Algorithms Seminar 1999–2000,
F. Chyzak (ed.), INRIA, (2000), pp. 97–107.

Available online at the URL
http://algo.inria.fr/seminars/.

Synchronous Decision Diagrams: a Data Structure for Representing Finite Sequential Digital Functions

Jean Vuillemin

DMI, École normale supérieure

May 22, 2000

Summary by Philippe Dumas and Philippe Flajolet

Abstract

Binary Diagrams (BDD's) are an important way to represent boolean functions, that is, combinational circuits. Vuillemin proposes Synchronous Decision Diagrams (SDD's) that are capable of representing all causal circuits with finite memory. The framework provides a general basis for the analysis and synthesis of digital circuits. On the mathematical side, it provides unexpected connections between hardware design and the classical notion of automatic sequences in number theory.

Researchers working in circuit theory are concerned with design (given a function, how can it be realized efficiently?) and analysis (what is the function computed by a given circuit?). This talk presents a mathematical framework for the design and analysis of boolean circuits, either combinational (i.e., without memory) or sequential (i.e., with memory). It is superbly elegant as well as conceptually simple. We shall start here with a review of Binary Decision Diagrams (BDD's) that constitute a canonical way to represent boolean functions and serve the purpose of a gentle introduction to the subject. Then, we shall proceed with Synchronous Decision Diagrams (SDD's) that can represent any type of circuit likely to be encountered in practice (i.e., circuits with finite memory of the past whose output does not depend on the future). Due to severe time constraints imposed by the editor of the seminar proceedings,[1] the authors of this summary regret that they cannot do full justice to the work presented and refer to the paper [8] for an introduction to the main ideas.

1. Binary Decision Diagrams

Let $\mathcal{B}$ be the boolean domain $\mathcal{B} = \{0, 1\}$. A boolean function of n variables is a function from $\mathcal{B}^n$ into $\mathcal{B}$. Such a function may be specified by its truth table that is the sequence of its values on its 2^n possible inputs. Let Φ_n be the set of n-ary functions and ϕ_n the corresponding cardinality. Clearly, one has $\phi_n = 2^{2^n}$, hence the identity

$$\phi_{n+1} = 2^{2^{n+1}} \equiv \left(2^{2^n}\right)^2 = (\phi_n)^2 .$$

This trivial identity suggests the existence of a fundamental isomorphism

$$\Phi_{n+1} \simeq \Phi_n \times \Phi_n.$$

[1] *Editor's Note.* I acknowledge the promptitude of the authors of the summary. Especially their promptitude to renegociate deadlines.

Indeed any boolean function $f(x_1, \ldots, x_n, x_{n+1})$ with $f \in \Phi_{n+1}$ can be specified by a pair (f_0, f_1), where $f_0 \in \Phi_n$ and $f_1 \in \Phi_n$ are "specializations" of f,

$$f_0(x_1, \ldots, x_n) := f(x_1, \ldots, x_n, 0), \qquad f_1(x_1, \ldots, x_n) := f(x_1, \ldots, x_n, 1).$$

Consequently, when the decomposition is iterated, any boolean function of n variables becomes representable as a perfect binary tree, the *binary decision tree* $\mathrm{bdt}(f)$, whose height is n, whose internal nodes correspond to partial specializations of f, and whose external nodes are either the constant function 0 or the constant function 1. Observe that reading the $\mathcal{B}$ labels of the extrenal nodes of $\mathrm{bdt}(f)$ from left to right produces precisely the truth table of f.

The *binary decision diagram*[2] of f, $\mathrm{bdd}(f)$, is then nothing but the directed acyclic graph (dag) representation of this tree obtained in the usual way by sharing repeated subtrees and representing them only once. It is classically known that such a dag representation of a tree of size N can always be constructed in time $O(N)$; see for instance [5] for a discussion. Here, one has $N = 2^n$ for functions in Φ_n, so that the sharing algorithm approach is of exponential time complexity when f is given by its truth table or, equivalently, by its tree $\mathrm{bdt}(f)$. In many cases, fortunately, one can operate with polynomial time complexity.

Here is an example. Consider the adder function on three variables,

$$f(a, b, c) = a \oplus b \oplus c.$$

We purposely refrain from operating with the truth-table description of f in order to emphasize that BDD's are directly accessible via a symbolic calculus on boolean functions. Here, two "subfunctions" are first obtained upon setting the variable c to either 0 or 1:

$$f_0(a, b) = a \oplus b, \qquad f_1(a, b) = a \oplus b \oplus 1.$$

Next, specialize b, which yields here *only two* (and not four!) distinct functions, namely,

$$f_{00}(a) = a, \qquad f_{10}(a) = a \oplus 1 \qquad (\text{with } f_{01} \equiv f_{10} \text{ and } f_{11} \equiv f_{00}).$$

Finally, specialize a, which eventually leads to a reduction to the two constant functions

$$f_{000}() = 0, \qquad f_{010}() = 1 \qquad (\text{with } f_{100} \equiv f_{010} \text{ and } f_{110} \equiv f_{000}).$$

This example shows, more generally, that the BDD of the n-fold adder $f(x_1, \ldots, x_n)$ can be determined in time linear in n via basic boolean algebra alone, this despite the fact that the truth table has size 2^n. The construction in the case of the function $f(a, b, c) = a \oplus b \oplus c$ is described in Figure 1.

Bryant has invented the BDD concept in 1986 (see [1, 2]). The BDD of an n-ary function can often be computed in time much less than $O(2^n)$ (cf. the adder example), since it captures the regularities that are likely to be present in most functions occurring in practice.[3] Also, given the BDD's of f and g it is possible, in low polynomial time, to determine BDD's for various compositions of f and g like $f \oplus g$, $f \circ g$, etc. Finally, once an ordering on variables has been fixed,[4] the BDD

[2]The BDD's described here are sometimes called *OBDD's*, where the 'O' stands for "ordered" and refers to a fixed ordering on boolean variables.

[3]In the worst case, a BDD contains up to $O(2^n/n)$ nodes. A similar bound [6] even holds on average, for a random boolean function. Such properties are also related to a famous theorem of Shannon and Muller [7, p. 763] to the effect that almost all boolean functions have minimal circuit complexity of the order of $2^n/n$. This theoretical discussion is however to be counterbalanced by the fact that functions destined to be realized in silicon are seldom chosen at random!

[4]The structure and size of a BDD depends on the ordering of variables. Several heuristics have been developed in order to try and come up with "good" orders.

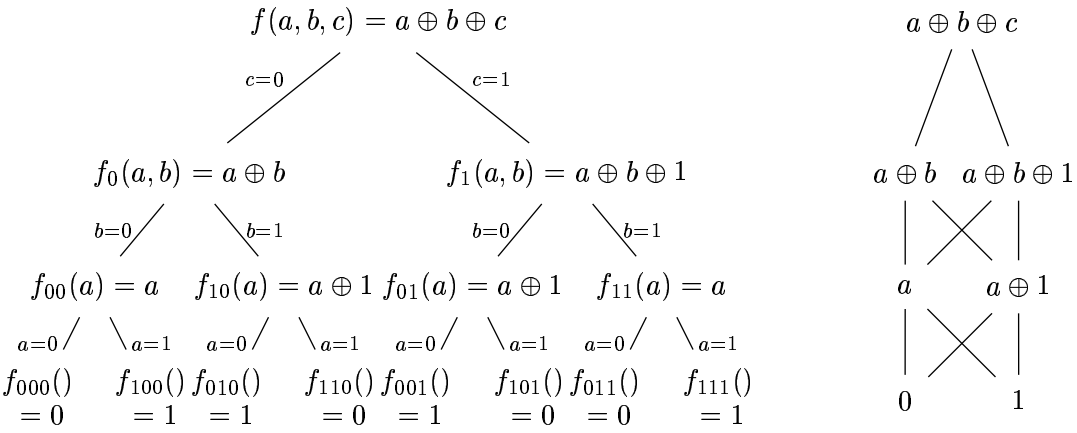


FIGURE 1. The adder function, $f(a,b,c) = a \oplus b \oplus c$: its Binary Decision Tree (left) and its Binary Decision Diagram (right).

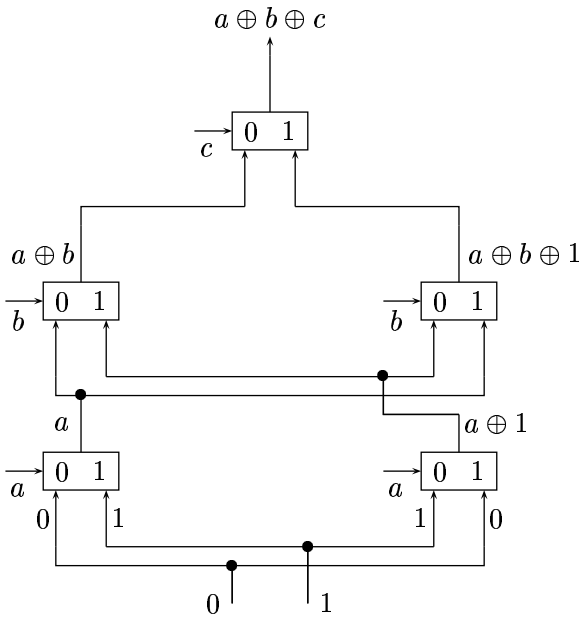


FIGURE 2. A realization of the adder function based on the BDD representation and multiplexers.

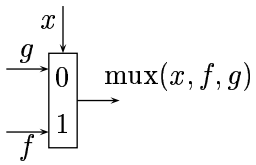
becomes a canonical representation of the function it represents, so that equivalence of boolean functions becomes decidable in time linear in the sizes of the compact BDD representations. In particular, this observation makes it possible to compare any combinational circuit design against a canonical specification (the "semantics" of the function) in a computationally efficient manner. This constitutes one of the powerful implications of the BDD concept.

Finally, we mention that once the BDD form of a boolean function has been obtained, a circuit realization of proportional size is immediate: all that is needed is "Shannon's switch" also known

as "multiplexer,"

$$\mathrm{mux}\,(x,f,g) := \text{'}\mathbf{if}\ x\ \mathbf{then}\ f\ \mathbf{else}\ g\text{'} = (x \wedge f) \vee (\overline{x} \wedge g),$$

together with entries grounded at 0 and 1. A diagrammatic representation is as follows:



The way the BDD of the adder function "compiles" into a circuit based on multiplexers is displayed in Figure 2.

2. Polynomial Representations of BDD's

As a preparation for the treatment of synchronous decision diagrams, we now introduce a representation of boolean functions by means of univariate polynomials with coefficients in the binary field $\mathbb{F}_2$. Let f be a boolean function in n variables. Its *truth-table polynomial* $F = \mathbf{T}f$ is defined as follows: interpret each n-tuple $(x_1, x_2, \ldots, x_m)$ of boolean values as the binary representation of an integer,

$$\beta(x_1,\ldots,x_n) := (x_1x_2\ldots x_n)_2 = x_1 2^{n-1} + \cdots + x_n,$$

(observe the convention that lower order bits are on the right), and set

$$\mathbf{T}f(z) = \sum_{x_1,\ldots,x_n \in \mathcal{B}} f(x_1,x_2,\ldots,x_n) z^{\beta(x_1,\ldots,x_n)}.$$

For instance the adder function $f(a,b,c) = a \oplus b \oplus c$ has the standard truth table

$x_1x_2x_3$	000	001	010	011	100	101	110	111
$\beta(x_1,x_2,x_3)$	0	1	2	3	4	5	6	7
$f(x_1,x_2,x_3)$	0	1	1	0	1	0	0	1

so that its truth-table polynomial is

$$\mathbf{T}f(z) = z + z^2 + z^4 + z^7.$$

The BDD algorithm is amenable to interpretation in this formalism. Define the two "sectioning" operators on polynomials $\mathbb{F}_2[z]$ by

$$S_0\Big(\sum_k f_k z^k\Big) = \sum_k f_{2k} z^k, \qquad S_1\Big(\sum_k f_k z^k\Big) = \sum_k f_{2k+1} z^k.$$

(The definition is also valid for power series of $\mathbb{F}_2[[z]]$, a fact to be used later.) The specialization of the last bit in a function $f(x_1,\ldots,x_n)$ is then seen to be isomorphic to sectioning. Indeed, a simple calculation shows that

$$S_0\mathbf{T}\big(f(x_1,\ldots,x_{n-1},x_n)\big) = \mathbf{T}\big(f(x_1,\ldots,x_{n-1},0)\big)$$
$$S_1\mathbf{T}\big(f(x_1,\ldots,x_{n-1},x_n)\big) = \mathbf{T}\big(f(x_1,\ldots,x_{n-1},1)\big).$$

Consequently, the BDD construction can be regarded as being equivalent to decomposing a polynomial by means of S_0, S_1 until an eventual reduction to the constants 0 and 1 is attained. In this framework, the BDD algorithm applied to the adder example corresponds to the tree and the diagram of Figure 3.

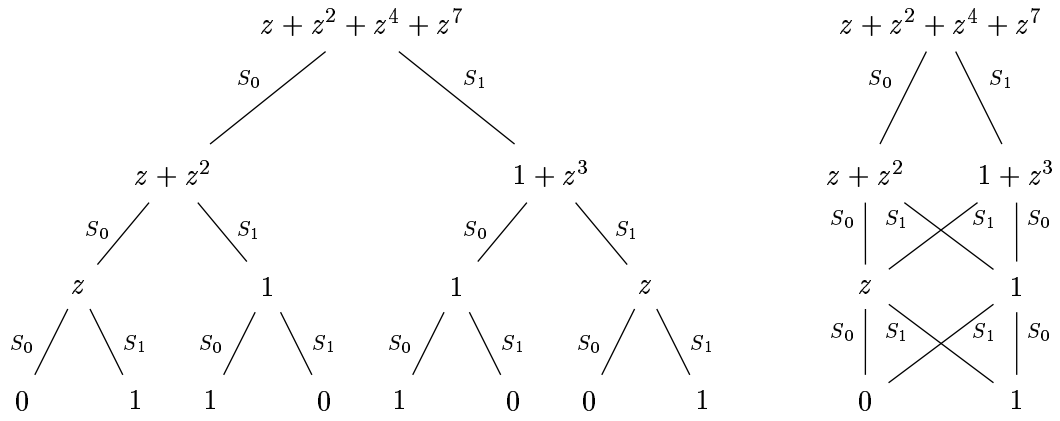


FIGURE 3. Polynomial representations of the Binary Decision Tree and the Binary Decision Diagram of the ternary adder.

3. Synchronous Decision Diagrams

In all generality, a *sequential function* maps infinite sequences of binary inputs into infinite sequences of binary outputs. It thus takes as input a "stream" of bits $(x_t)_{t\geq 0}$ and produces another "stream" $(y_t)_{t\geq 0}$. In other words, a sequential function is a mapping from $\mathcal{B}^\infty$ to $\mathcal{B}^\infty$. For practical purposes, additional constraints must clearly be imposed on the sequential functions considered.

First, we say that a function f from $\mathcal{B}^\infty$ to $\mathcal{B}^\infty$ is *causal* when the output at time t depends exclusively upon the input values from times 0 through t. In what follows, only causal functions are considered. (For the mathematically inclined reader, we note that causal functions are particular continuous functions on the set $\mathcal{B}^\infty$ endowed with the topology induced by the metric $d(a,b) = 2^{-\min\{\, t \mid a_t \neq b_t \,\}}$.)

For f causal, we let f_t be the output at time t:

$$y_t = f_t(x_0, \ldots, x_t).$$

By analogy with the specialization of combinational functions, we define the *predictors*, $\varpi_0 f$ and $\varpi_1 f$, by the properties:

$$(\varpi_0 f)_{t+1} = f_t(x_0, \ldots, x_t, 0), \qquad (\varpi_1 f)_{t+1} = f_t(x_0, \ldots, x_t, 1).$$

These predictors tabulate which value of f will be taken when the input bit to arrive next is specialized to 0 or 1. For $b_0 \ldots b_r$ a sequence of bits, we then have the (generalized) predictor of order $r+1$,

$$\varpi_{b_0 \ldots b_r} f = \varpi_{b_r} \ldots \varpi_{b_0} f.$$

By infinite iteration, we can then construct the *synchronous decision tree* (SDT) denoted by $\mathrm{sdt}(f)$ as the tree where the nodes are the quantities $\nu = \varpi_w(f)$ and the descendents of node ν are $\varpi_0(\nu), \varpi_1(\nu)$. The tree $\mathrm{sdt}(f)$ can be realized by an infinite tree circuit using only multiplexers and registers (i.e., circuits capable of storing one binary value), much in the same way as combinational circuits are realized by finite tree circuits. See Figure 4 for an illustration.

Next, in order to be computable by some physical device, a digital function must be causal, but also representable by some finite system. To formalize this, we introduce the notion of *on-line computable function*: by this is meant a function such that the collection of all predictors of all

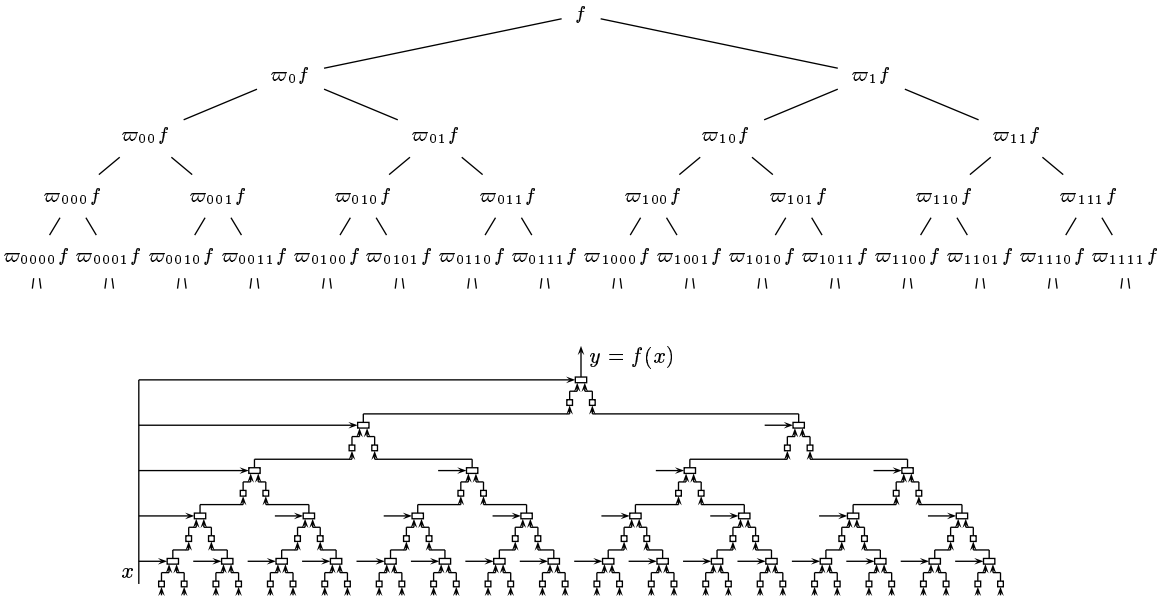


FIGURE 4. The infinite synchronous decision tree (top) and its circuit realization (bottom).

orders forms a finite set. In this case, the (infinite) tree can be converted to a (finite) graph[5] by identifying nodes of the SDT associated with functions that are equal. The resulting graph is called the *synchronous decision diagram* (SDD) and it is obtained by a simple algorithm: *(i)* build the infinite SDT for f; *(ii)* systematically share all the subexpressions generated during this process. (Optionally, one may also consider functions f, g to be isomorphic if either $f = g$ or $f = \neg g$; in that case the SDD will also involve logical negation gates but will be more compact.)

When presented as above, the SDD algorithm looks like an infinite process. However, it can be seen [8] that if a function is realizable by a finite transducer (i.e., an automaton with output), then the SDD algorithm terminates in finite time. In fact, the SDD algorithm provides an integrated alternative to the classical design of sequential circuits.[6]

In order to illustrate the SDD concept, we apply it now to the design of a circuit that takes as input a stream of bits (x_t) meant to represent the real number $\xi = \sum_{t\geq 0} x_t 2^{-t}$ and produces as output the stream (y_t) where the real number $\eta = \sum_{t\geq 0} y_t 2^{-t}$ satisfies $\eta = (1/3)\ \xi$. Introduce the integers

$$x_{(t)} = 2^t \sum_{s\leq t} \frac{x_s}{2^s}, \quad y_{(t)} = 2^t \sum_{s\leq t} \frac{y_s}{2^s}, \quad t \geq 0,$$

and the carry r_t defined by

$$x_{(t)} = 3y_{(t)} + r_t, \quad 0 \leq r_t < 3, \quad t \geq 0.$$

An easy calculation that mimics high school arithmetics yields

$$2r_t + x_{t+1} = 3y_{t+1} + r_{t+1}, \quad t \geq 0.$$

[5]Observe that, as opposed to the case of combinational circuits, the corresponding graph is no longer acyclic since nodes at different levels in the tree may be collapsed.

[6]A classical construction starts from a specification of a finite automaton and stores the current state of the automaton in binary registers while realizing the transition function by means of a combinational circuit (itself possibly optimized via BDD's).

These formulæ show that the function $\eta = \xi/3$ is causal and that the bit $y_{t+1} = f_{t+1}(x_0, \ldots, x_{t+1})$ depends only on the last input bit together with the "carry" r_t that is inherited from past history. The carry can only assume three values and accordingly the number of predictors is finite, to the effect that the SDT has only six nodes. Thus, f is on-line computable. Figure 5 shows the result of the construction. (In the diagram, a transition denoted by α/β is triggered by reading the bit α and results in producing the bit β.)

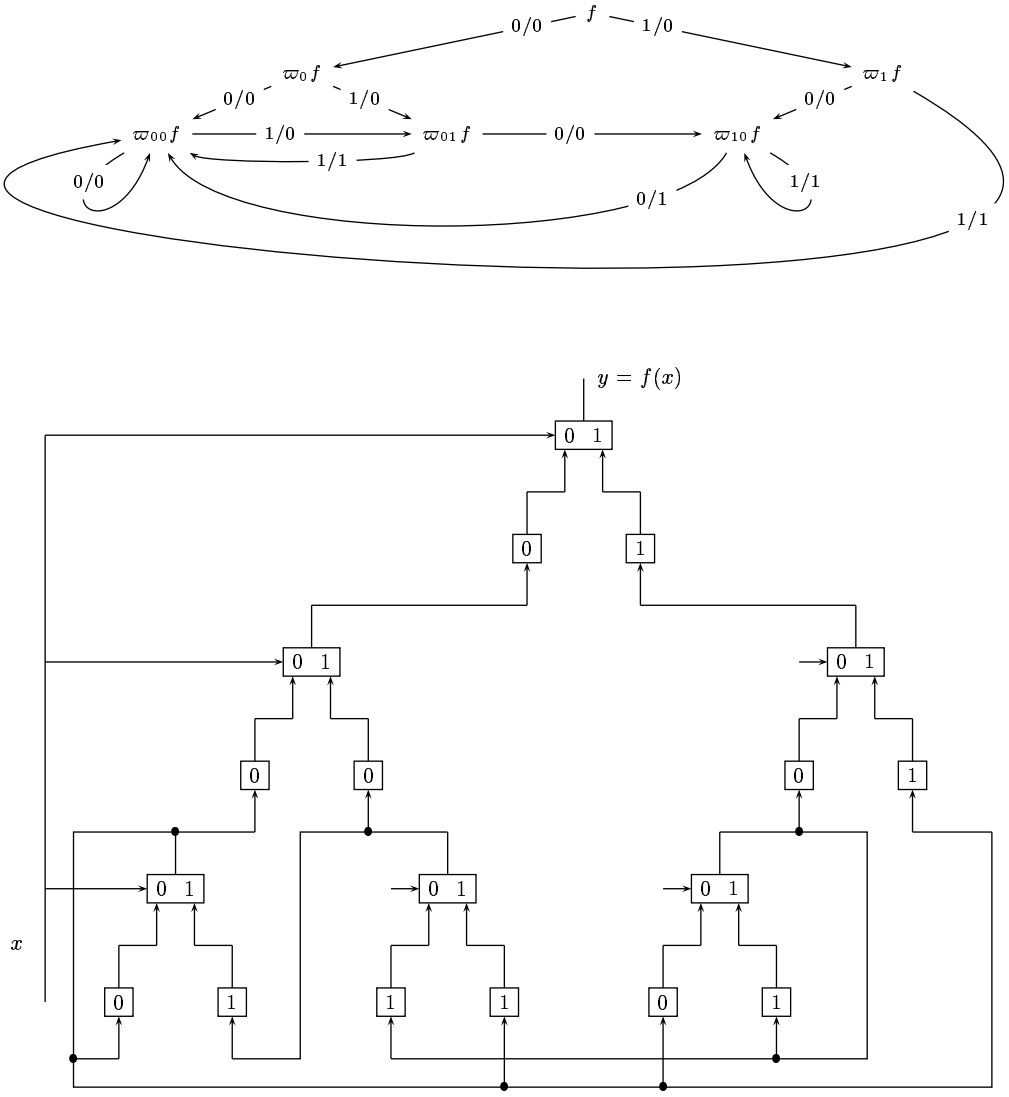


FIGURE 5. The '(1/3) ξ' function: its abstract SDD representation (top) and the circuit realization (bottom).

4. Formal Power Series Representations of SDD's

A causal function f is characterized by its truth table. This is the power series representation $\mathbf{T}f$, an element of $\mathbb{F}_2[[z]]$ defined as

$$\mathbf{T}f(z) := \sum_{t\geq 0} \sum_{x_0,\ldots,x_t \in \mathcal{B}} f_t(x_0,\ldots,x_t) z^{\beta(1x_0\ldots x_t)-2}.$$

(The correction of -2 in the exponent is a convenience chosen to ensure that exponents start at 0.) We shall refer to $\mathbf{T}f$ as the *truth-table representation* of f. This notion extends in a natural way the corresponding definition for combinational functions. Indeed, an alternative definition of $\mathbf{T}f$ for causal functions is as follows: take F_t as the truth table of f_t in "listed" form, and build the truth table of f in "listed" form by

$$F_0F_1F_2\ldots = \big[f_0(0)f_0(1)\big]\big[f_1(00)f_1(01)f_1(10)f_1(11)\big]\big[f_2(000)f_2(001)\ldots f_2(111)\big]\ldots;$$

then $F(z) = \mathbf{T}f(z)$ satisfies a sort of a "generating function relation,"

$$F(z) = \big[f_0(0) + zf_0(1)\big] + z^2\big[f_1(00) + zf_1(01) + z^2f_1(10) + z^3f_1(11)\big] + z^6\big[f_2(000) + zf_2(001) + \cdots + z^7f_2(111)\big] + \cdots,$$

so that there is a simple relation between truth tables of combinational functions and of sequential functions:

$$F(z) = \sum_{t\geq 0} z^{2^{t+1}-2}\mathbf{T}f_t(z).$$

Equipped with these definitions, we observe the action of sections,

$$S_0F(z) = \big[f_0(0)\big] + z\big[f_1(00) + zf_1(10)\big] + z^3\big[f_2(000) + zf_2(010) + \cdots + z^3f_2(110)\big] + \cdots,$$

$$S_1F(z) = \big[f_0(1)\big] + z\big[f_1(01) + zf_1(11)\big] + z^3\big[f_2(001) + zf_2(011) + \cdots + z^3f_2(111)\big] + \cdots,$$

which entails

$$S_0F(z) = \sum_{t\geq 0} z^{2^t-1}S_0F_t(z) = f_0(0) + z\sum_{t\geq 0} z^{2^{t+1}-2}S_0F_{t+1}(z),$$

$$S_1F(z) = \sum_{t\geq 0} z^{2^t-1}S_1F_t(z) = f_0(1) + z\sum_{t\geq 0} z^{2^{t+1}-2}S_1F_{t+1}(z).$$

This provides a direct relation between the sections of the truth table of any causal f and the predictors of f, namely,

$$S_0(\mathbf{T}f)(z) = f_0(0) + z\mathbf{T}(\varpi_0 f)(z), \qquad S_1(\mathbf{T}f)(z) = f_0(1) + zT(\varpi_1 f)(z).$$

Now, by definition, f is on-line computable when its predictors lie in a finite set. The equation above shows that this is equivalent to the finiteness of vector space over $\mathbb{F}_2(z)$ of all the (iterated) sections of the truth table. The connection is thereby established with what is otherwise known as automatic series;[7] see the foundational paper by Christol *et al.* [3], Dumas's thesis [4], and several summaries in previous issues of the Algorithms Seminar Proceedings. We state:

Theorem 1. *The truth table* $\mathbf{T}f$ *of an online computable function* f *is a 2-automatic series. Consequently, it is an algebraic function over the field* $\mathbb{F}_2(z)$.

[7] A sequence is defined to be automatic if its nth element is produced by a finite transducer applied to the binary representation of n; a series is automatic if its sequence of coefficients is automatic. Equivalent characterizations of automatic series are as algebraic elements over $\mathbb{F}_2(z)$ or as solutions to Mahlerian equations; refer to [3, 4].

Each causal finite function f may thus be represented by a bivariate characteristic polynomial $P(z,y)$ so that the truth table $\mathbf{T}f$ is the only root $y \in \mathbb{F}_2[[z]]$ of $P(z,y) = 0$. Conceivably, this theorem opens an avenue to circuit design and verification by means of polynomial elimination algorithms—typically, Gröbner bases. Given the superexponential complexity of algebraic elimination, it seems however to the authors of the summary that a direct approach based on linear algebra (in accordance with standard techniques of 2-automatic series [4]) should yield decision procedures of lower complexity.

5. From Circuits to Functions

In this section, we show how to put to good use the formalism introduced above in order to analyse circuits: starting from a given circuits, the goal is to determine a mathematical specification of what it does. Note that the dual problem of synthesis has been already implicitly tackled on the occasion of the "one-third" function ($\xi \mapsto \xi/3$).

Let us first consider a circuit that takes as input a stream of bits (g_t) and produces the stream (h_t) which is the same stream delayed by 1 in time. In other words, we have $h_0 = z_0$ (the initialization value) and $h_t = g_{t-1}$ for $t \geq 1$. In the context of a finite circuit, the values h_t are described by their truth table and they depend on the global input sequence $x = (x_t)_{t\geq 0}$ of the circuit. Thus, in terms of the finite boolean functions $g_t(x_0, \ldots, x_t)$ and $h_t(x_0, \ldots, x_t)$, we have

$$h_0 = z_0, \quad h_t(x_0, \ldots, x_t) = g_{t-1}(x_0, \ldots, x_{t-1}) \quad \text{for } t \geq 1.$$

This relation translates into a relation between the truth tables of the input (G) and the output (H) of the register,

$$H(z) = (1+z)\big(z_0 + z^2 G(z^2)\big). \tag{1}$$

Thus, in the formal power series representation, a register operates by way of the "Mahlerian operator," $G(z) \mapsto G(z^2)$.

Consider next a multiplexer that takes as input two streams of bits $a(x)$ and $b(x)$ (themselves causal functions of the input stream x) and assume that control is achieved by the input stream x. The output $m(x)$ is a causal function defined by

$$m_t(x_0, \ldots, x_t) = \mathrm{mux}\big(x_t, a_t(x_0, \ldots, x_t), b_t(x_0, \ldots, x_t)\big),$$

which we abbreviate as

$$m(x) = \mathrm{mux}\big(x, a(x), b(x)\big).$$

A little reflection shows that the truth table of m is obtained by suitably merging the truth tables of a and b as follows

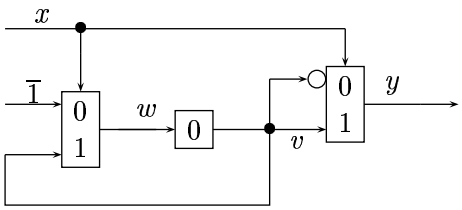
A	$a_0(0)$	$a_0(1)$	$a_1(00)$	$a_1(01)$	$a_1(10)$	$a_1(11)$	$a_2(000)$	$a_2(001)$	$\ldots$
B	$b_0(0)$	$b_0(1)$	$b_1(00)$	$b_1(01)$	$b_1(10)$	$b_1(11)$	$b_2(000)$	$b_2(001)$	$\ldots$
M	$b_0(0)$	$a_0(1)$	$b_1(00)$	$a_1(01)$	$b_1(10)$	$a_1(11)$	$b_2(000)$	$a_2(001)$	$\ldots$.

This relation translates into

$$M(z) = (S_0 B)(z^2) + z(S_1 A)(z^2), \tag{2}$$

which now involves a blend of sectioning and Mahlerian operators.

Now, a finite circuit can be translated into a system of fixed-point equations: to each entity is associated its truth table; then relations (1) and (2) (used repeatedly) provide the system of equations. Here is an application to a circuit discussed in [8]. This circuit comprises one inverter (represented by a circle), two multiplexers, and one register that is initially set at 0. The upper entry of the leftmost multiplexer receives a continuous stream of 1's which is represented by $\overline{1}$.



What is required is to verify that the circuit computes the function

$$\xi \mapsto \eta = \xi + 1,$$

where the input and output streams are now interpreted as dyadic numbers, that is

$$\xi = \sum_{t\geq 0} x_t 2^t, \quad \eta = \sum_{t\geq 0} y_t 2^t, \quad x, y \in \mathbb{Z}_2.$$

To each of the flows, y, v, w, one associates its truth table, respectively $Y(z), V(z), W(z)$. Given the rules (1) and (2), the structural description of the circuit is translated (compiled!) into the system of equations:

$$\begin{aligned}
Y(z) &= \left(S_0\left(\frac{1}{1-z}+V\right)\right)(z^2) + z\,(S_1V)\,(z^2) = \frac{1}{1-z^2} + (S_0V)\,(z^2) + z\,(S_1V)\,(z^2) \\
&= \frac{1}{1-z^2} + V(z), \\
V(z) &= (1+z)z^2W(z^2), \\
W(z) &= \left(S_0\frac{1}{1-z}\right)(z^2) + z\,(S_1V)\,(z^2) = \frac{1}{1-z^2} + z\,(S_1V)\,(z^2).
\end{aligned}$$

In order to understand the function computed by the circuit, we proceed to solve this system. The second equation provides $S_1V(z) = zW(z)$, a relation that, when carried into the third equation, gives:

$$W(z) = \frac{1}{1-z^2} + z^3W(z^2).$$

Such a functional equation is now easily solved by iteration,

$$W(z) = \sum_{k=0}^{+\infty} \frac{z^{3(2^k-1)}}{1-z^{2^{k+1}}},$$

and this form entails in turn

$$\begin{aligned}
Y(z) = \sum_{k=0}^{+\infty} \frac{z^{3\cdot 2^k-1}}{1-z^{2^{k+1}}} + \sum_{k=0}^{+\infty} \frac{z^{3\cdot 2^k}}{1-z^{2^{k+1}}} \\
= \big[1\big] + z^2\big[z+z^2\big] + z^6\big[z+z^3+z^5+z^6\big] + z^{14}\big[z+\cdots\big] + \cdots.
\end{aligned}$$

From there, it is an easy exercise (left to the reader) to check that the truth table $Y(z)$ is equal to the truth table corresponding to the dyadic function $\xi \mapsto \xi + 1$.

6. Conclusion

Due to constraints already evoked, we could only scratch the surface in this brief[8] seminar summary. The point of view developed in the talk bases itself further on the existence of isomorphisms between various domains. For instance, as we have seen, the boolean domain may be viewed as $\mathcal{B}$ or $\mathbb{F}_2$; boolean functions are representable as elements of $\mathbb{F}_2[z]$; on-line computable functions are equivalent to algebraic elements of $\mathbb{F}_2[[z]]$ and to 2-automatic series. There exist several other interesting connections, for instance, with the ring of dyadic integers $\mathbb{Z}_2$ that form one of the conceptual basis of the original paper [8]. Such isomorphisms do increase the expressive power of the SDD formalism that we have opted to develop here only over $\mathcal{B}$ while making use of representations in $\mathbb{F}_2[[z]]$.

There is also great practical potential in the algorithms associated with the SDD concept. Quoting from Vuillemin: *The SDD of f is a cyclic data structure, which represents the minimal finite state machine for f. In the worst case, its size is doubly exponential in the size of f. However, efficient algorithms exist to operate on the SDD representations with the following characteristics: constant time*[9] *for $f(\lambda x)$, $f(1+\lambda x)$; linear time for $\neg f(x)$, $\lambda f(x)$, the inverse $g\big(f(x)\big) = x$, and the fixed point $y = \lambda g(x,y)$; quadratic time for the composition $f\big(g(x)\big)$ and for boolean operations, $f(x) \wedge f(y)$, etc; cubic time for the more general composition $f\big(g(x),h(x)\big)$. The SDD opens an approach to sequential circuit synthesis and verification whose implementation is straightforward in a high-level language, and which can cope automatically with synchronous circuits of limited size.*

Bibliography

[1] Bryant (Randal E.). – Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, vol. C-35, n° 8, 1986, pp. 679–691.

[2] Bryant (Randal E.). – Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, vol. 24, n° 3, 1992, pp. 293–318.

[3] Christol (G.), Kamae (T.), Mendès France (M.), and Rauzy (G.). – Suites algébriques, automates et substitutions. *Bulletin de la Société Mathématique de France*, vol. 108, n° 4, 1980, pp. 401–419.

[4] Dumas (Philippe). – *Récurrences mahlériennes, suites automatiques, études asymptotiques.* – Institut National de Recherche en Informatique et en Automatique, Rocquencourt, 1993, 241p. Thèse, Université de Bordeaux I, Talence, 1993.

[5] Flajolet (Philippe), Sipala (Paolo), and Steyaert (Jean-Marc). – Analytic variations on the common subexpression problem. In Paterson (M. S.) (editor), *Automata, languages and programming. Lecture Notes in Computer Science*, vol. 443, pp. 220–234. – Springer, New York, 1990. Proceedings of the 17th ICALP Conference, Warwick, July 1990.

[6] Liaw (Heh Tyan) and Lin (Chen Shang). – On the OBDD-representation of general boolean functions. *IEEE Transactions on Computers*, vol. 41, n° 6, 1992, pp. 661–664.

[7] van Leeuwen (Jan) (editor). – *Handbook of theoretical computer science. Vol. A.* – Elsevier Science Publishers, Amsterdam, 1990, x+996p. Algorithms and complexity.

[8] Vuillemin (Jean E.). – On circuits and numbers. *IEEE Transactions on Computers*, vol. 43, n° 8, 1994, pp. 868–879.

[8] *Editor's Note.* I wish to express my true gratitude to the authors of the summary for not exceeding three times the expected length of a typical summary.

[9] Dyadic interpretations ($\mathbb{Z}_2$) are understood in the first two examples.