{0,1\}^n$ is defined by

$$\tau_\sigma(a_1, \ldots, a_n) = (a_{\sigma_a(1)}, \ldots, a_{\sigma_a(n)}).$$

Proposition 5. *Let σ be a complete type over X_n. Then τ_σ defines a cube transformation of $\{0,1\}^n$.*

Proof. Since σ is a complete type, the mapping τ_σ is fully defined. In order to show that τ_σ is a cube transformation, it suffices to show that τ_σ is injective. Let $a, b \in \{0,1\}^n$ with $\tau_\sigma(a) = \tau_\sigma(b)$, and, hence, $a_{\sigma_a(i)} = b_{\sigma_b(i)}$ for all i, $1 \le i \le n$. In order to prove that τ_σ is injective it suffices to prove $\sigma_a(i) = \sigma_b(i)$ for all i. Since, due to the definition, $\sigma_c(1) = \sigma_d(1)$ for any $c, d \in \{0,1\}^n$ from $a_{\sigma_a(1)} = b_{\sigma_b(1)}$ we get $\sigma_a(2) = \sigma_b(2)$. Next, $a_{\sigma_a(2)} = b_{\sigma_b(2)}$ implies $\sigma_a(3) = \sigma_b(3)$. The assertion now follows by induction. □

Note that any ordering of the variables defines a cube transformation that merely permutes the set of variables. However, in general, complete types define much more sophisticated cube transformations which permute arguments according to their values.

B. *Circuit Verification with Type-based TBDDs*

The usability of TBDD's as a tool for circuit verification was already mentioned in the previous section, where the general approach of symbolic simulation was discussed, too. Now we describe the first phase of this procedure – the transformation of the variables – in the case where the cube transformation is defined by a complete type: Let C be a circuit, and let σ be a complete type over X_n chosen to represent C as a τ_σTBDD. The idea behind the construction of the (reduced) τ_σTBDD's for the variables $x_i \in X_n$ (i.e., OBDD's for the transformed variables) is the following: All nodes below the nodes labeled with x_i are redundant and can be removed. We add a 1-sink as the right and a 0-sink as the left successor of the leaves. Then we relabel the remaining variables with new variables indexed by the respective level number, which is defined as the distance from the source, incremented by 1. Finally, by means of the usual reduction rules, we reduce nodes that have become redundant or equivalent in the course of the construction.

Algorithm 1 presents a pseudocode which implements this idea.

Algorithm 1.
input:
$i,\ 1 \le i \le n$,
σ – a complete type over X_n
output:
τ_σTBDD(x_i) over $Y_n = \{y_1, ..., y_n\}$,
where τ is a cube transformation
induced by σ.
begin
  x_i := *transform_step*$(i, 1, X, \sigma)$;
  clear_mark_below(σ);
end

transform_step(i, r, M, t);
/* a node of a TBDD or a type is represented by a triple (label, then_son, else_son) */
begin
  if $M = \emptyset$ return UNDEFINED;
  if *marked(t)* then return *result(t)*;
  set_mark(t);
  let $t = (x, t_1, t_0)$;
  if $x_i \in M \setminus \{x\}$ then
    f_1 := *transform_step*$(i, r+1, M \setminus \{x\}, t_1)$;
    f_0 := *transform_step*$(i, r+1, M \setminus \{x\}, t_0)$;
    reduce_and_return(y_r, f_1, f_0);
  else if $x_i \neq x$ then return UNDEFINED;
    else *reduce_and_return*$(y_r, 1, 0)$;

end

The complexity analysis of Algorithm 1 yields the following estimation:

Proposition 6. *Let σ be a complete type over X_n, and let P_i, $1 \le i \le n$, be the reduced τ_σTBDD's of the variables. Each P_i has at most size $size(\sigma)$, and can be constructed in linear time and space, with respect to $size(\sigma)$.* □

The algorithm is implemented in TRUST– an environment developed at the University of Trier for BDD–based Boolean function manipulation. For the experiments, the packages of Brace, Rudell, and Bryant [BRB90], and Long were used.

C. *Experiments with HWB-Functions*

The *hidden weighted bit functions* $HWB_n \in \mathbb{B}_n$, $n \in \mathbb{N}$, discussed by Bryant in [Bry91] provide classical examples of those functions which need necessarily exponential size OBDD's. Let $a = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$, and let $wt(a) = \sum_{i=1}^{n} a_i$ be the *weight of* a. Then, $HWB_n(a)$ is defined by

$$HWB_n(a) = \begin{cases} a_{wt(a)} & \text{if } wt(a) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Fact 6. [Bry91] *The OBDD–representation of HWB_n is of exponential size even if we allow any variable ordering.* □

While, due to Fact 6, it is impossible to represent HWB_n in terms of small OBDD's, we experimentally show that there are very small type–based TBDD–representations of HWB_n: In order to do this we start with the complete types σ_n, whose construction is shown in Figure 2, and consider the cube transformation τ_{σ_n} defined by this type. Then, by means of Algorithm 1, we compute the reduced τ_{σ_n}TBDD's of the variables x_i, $1 \le i \le n$ (i.e., the OBDD's of the transformed variables y_i). Now the TBDD-representation of HWB_n can be derived by symbolic simulation of any circuit that computes HWB_n. In our experiments, we have used a circuit that was designed in accordance with that one proposed by Bryant in [Bry91].

Table 1 summarizes the results of a comparison between the sizes of the τ_{σ_n}TBDD-representations of HWB_n, $n = 2^k$, $2 \le k \le 5$, and the sizes of the OBDD–representations obtained with SIS, which gives overwhelming evidence of the computational advantages of TBDD's in comparison with OBDD's. For the sake of completeness, we also give the sizes of the types which play a crucial role in Algorithm 1. (Note that the types have to be present in the memory merely in the phase of constructing the TBDD's for the variables.)

D. *Experiments with Some ISCAS85 Benchmark Circuits*

Table 2 summarizes some experiments with ISCAS benchmark circuits. We compared the sizes of OBDD–representations (generated with respect to the order heuristic used in SIS) and the time needed to generate these OBDD's with the sizes of TBDD-representations

TABLE 1

Comparison of the TBDD–sizes and the OBDD–sizes of the HWB_n function. For sake of completeness, the sizes of the corresponding types σ_n are also shown.

$n = 2^k$	2^2	2^3	2^4	2^5
Size of OBDD (SIS)	4	31	536	58,260
Max. Universe	13	75	701	126,246
Time (s)	0	0	0.4	55.3
Size of Type σ_n	10	50	226	962
Size of σ_nTBDD	1	1	1	1
Max. Universe	21	287	4,739	61,586
Time	0	0	0.3	12.4

TABLE 2

Comparing resources needed for OBDD-representations and for TBDD-representations, respectively, of some ISCAS–benchmark circuits

Circuit	OBDD (SIS)			TBDD (Types of [BGMS94])		
	Size	Univ.	Time	Size	M. Univ.	Time
c1908	23,854	45,159	9.3	11,967	22,244	4.1
c2670	—	—	—	1,013,035	1,177,519	474.4
c3540	—	—	—	107,773	243,265	53.4
				115,764	239,355	60.0
c432	31,291	124,587	11.2	15,753	39,807	3.9
c5315	64,539	85,238	20.0	17,365	19,475	10.6
c6288:						
5672gat	—	—	—	106,937	616,922	245.8
5971gat	—	—	—	269,481	1,597,388	601.4
lower13	—	—	—	381,804	324,067	92.3
c880	24,893	26,172	3.1	8,026	12,404	1.6
s35932	—	—	—	5,708	14,645	203.5
				6,193	11,851	201
s5378	5,487	7,535	9.5	4,144	62,594	15.8
s838.1	15,990	31,538	2.7	893	5,997	0.9
				959	3,731	0.7
s9234.1	—	—	—	20,936	76,155	17.2

(generated with respect to certain type heuristics described in [BGMS94][1]) and the time needed to generate these TBDD's.

E. Experiments with the ISCAS85 Multiplier C6288

As a further practical test for the capabilities of type–based TBDD's, we addressed the problem of representing the multiplier C6288 from the ISCAS85 benchmark circuits in a computer with relatively tight memory and CPU limits. We considered $n \times n$ multipliers, $8 \leq n \leq 13$, derived from the 16×16 Multiplier C6288 by fixing the most significant inputs to 0.

Table 3 shows the results we have obtained experimenting with a very simple complete type τ_c. Since this type was almost a linear type, the obtained improvements were merely moderate. Nevertheless, they show that it is useful in this case, too, to work with TBDD–representations instead of OBDD–representations.

[1] Note that the TBDD's generated with the types of [BGMS94] are generally different from the FBDD's generated by means of these types.

TABLE 3

Comparing resources needed for OBDD-representations and for TBDD-representations for some $n \times n$ multipliers obtained from ISCAS85–benchmark circuit C6288

Multiplier Size	OBDD (SIS) Size	OBDD (SIS) Time	TBDD Size	TBDD Time
8x8	16,696	10.4	11,957	12.7
9x9	51,878	37.6	35,262	44.2
10x10	159,277	131.1	97,518	139.2
11x11	492,740	435.9	287,459	447.9
12x12	1,513,078	1495.3	869,292	1635.1
13x13	—	—	2,652,972	* [a]

[a] Run on different hardware.

Conclusions

We introduce the concept of TBDD's which considerably enlarges the class of Boolean functions that can be efficiently manipulated in terms of small size OBDD's. Instead of working with the OBDD-representation of a function f, TBDD's allow to work with an OBDD-representation of a suited cube transformed version of f. Besides of giving some theoretical insight into the computational power of the TBDD–concept, we investigate in this paper cube transformations which are based on complete types. First, we show that circuits can be symbolically simulated in terms of such TBDD's similarly as in terms of OBDD's. Second, we give some evidence of the practical importance of such TBDD's by presenting constant-size TBDD-representations of the hidden weighted bit functions HWB_n which were proved to have no small OBDD-representations. Finally, we report some promising experimental results with ISCAS85 benchmark circuits including the multiplier circuit C6288. However, in order to be able to make full use of the computational capabilities of type-based TBDD's, more insight into the problem of constructing suited complete types is needed.

Acknowledgments

We would like to thank Jordan Gergov for his very motivating remarks in our seminal discussions about the possibility to compress FBDD's by means of their transformation into OBDD's, where the first ideas of the TBDD–concept arose.

References

[BGMS94] J. Bern, J. Gergov, Ch. Meinel, A. Slobodová: Boolean Manipulation with Free BDDs. First Experimental Results, Proc. of European Design and Test Conference 1994, IEEE Computer Society Press, 200–207, 1994.

[BCMD90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill: Symbolic Model Checking: 10^{20} states and beyond, Proc. of 5th IEEE Symposium on Logic in Computer Science, 46–51, 1990.

[BRB90] K.S. Brace, R.L. Rudell, R.E. Bryant: Efficient Implementation of a BDD–Package, Proc. 27th ACM/IEEE Design Automation Conference, 40–45, 1990.

[Bry86] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput. C-35, 6 (Aug.), 677–691, 1986.

[Bry91] R. E. Bryant: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication, IEEE Trans. Comput. 40, 2 (Feb.), 205–213, 1991.

[Bry92] R. E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (Sep.), 293–318, 1992.

[GM93] J. Gergov, Ch. Meinel: Frontiers of Feasible and Probabilistic Feasible Boolean Manipulation with Branching Programs, Proc. of 10th Annual Symposium on Theoretical Aspects of Computer Science (Feb.), Lecture Notes in Computer Science 665, 576–585, 1993.

[JBFA92] J. Jain, J. Bitner, D. S. Fussel, J. Abraham: Probabilistic Verification of Boolean Functions, Formal Methods in System Design, 1: 63–117, 1992.

[Kar89] K. Karplus: Using if-then-else DAGs for Multi-Level Logic Minimization, Proc. Advanced Research in VLSI, C. Seitz Ed., MIT Press, Cambridge, Mass., 101–118, 1989.

[MWBS88] S. Malik, A. Wang, R. K. Brayton, A. Sangiovanni–Vincentelli: Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment, Proc. of the IEEE International Conference on Computer-Aided Design (Santa Clara, Calif., Nov.), 6–9, 1988.

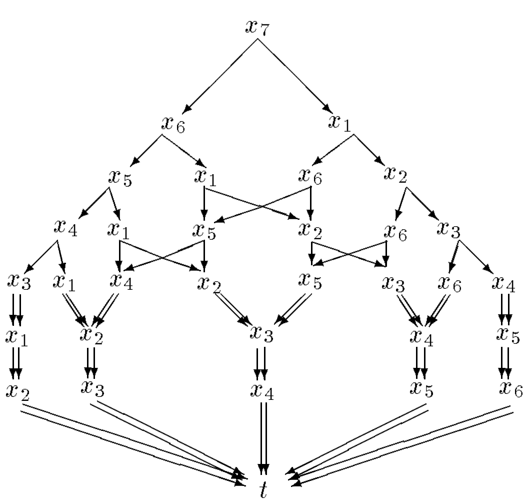


Figure 2. *The complete type σ_n for $n = 7$.*