Playing with Boolean Blocks: Post's Lattice with Applications to Complexity Theory[1]

Elmar Böhler[2], *Nadia Creignou*[3], *Steffen Reith*[4], *and Heribert Vollmer*[5]

Introduction

Let us imagine children playing with a box containing a large number of building blocks such as LEGO™, fischertechnik®, Polydron, or something similar. Each block belongs to a certain class (given by, e. g., color, shape, or size) and usually the number of different such classes is relatively small. It is amazing to see how involved the constructions are that can be built by the kids. From an abstract point of view, one might ask if there is something similar to all objects that may be built using only blocks from some given classes, or if a given object (i.e., given by some description) can be constructed using the available blocks. These questions become interesting from a theoretical point of view if the number of different types or classes of blocks is finite, but from each type of block we have an infinite number of "copies" available for construction.

In this and the next part [BCRV04] of this two-article survey, we will play with Boolean blocks. In the present part we will mainly consider Boolean functions as basic blocks. We want to use them as gates in the construction of Boolean circuits. Thus, the way of building blocks together here is the operation of *soldering*. In the next issue we will use Boolean constraints, i. e., Boolean relations, to construct so called conjunctive queries to a database.

To be a bit more formal, let B be a set of Boolean functions. By $[B]$ we denote the class of all Boolean functions that can be obtained from functions out of $B \cup \{id\}$ by soldering them together, i. e., $[B]$ consists of all arbitrary compositions of functions from $B \cup \{id\}$. Here, id denotes the unary identity, $id(x) = x$. We will see that for certain "syntactical" reasons we always have to include the identity. Thus, $[B]$ consists of all functions of $B \cup \{id\}$ plus those, that can be obtained by the following rule: If $f(x_1, \ldots, x_n) \in [B]$ and $X_1, \ldots, X_n$ are either Boolean variables or elements from $[B]$, then $f(X_1, \ldots, X_n) \in [B]$, too. Note that we are able to permute and identify variables of a Boolean function by this operation. We say that a class B of Boolean functions is *closed* if $[B] = B$ which means that we cannot generate new Boolean functions by composing functions out of B.

It is interesting to note that $[B]$ is exactly the class of Boolean functions that can be computed by circuits with gates from B, or, equivalently, can be defined by propositional formulas with connectives from B. We will come back to this observation in Sect. 1.2.

In the forties of the previous century, Emil Post obtained a complete list of all closed classes of Boolean functions, nowadays called *Post's lattice*, and moreover, he proved that each of them has a finite basis and obtained a list of bases for all closed classes [Pos41] (see Figs. 1 and 2). Closed classes are also referred to as *clones*, a term used in universal algebra. We will see in the second part that the substantial clone theory can help us to obtain elegant and short proofs for results

 Supported in part by ÉGIDE 05835SH, DAAD D/0205776 and DFG VO 630/5-1.

[2]Theoretische Informatik, Fachbereich Mathematik und Informatik, Universität Würzburg, Am Hubland, 97072 Würzburg, GERMANY, `boehler@informatik.uni-wuerzburg.de`.

[3]Laboratoire d'informatique fondamentale, Faculté des sciences de Luminy, Université de la Méditerranée, 163 avenue de Luminy, 13288 Marseille cedex 9, FRANCE, `creignou@lidil.univ-mrs.fr`

[4]Lengfelderstr. 35b, 97078 Würzburg, GERMANY, `streit@streit.cc`.

[5]Theoretische Informatik, Fachbereich Informatik, Universität Hannover, Appelstraße 4, 30167 Hannover, GERMANY, `vollmer@thi.uni-hannover.de`.

concerning constraint satisfaction problems. In the present part I we will use Post's lattice as a tool in complexity examinations of Boolean circuits and propositional formulas.

A description of Post's lattice (without a proof of the classification, but a proof of the finite basis theorem) can be found in [Pip97, Chap. 1]. A complete and self-contained presentation of Post's result is given in [JGK70]. Different and much shorter proofs can be found in [Ugo88, Zve00].

As just mentioned, Post's lattice can be a very helpful tool for complexity studies of Boolean circuits and propositional formulas. Before turning to a detailed discussion of the lattice in Sect. 1 we want to show the reader a glimpse of what is ahead in Sect. 2 where we turn to applications in complexity theory. The famous Cook-Levin-Theorem states that the problem, given a propositional formula or circuit to determine if it is satisfiable, is NP-complete [Coo71, Lev73]. A question that arises immediately is if there are subclasses of propositional formulas for which the satisfiability problem is easier, e. g., efficiently solvable. One direction one might pursue is to restrict the types of Boolean connectives allowed. Instead of the usual formulas or circuits over $\{\wedge, \vee, \neg\}$ one might for example restrict oneself to $\{\wedge, \oplus\}$. Since $[B]$ is the class of those functions that can be defined using propositional formulas with allowed Boolean connectives taken from B, this question is most naturally studied in connection to Post's lattice: Allowing connectives from B is the same as allowing connectives from $[B]$. Lewis in 1979 [Lew79] obtained the remarkable result that the satisfiability of propositional formulas is NP-complete if and only if the connective $x \wedge \neg y$ can be represented (in Post's lattice, this corresponds to the class S_1). We will give a very short proof of Lewis' result in Sect. 2. Coming back to the above example set $\{\wedge, \oplus\}$, satisfiability turns out to be again NP-complete, since, as we will see, $[\{\wedge, \oplus\}]$ is the class R_0 in Post's lattice, a superclass of S_1.

1 Closed Classes of Boolean Functions: Post's Lattice

Boolean circuits and Boolean functions attract and deserve a lot of attention in computer science, and the theory behind them is exhaustively used in circuit design and various other important fields. We will have a look at superposition, i. e., the mathematical operations that correspond to operations carried out when soldering Boolean circuits. Whether or not a Boolean function can be described by a Boolean circuit depends solely on the sort of gates one is allowed to use in the construction of the circuit. There are classes of Boolean functions that are closed under superposition: In this section we will discuss these and the result from E. L. Post [Pos41] who identified and characterized each of them.

1.1 Boolean Functions

An *n-ary Boolean function* is a function from $\{0,1\}^n$ to $\{0,1\}$. There are several ways to describe an n-ary Boolean function f. One is to explicitly specify for each n-tuple $a_1, \ldots, a_n$ the function value $f(a_1, \ldots, a_n)$, i. e., to construct a lookup table for all possible inputs of the function, the so called *truth-table*. This form of representation is often very inconvenient, since the truth table has exponentially many (exactly 2^n) entries.

Since we have binary function values, every Boolean function also defines an n-ary relation: The set of all n-tuples α with $f(\alpha) = 1$. A second possibility to describe a Boolean function thus is by listing all of these, but again, we will have up to exponentially many tuples.

Much more compact methods to describe Boolean functions are Boolean circuits and propositional formulas, which we will define formally in the next subsection.

Throughout the text, we will refer to some basic Boolean functions (that are often used as gates when building circuits or as connectives when building formulas), with the notations listed below:

- 0-ary Boolean functions: $c_0 =_{\mathrm{def}} 0$ and $c_1 =_{\mathrm{def}} 1$.
  (We write 0 and 1 in formulas.)

- 1-ary Boolean functions: $id(x) =_{\mathrm{def}} x$; and $not(x) = 1$ iff $x = 0$.
  (In formulas we simply write x for $id(x)$ and $\overline{x}$ or $\neg x$ for $not(x)$.)

- some prominent 2-ary Boolean functions: $and(x, y) = 1$ iff $x = y = 1$, $or(x, y) = 0$ iff $x = y = 0$, $xor(x, y) = 1$ iff $x \neq y$, $eq(x, y) = 1$ iff $x = y$, $imp(x, y) = 0$ iff $x = 1$ and $y = 0$, and $nand(x, y) = 0$ iff $x = y = 1$.
  (In formulas, we use $\wedge, \vee, \oplus, \leftrightarrow, \rightarrow, |$, resp., all operators are used in infix notation.)

1.2 Assembling Boolean Functions – B-Circuits and Superposition

Let B be a set of Boolean functions. An *n-input B-circuit* (or, an n-input circuit over *basis* B) is a directed, acyclic graph where each node is labeled either with a variable x_i, $i = 1, 2, 3, \ldots, n$, or a function from B. We will call the nodes of this graph *gates*. The number of edges (also called *wires*) pointing into a gate is called *fan-in*, the number of wires leaving a gate is called *fan-out* of that gate. For reasons that will become clear shortly, we also order the edges pointing into a gate. If a wire leaving gate u is pointing into gate v, we say that u is a *predecessor gate* of v. The gates labeled with a variable must have a fan-in of 0; we call these *input gates*. The fan-in of gates labeled with a Boolean function has to be as large as the arity of that function. We mark one particular gate and call it *output gate*. Note that an input-gate can be the output-gate, too. Hence the function id can be computed by B-circuits for any B.

The computation of an n-input B-circuit C proceeds as we describe next. Given an n-bit input string $x = a_1 a_2 \ldots a_n$, every gate in C computes a Boolean value as follows: A gate v labeled with a variable x_i returns the bit a_i. A gate v of fan-in k labeled by a Boolean function f computes the value $f(b_1, \ldots, b_k)$, where $b_1, \ldots, b_k$ are the values computed by the predecessor gates of v, ordered according to the order of those wires connecting v with its predecessors. The value of C on input x is the value computed by the output gate. In this way, C computes an n-ary Boolean function, which we denote by f_C. (We will use the notations $f_C(a_1 \ldots a_n)$ and $f_C(a_1, \ldots, a_n)$ interchangeably with identical meaning.) The class $\mathrm{CIRC}(B)$ is the set of all functions that can be computed by B-circuits.

Note that a propositional formula can be seen as a circuit where the fan-out of each node is at most 1. So we treat B-formulas as a special case of (tree-like) B-circuits.

Given a fixed set B, what are the Boolean functions that can be computed by a B-circuit? More precisely, we are looking for a set of operations such that $\mathrm{CIRC}(B)$ can be obtained as the closure of B under these. The operations have to describe in a mathematical way what "soldering some given circuits together" means.

First note that surely every f from B is in $\mathrm{CIRC}(B)$: Just build a circuit that has as many input nodes as f has arguments, and draw an edge from every input node to one additional node, labeled with f. Suppose we have a B-circuit C_1 computing the n-ary Boolean function f_{C_1}, and another B-circuit C_2 computing m-ary f_{C_2}. Now we can derive new B-circuits by performing one of the following operations:

(i) We get a new circuit C_1' by just adding one input node to C_1. Since we do not add any new edges, the new node has no influence on the computed Boolean function besides the higher arity. We call this operation *introduction of a fictive variable* and get for all $a_1, \dots, a_{n+1} \in \{0,1\}$: $f_{C'_1}(a_1, \dots, a_{n+1}) = f_{C_1}(a_1, \dots, a_n)$. In general, we will say a variable of a Boolean function (an input gate of a circuit) is *fictive*, if the value of the function (the circuit) never depends on this variable (this input gate).

(ii) We may obtain circuit C'_1 from C_1 by arbitrarily permuting the input variables. This operation will be called *permutation of variables* and if $\pi \in \mathrm{S}_n$ is our permutation, we get for all $a_1, \dots, a_n \in \{0,1\}$: $f_{C'_1}(a_1, \dots, a_n) = f_{C_1}(a_{\pi^{-1}(1)}, \dots, a_{\pi^{-1}(n)})$.

(iii) Since the fan-out of gates is not restricted whatsoever, we can remove all outgoing edges of one input gate x_i of C_1 and assign them to another input gate x_j. After this operation, x_i is a fictive gate and we drop it. The thus derived circuit C'_1 is a B-circuit and computes a function of arity $n-1$, given by $f_{C'_1}(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n) = f_{C_1}(a_1, \dots, a_{i-1}, a_j, a_{i+1}, \dots, a_n)$. We call this operation *identification of variables.*

(iv) If we replace input gate x_n of C_1 by the whole circuit C_2, i.e., we replace x_n with the output gate of C_2 and we replace in C_2 every input gate x_j (for $1 \leq j \leq m$) by x_{n-1+j}, we get a new B-circuit C' of arity $n+m-1$. For the Boolean function $f_{C'}$ obtained in this way, we have: $f_{C'}(a_1, \dots, a_{n-1}, a_n, \dots, a_{n+m-1}) = f_{C_1}(a_1, \dots, a_{n-1}, f_{C_2}(a_n, \dots, a_{n+m-1}))$. This operation is called *substitution.*

The important observation now is that every B-circuit can be obtained by a sequence of these operations. Or, speaking about Boolean functions instead of circuits: The operations (i)–(iv) though defined on circuits correspond immediately to operations on Boolean functions. Define the closure of a set B of Boolean functions under *superposition*, denoted by $[B]$, to be the set of those functions that can be obtained from functions in $B \cup \{id\}$ by a finite sequence of applications of (i)–(iv). It is interesting to note that superposition is equivalent to arbitrary composition of Boolean functions and introduction of fictive variables. So $f \in [B]$ if and only if $f \in B \cup \{id\}$ or there is a $g \in [B]$ and $X_1, \dots, X_n$, that are either variables or functions from $[B]$, such that $f = g(X_1, \dots, X_n)$.

Superposition on the level of Boolean functions corresponds to soldering on the level of circuits, thus $[B] = \mathrm{CIRC}(B)$. We conclude that, if we want to know which Boolean functions can be computed by B-circuits, we do not have to talk about circuits at all: It suffices to study the closures of classes of Boolean functions under superposition.

Example 1.1. Let $f(x,y) =_{\text{def}} x \wedge \overline{y}$ be a Boolean function. Is the function $and(x,y)$ in $[f]$? (We use $[f]$ as a shorthand for $[\{f\}]$.) To answer this question, we have to find a composition of f's that is equal to $and(x,y)$. In this case it is easy to verify that $and(x,y) = f(x, f(x,y))$.

1.3 Post's Lattice

We say a set of Boolean functions B is a *clone* (or, B is *closed*) if $B = [B]$, i.e., no new functions can be derived from compositions of functions from $B \cup \{id\}$. Every set $B_0 \subseteq B$ with $[B_0] = B$ is called a *base* (or *basis*) of B. We also say that such a set B_0 is *complete in* B. As mentioned before, all closed classes of Boolean functions were identified by Post [Pos41], who also found a finite basis for each of them (see Fig. 1). Post also detected the very useful inclusion structure of the classes (see Fig. 2), hence the name *Post's graph* or (as we will shortly prove that the structure is a lattice) *Post's lattice.*

We would like to mention that the names of the classes in Fig. 2 are not those Post used originally. Post was not only interested in closed classes of Boolean functions, but he additionally considered so called "iterative classes" which are missing some of the closure properties of our notion of superposition. This leads to the idiosyncrasy that if we used Post's names, we would have, e. g., classes $\mathrm{P}_1, \mathrm{P}_3, \mathrm{P}_5, \mathrm{P}_6$, but no P_2 and P_4. The terminology used in Fig. 2 was developed by Klaus Wagner in an attempt to construct a consistent scheme of names for closed classes, and further propagated in [RW00, RV00].

We want to introduce some clones:

- BF is the class of all Boolean functions.
- For $a \in \{0,1\}$, a Boolean function f is called *a-reproducing* if $f(a,\ldots,a) = a$. The clones R_a contain all a-reproducing Boolean functions.
- For $(a_1,\ldots,a_n),(b_1,\ldots,b_n) \in \{0,1\}^n$, we say $(a_1,\ldots,a_n) \leq (b_1,\ldots,b_n)$ if $a_i \leq b_i$ for $1 \leq i \leq n$. An n-ary Boolean function f is called *monotonic* if for all $\alpha,\beta \in \{0,1\}^n$ holds: If $\alpha \leq \beta$ then $f(\alpha) \leq f(\beta)$. Typical monotonic Boolean functions are *and* and *or*. The class of all monotonic Boolean functions is denoted by M.
- A Boolean function f is called *self-dual* if for all $a_1,\ldots,a_n \in \{0,1\}$ we have $f(a_1,\ldots,a_n) = \neg f(\overline{a_1},\ldots,\overline{a_n})$. Therefore constants are never self-dual and simple self-dual functions are *id* and *not*. The class of all self-dual Boolean functions is called D.
- Boolean functions that can be defined with a linear formula are called *linear*, i.e., an n-ary function f is linear if there exist constants $c_0,\ldots,c_n \in \{0,1\}$ such that $f(x_1,\ldots,x_n) = c_0 \oplus c_1x_1 \oplus \cdots \oplus c_nx_n$. (Here and in the following, we do not distinguish between a Boolean function and the defining propositional formula. Also, for variables x,y we use xy as a shorthand for $x \wedge y$.) The typical linear functions are *xor* and *eq*. The class of all linear Boolean functions is called L.
- Let $T \subseteq \{0,1\}^n$ and $a \in \{0,1\}$. We call T *a-separating* if there exists an $i \in \{1,\ldots,n\}$ such that for all $(b_1,\ldots,b_n) \in T$ holds $b_i = a$. A Boolean function f is called *a-separating* if $f^{-1}(a)$ is a-separating. The function f is called *a-separating of level k* if every $T \subseteq f^{-1}(a)$ with $|T| = k$ is a-separating. The classes of all a-separating functions are called S_a and the classes of a-separating functions of level k are called S_a^k.
- The class E is the class of all Boolean functions that can be described by formulas build over $\wedge$, 0 and 1: $\mathrm{E} = \{\, f \mid n \in \mathbb{N} \text{ and } f(x_1,\ldots,x_n) = c_0 \wedge (c_1 \vee x_1) \wedge \ldots \wedge (c_n \vee x_n) \text{ for some constants } c_i,\ 0 \leq i \leq n \,\}$. Analogously, V is the class of Boolean functions that can be described by formulas build over $\vee$, 0 and 1. ("E" and "V" stand for "et" and "vel", the Latin names of AND and OR.)
- The class I_2 contains all projections (i.e., all Boolean functions I_k^n with $I_k^n(a_1,\ldots,a_n) = a_k$ for all $a_1,\ldots,a_n \in \{0,1\}$), and I contains all projections and additionally all constants (i.e., all Boolean functions c_a^n, $a \in \{0,1\}$, with $c_a^n(a_1,\ldots,a_n) = a$ for all $a_1,\ldots,a_n \in \{0,1\}$). N_2 contains all projections and all negations of projections. The class N contains N_2 and all constants.

All other classes in Post's graph can be obtained from the above by intersection, as we illustrate next. Let A and B be clones and

Class	Definition	Base(s)
BF	all Boolean functions	$\{and, not\}$
R_0	$\{f \in \mathrm{BF} \mid f$ is 0-reproducing$\}$	$\{and, xor\}$
R_1	$\{f \in \mathrm{BF} \mid f$ is 1-reproducing$\}$	$\{or, x \oplus y \oplus 1\}$
R_2	$\mathrm{R}_1 \cap \mathrm{R}_0$	$\{or, x \wedge (y \oplus z \oplus 1)\}$
M	$\{f \in \mathrm{BF} \mid f$ is monotonic$\}$	$\{and, or, c_0, c_1\}$
M_1	$\mathrm{M} \cap \mathrm{R}_1$	$\{and, or, c_1\}$
M_0	$\mathrm{M} \cap \mathrm{R}_0$	$\{and, or, c_0\}$
M_2	$\mathrm{M} \cap \mathrm{R}_2$	$\{and, or\}$
S_0^n	$\{f \in \mathrm{BF} \mid f$ is 0-separating of degree $n\}$	$\{imp, \mathrm{dual}(h_n)\}$
S_0	$\{f \in \mathrm{BF} \mid f$ is 0-separating$\}$	$\{imp\}$
S_1^n	$\{f \in \mathrm{BF} \mid f$ is 1-separating of degree $n\}$	$\{x \wedge \overline{y},\ h_n\}$
S_1	$\{f \in \mathrm{BF} \mid f$ is 1-separating$\}$	$\{x \wedge \overline{y}\}$
S_{02}^n	$\mathrm{S}_0^n \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z}), \mathrm{dual}(h_n)\}$
S_{02}	$\mathrm{S}_0 \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z})\}$
S_{01}^n	$\mathrm{S}_0^n \cap \mathrm{M}$	$\{\mathrm{dual}(h_n), c_1\}$
S_{01}	$\mathrm{S}_0 \cap \mathrm{M}$	$\{x \vee (y \wedge z), c_1\}$
S_{00}^n	$\mathrm{S}_0^n \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z), \mathrm{dual}(h_n)\}$
S_{00}	$\mathrm{S}_0 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z)\}$
S_{12}^n	$\mathrm{S}_1^n \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z}), h_n\}$
S_{12}	$\mathrm{S}_1 \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z})\}$
S_{11}^n	$\mathrm{S}_1^n \cap \mathrm{M}$	$\{h_n, c_0\}$
S_{11}	$\mathrm{S}_1 \cap \mathrm{M}$	$\{x \wedge (y \vee z), c_0\}$
S_{10}^n	$\mathrm{S}_1^n \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z), h_n\}$
S_{10}	$\mathrm{S}_1 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z)\}$
D	$\{f \mid f$ is self-dual$\}$	$\{x\overline{y} \vee x\overline{z} \vee \overline{yz}\}$
D_1	$\mathrm{D} \cap \mathrm{R}_2$	$\{xy \vee x\overline{z} \vee y\overline{z}\}$
D_2	$\mathrm{D} \cap \mathrm{M}$	$\{xy \vee yz \vee xz\}$
L	$\{f \mid f$ is linear$\}$	$\{xor, c_1\}$
L_0	$\mathrm{L} \cap \mathrm{R}_0$	$\{xor\}$
L_1	$\mathrm{L} \cap \mathrm{R}_1$	$\{eq\}$
L_2	$\mathrm{L} \cap \mathrm{R}_2$	$\{x \oplus y \oplus z\}$
L_3	$\mathrm{L} \cap \mathrm{D}$	$\{x \oplus y \oplus z \oplus c_1\}$
V	$\{f \mid f$ is an n-ary or-function or a constant function$\}$	$\{or, c_0, c_1\}$
V_0	$[\{or\}] \cup [\{c_0\}]$	$\{or, c_0\}$
V_1	$[\{or\}] \cup [\{c_1\}]$	$\{or, c_1\}$
V_2	$[\{or\}]$	$\{or\}$
E	$\{f \mid f$ is an n-ary and-function or a constant function$\}$	$\{and, c_0, c_1\}$
E_0	$[\{and\}] \cup [\{c_0\}]$	$\{and, c_0\}$
E_1	$[\{and\}] \cup [\{c_1\}]$	$\{and, c_1\}$
E_2	$[\{and\}]$	$\{and\}$
N	$[\{not\}] \cup [\{c_0\}] \cup [\{c_1\}]$	$\{not, c_1\}$, $\{not, c_0\}$
N_2	$[\{not\}]$	$\{not\}$
I	$[\{id\}] \cup [\{c_1\}] \cup [\{c_0\}]$	$\{id, c_0, c_1\}$
I_0	$[\{id\}] \cup [\{c_0\}]$	$\{id, c_0\}$
I_1	$[\{id\}] \cup [\{c_1\}]$	$\{id, c_1\}$
I_2	$[\{id\}]$	$\{id\}$

Figure 1: List of all Boolean clones with bases ($h_n = \bigvee_{i=1}^{n+1} x_1 \cdots x_{i-1} x_{i+1} \cdots x_{n+1}$ and $\mathrm{dual}(f)(a_1, \ldots, a_n) = \neg f(\overline{a_1}, \ldots, \overline{a_n})$).

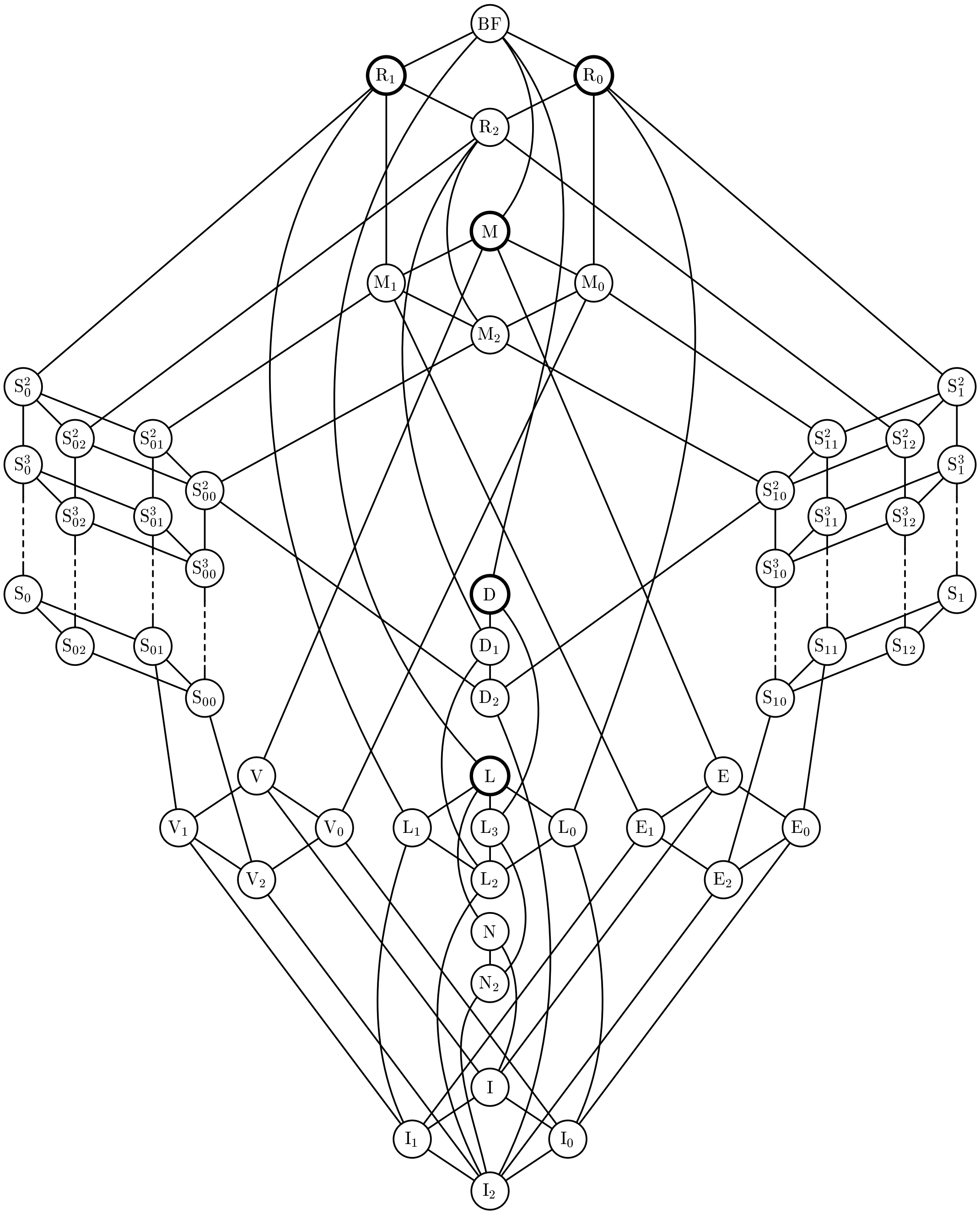

Figure 2: Graph of all Boolean clones

- let $A \sqcap B$ be the largest clone that is contained in both A and B, and
- let $A \sqcup B$ be the smallest clone that contains both A and B.

Note that $A \cap B \subseteq [A \cap B]$. On the other hand we have $A \cap B \subseteq A$ and hence $[A \cap B] \subseteq [A] = A$. Analogously we obtain $[A \cap B] \subseteq [B] = B$, which gives us $[A \cap B] \subseteq A \cap B$. We conclude that $[A \cap B] = A \cap B$, hence $A \cap B$ is again a clone. But since $A \sqcap B$ is the largest clone that is contained in both A and B we have that $A \cap B \subseteq A \sqcap B$. Since by definition, $A \sqcap B \subseteq A \cap B$, we conclude $A \cap B = A \sqcap B$. Similarly we can show that $[A \cup B] = A \sqcup B$ holds. It is easy to check, that $\sqcap$ and $\sqcup$ are both associative and commutative. Besides that, $A \sqcap (A \sqcup B) = A$ and $A \sqcup (A \sqcap B) = A$ hold, so the Boolean clones form a lattice.

Example 1.2. The class E_1 consists of functions from E that are also 1-reproducing, $\mathrm{E}_1 = \mathrm{E} \cap \mathrm{R}_1$. Similarly, $\mathrm{E}_0 = \mathrm{E} \cap \mathrm{R}_0$. The class E_2 finally consists of all functions from E that are both 1-reproducing and 0-reproducing. As a general mnemonic, functions from classes with index 0 are 0-reproducing, functions from classes with index 1 are 1-reproducing, and functions from classes with index 2 are both. This implies that, while E contains all constant functions, the only constant functions in E_0 are constant 0 functions, the only constant functions in E_2 are constant 1 functions, and that E_2 does not contain any constant functions. This is similarly valid for other classes as R_i, M_i, L_i, V_i, E_i, N_i and I_i, where i is empty or an index 0, 1 or 2. Observe that D is excluded here since constant functions are never self-dual.

The well known *Post's classes* are those classes B in Post's lattice with the properties that $B \subsetneq \mathrm{BF}$ and that for every B' with $B \subsetneq B' \subseteq \mathrm{BF}$ we obtain $[B'] = \mathrm{BF}$, hence they are the maximal classes which do not contain all Boolean functions; in other words: Post's classes are the dual atoms of Post's lattice. (Sometimes they are simply called *maximal clones* [Sze86].) There are five such classes, namely R_0, R_1, M, D, and L; these are circled bold in Fig. 2. This suggests a very convenient test whether a given function f (or set B) is complete in the sense that $[f] = \mathrm{BF}$ ($[B] = \mathrm{BF}$): One just has to make sure that f (or B) is not contained in one of Post's classes.

Example 1.3. Consider the Boolean function *nand*. Clearly $\mathit{nand} \notin \mathrm{R}_0$ and $\mathit{nand} \notin \mathrm{R}_1$. Moreover we have $\mathit{nand} \notin \mathrm{M}$, because $(0,0) \leq (1,1)$, but $\mathit{nand}(0,0) \not\leq \mathit{nand}(1,1)$. The *nand*-function is not self-dual since $\mathit{nand}(0,1) \neq \neg\mathit{nand}(1,0)$. Finally note that *nand* is not linear. To show this suppose that *nand* is linear. In this case we have $\mathit{nand}(x,y) = c_0 \oplus c_1x \oplus c_2y$. Because $\mathit{nand}(0,0) = 1$ we have $c_0 = 1$, by $\mathit{nand}(1,0) = 1$ we obtain $c_1 = 0$ and by $\mathit{nand}(0,1) = 1$ we obtain $c_2 = 0$ too. But this is a contradiction since *nand* is not a constant function. Since *nand* is not contained in one of the five maximal clones we conclude that $[\mathit{nand}] = \mathrm{BF}$. Hence each Boolean function can be implemented by a *nand*-circuit.

Another interesting and useful aspect is *duality*. We say a function f is the dual function of g if they both have the same arity n and for all $a_1, \dots, a_n \in \{0,1\}$ holds $f(a_1, \dots, a_n) = \overline{g(\overline{a_1}, \dots, \overline{a_n})}$. We define $\mathrm{dual}(f)$ to be the dual function of f and for a set of Boolean functions B we let $\mathrm{dual}(B) = \{\mathrm{dual}(f) \mid f \in B\}$.

Duality Principle. Let $\alpha \in \{0,1\}^n$, let g be an m-ary and $f_1, \dots, f_m$ be n-ary Boolean functions, and let $h(\alpha) =_{\mathrm{def}} g(f_1(\alpha), \dots, f_m(\alpha))$. Then $\mathrm{dual}(h)(\alpha) = \mathrm{dual}(g)(\mathrm{dual}(f_1)(\alpha), \dots, \mathrm{dual}(f_m)(\alpha))$.

This implies that, if a Boolean function f is computed by some B-circuit C, then $\mathrm{dual}(f)$ is computed by the circuit obtained from C by replacing each gate with the dual one.

For each clone B the set $\mathrm{dual}(B)$ is again a clone. When looking at Fig. 2 we can find the dual classes very easily. Imagine a symmetry axis through BF and I_2. Now, for each class on the one side of the axis the dual one is the mirror image on the other side of the axis. For classes B located on the axis, we have $\mathrm{dual}(B) = B$. Special cases are the so called *self-dual classes*; these are the subclones of D, the clone of all self-dual functions, where a function f is called self-dual if $\mathrm{dual}(f) = f$. A lot of properties hold for a Boolean function iff they hold for its dual. Thus, the symmetry of Post's lattice can be used to simplify proofs.

We want to give some examples that illustrate how comfortable life becomes with Post's lattice:

Example 1.4. Let $f = x \wedge \overline{y}$ be the function from Example 1.1 and let us again wonder whether or not the function *and* is in $[\{f\}]$. When we look at Fig. 1 we see that $\{f\}$ is the base of a class named S_1. Fig. 2 shows us that the class E_2, that is a subset of S_1, has the base $\{and\}$. So obviously $and \in \mathrm{S}_1 = [\{f\}]$.

Example 1.5. For decision problems dealing with Boolean circuits (e.g., the circuit value problem or satisfiability of circuits; we discuss these problems in detail in Sect. 2), often two variations exist: The first one is that we have constants "for free" in the circuit and the second one is to not allow them. Suppose we want to examine such a decision problem where $\{f\}$-circuits are the input instances and f is again $x \wedge \overline{y}$.

For a given $\{f\}$-circuit C, if we exchange some of the input-gates of C with constants 0, we get an $\{f, c_0\}$-circuit. But since $\{c_0\}$ is contained in the clone $[and, c_0] = \mathrm{E}_0 \subseteq \mathrm{S}_1 = [\{f\}]$, we can replace each of the c_0's with an $\{f\}$-subcircuit computing c_0 (in this example, $c_0 = f(x, x)$), so the whole circuit becomes an $\{f\}$-circuit again. Thus, problems are as difficult for $\{f, c_0\}$-circuits as they are for $\{f\}$-circuits.

Now, if we exchange some of the input gates of C with constants 1, we get an $\{f, c_1\}$-circuit. The function c_1 is not contained in any subclass of S_1, so by using the constant 1 we will be able to compute more than just functions from $[\{f\}]$. Which are these? Since $[\{f, c_1\}] = [\mathrm{S}_1 \cup \{c_1\}]$ this is the smallest clone containing both S_1 and I_1 (the class I_1 is the smallest class that contains c_1). A look at Fig. 2 shows us that this is BF, the set of all Boolean functions! So problems for $\{f, c_1\}$-circuits are as difficult as they are for $\{\wedge, \vee, \neg\}$-circuits.

We see that those classes in Post's lattice that contain the constant functions are of particular interest. These are exactly the classes $[B \cup \{c_0, c_1\}]$, where B is an arbitrary set of Boolean functions. Making use of the lattice properties, we thus only have to determine all classes $B \sqcup \mathrm{I}$ (I is the smallest class containing both c_0 and c_1.) for all B in Post's lattice. In this way, one immediately obtains Fig. 3, presenting the inclusion structure among all Boolean clones with constants.

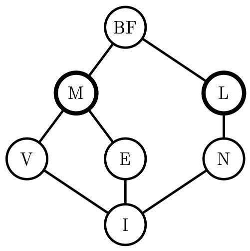

Figure 3: Graph of all Boolean clones with constants.

Example 1.6. Suppose f is an n-ary monotonic Boolean function, i.e., $f \in \mathrm{M}$. Is f also in L? For this, first note that $\mathrm{L} \not\subseteq \mathrm{M}$, and $\mathrm{M} \cap \mathrm{L} = \mathrm{M} \sqcap \mathrm{L} = \mathrm{I}$ (see Fig. 2). Since $\mathrm{I} = [\mathrm{I}_2 \cup \{c_0, c_1\}]$, f is in L if and only if it is a projection or a constant function. Since f is monotonic, f is constant if and only if $f(0, \ldots, 0) = 1$ or $f(1, \ldots, 1) = 0$.

We next claim that f is a projection if and only if there is exactly one $m \in \{1, \ldots, n\}$ such that $f(\alpha) = 1$ and $f(\beta) = 0$, where $\alpha = (0^{m-1}, 1, 0^{n-m})$ and $\beta = (1^{m-1}, 0, 1^{m-n})$: If f is a projection, then such an m (and only one) certainly exists. On the other hand, for every $\gamma \in \{0,1\}^{m-1} \times \{1\} \times \{0,1\}^{n-m}$ holds $\gamma \geq \alpha$. Therefore, and since f is monotonic, $f(\gamma) = 1$. Furthermore, for every $\delta \in \{0,1\}^{m-1} \times \{0\} \times \{0,1\}^{n-m}$ holds $\beta \geq \delta$ and therefore $f(\delta) = 0$. That means $f(a_1, \ldots, a_n) = a_m$ for all $a_1, \ldots, a_n \in \{0,1\}$ which means that f is a projection.

2 The Complexity of some Problems in Post's Lattice

In computational complexity theory, a large number of problems related to propositional formulas or Boolean circuits have been studied intensively. The most prominent of them is probably the first problem ever shown to be NP-complete [Coo71, Lev73], the problem SAT (see also [GJ79, Problem LO1]), asking if a given propositional formula F is satisfiable. A natural question now is of course, how the complexity of the problem changes if not all propositional formulas are allowed but only those with connectives from a previously fixed set B of Boolean functions. Thus, we are lead to the following problem, first studied systematically by Lewis in 1979 [Lew79]:

Problem:	SAT(B)
Input:	A B-formula F
Question:	Is F satisfiable?

As argued in the previous section, this problem leads immediately to the context of Post's classes. In order to obtain a solution, it is convenient to consider first a more general problem than the above, asking, given a *Boolean circuit* C with gates taken from a set B, if there is an input x such that C on x outputs the value 1. We will denote this problem by Circuit-SAT(B). We first present a complete classification of the complexity of Circuit-SAT(B), showing for which bases B the problem is NP-complete and for which bases it is efficiently solvable. The results says essentially that the problem is NP-complete iff $[B]$ contains the Boolean function $x \wedge \neg y$.

Theorem 2.1 [RW00]. *If* $[B] \supseteq \mathrm{S}_1$ *then* Circuit-SAT(B) *is* NP*-complete; in all other cases* Circuit-SAT(B) *is polynomial-time solvable.*

Proof. First note that Circuit-SAT(B) $\in$ NP for every B, since Circuit-SAT without restrictions on the gate types is in NP.

Our strategy in this proof is to first try to establish a number of clones B, to which NP-completeness for the unrestricted case carries over. Then, making use of Fig. 2, we will examine the hopefully not too many remaining cases.

Looking back at Example 1.5, we know that $[\mathrm{S}_1 \cup \{1\}]$ is the class of all Boolean functions, hence we know that Circuit-SAT($\mathrm{S}_1 \cup \{1\}$) is NP-complete. The question now is how to get rid of the constant 1. Since S_1 is a superclass of E_2 it contains the *and*-function. Given now an S_1-circuit C, we know that there is another S_1-circuit $\widehat{C}$, equivalent to $C \wedge x$, where x is a new input variable. Moreover, there is an input that makes C output 1 if and only if there is an input that makes $\widehat{C}$ output 1, and additionally, in every such input x is set to 1. Hence, we know that we can use variable x as a replacement for the constant 1. We conclude that for every $(\mathrm{S}_1 \cup \{1\})$-circuit C there is an S_1-circuit $\widehat{C}$ such that there is an input that makes C output 1 if and only if there is an input that makes $\widehat{C}$ output 1. Since we have seen that Circuit-SAT($\mathrm{S}_1 \cup \{1\}$) is NP-complete we now conclude that Circuit-SAT(S_1) is NP-complete. Obviously these arguments carry over to all closed supersets of S_1.

Looking at Fig. 2 we see that it only remains to address the classes R_1, M, D, and L. Every R_1-circuit outputs 1 for the input vector consisting only of 1's. Every M-circuit C has the property that the Boolean function it computes is coordinate-wise monotonic; hence there is an input that makes C output 1 iff the all 1's input makes C output 1. Every D-circuit C has the property that either the all 1's input makes C output 1, or, since the Boolean function C computes is self-dual, the all 0's input makes C output 1. Hence, there is an input that makes C output 1. Finally, let C be an L-circuit. We may assume that C consists only of $\oplus$ and 1-gates, since $\{\oplus, 1\}$ is a basis for L. There is an input such that C outputs 1 if and only if the number of paths from the output gate to a constant 1 gate is odd or there is some input variable such that the number of paths from the output gate to gates labeled with this variable is odd. This can be checked in polynomial time (in fact, in $\oplus$L). $\square$

Corollary 2.2 [Lew79]. *If* $[B] \supseteq S_1$ *then* SAT(B) *is* NP*-complete; in all other cases* SAT(B) *is polynomial-time solvable.*

Proof. Immediately from the above, we conclude that the easy cases carry over to the formula case, i.e., if $[B] \not\supseteq S_1$ then SAT(B) is in P. If $[B] \supseteq S_1$ we would like to proceed as above, transforming an arbitrary propositional formula F into an S_1-formula, but we encounter a problem: The $(S_1 \cup \{1\})$-formulas for the connectives in F that we need in the transformation may use some input variable more than once, leading to an explosion in formula size when going from F to an equivalent S_1-formula. However, we may assume that F is in conjunctive normal-form with at most 3 literals per clause (3-CNF), since the satisfiability problem for these formulas, denoted by 3SAT, is known to be still NP-complete [GJ79, Problem LO2]. Now, we insert parentheses in such a way that we get a tree of $\wedge$'s of depth logarithmic in the size of F. Now replacing every $\wedge$ by an equivalent $(S_1 \cup \{1\})$-formula increases the formula size by only a polynomial in the original size. Thus, 3SAT reduces to SAT(S_1), showing that SAT(S_1) is NP-complete. $\square$

Coming back to Boolean circuits, a more often looked at problem is not to ask if *there is* any input that makes the circuit output 1, but the problem, *given* an input, to determine the circuit's output. This is the so called *circuit value problem*:

Problem:	CVP(B)
Input:	A B-circuit C and an input vector x
Question:	Does C on input x output 1?

In the case that no restrictions on B are given, this problem is known to be P-complete under logspace-reductions [Lad75] and even under NC^1-reductions, cf. [GHR95, Chap. 6] or [Vol99, Chap. 4.6]. This means that most researchers in the field expect that it allows no efficient parallel solution in the sense of an NC-algorithm, i.e., no polylog-time algorithm using a reasonable amount of hardware (polynomial number of processors); formally the result says that if P $\neq$ NC then CVP is not in NC.

The next theorem presents a complete classification of those sets of allowed gates, for which an efficient parallel algorithm for the circuit value problem exists.

Theorem 2.3 [RW00]. *If* $B \subseteq$ V *or* $B \subseteq$ L *or* $B \subseteq$ E *then* CVP$(B) \in$ NC*; in all other cases* CVP(B) *is* P*-complete.*

Proof. We will first give efficient (NC-) algorithms in the cases $B \subseteq \mathrm{V} \cup \mathrm{L} \cup \mathrm{E}$. Making use of Post's lattice, we then examine all remaining cases and prove that the circuit value problem is P-complete there.

First, we observe that if the base B contains the constant functions 0 or 1, we may simply treat gates for these functions in the same way as inputs to the circuit, set to 0 or 1, resp. Thus, in general, $\mathrm{CVP}(B \cup \{0,1\})$ is of the same complexity as $\mathrm{CVP}(B)$. (Formally, these problems reduce to one another under logspace- or NC^1-reductions.)

If $B \subseteq \mathrm{V} \cup \mathrm{L} \cup \mathrm{E}$, we thus only have to look at $\{\vee\}$-circuits, $\{\oplus\}$-circuits, and $\{\wedge\}$-circuits. The output of a $\{\vee\}$-circuit is 1 iff there is one path leading back from the output to a 1-input. This is a graph accessibility problem, hence $\mathrm{CVP}(\mathrm{V}) \in \mathrm{NL}$. If C is a $\{\wedge\}$-circuit, the output is 0 iff there is a path leading back from the output to a 0-input. This leads to the upper bound $\mathrm{coNL} = \mathrm{NL}$. If C is a $\oplus$-circuit, the output is 1 iff the number of paths from the output to input gates with value 1 is odd. This is in $\oplus\mathrm{L}$. Since $\mathrm{NL} \cup \oplus\mathrm{L} \subseteq \mathrm{NC}^2$, the first part of the theorem is proven.

If now $B \not\subseteq \mathrm{V} \cup \mathrm{L} \cup \mathrm{E}$, a look at Fig. 2 reveals $\mathrm{S}_{00} \subseteq [B]$, $\mathrm{S}_{10} \subseteq [B]$, or $\mathrm{D}_2 \subseteq [B]$. Considering the bases of S_{00}, S_{10}, and D_2 (see Fig. 1) or making use of Fig. 2, we see that $\{\wedge, \vee\} \subseteq [B \cup \{0,1\}]$. Thus, $\mathrm{CVP}(B \cup \{0,1\})$ (and by the above, $\mathrm{CVP}(B)$) are at least as hard as the circuit value problem for $\{\vee, \wedge\}$-circuits. This is the so called *monotone circuit value problem*, known to be P-complete [Gol77], cf. also [GHR95, Chap. 6] or [Vol99, Chap. 4.6]. We conclude that $\mathrm{CVP}(B)$ is P-complete in these cases as well. $\square$

Further Complexity Results

A number of further computational problems were looked at in the Post context. These concern, among others, the question to determine the number of satisfying assignments of a given propositional formula and related threshold questions [RW00], the evaluation of quantified Boolean formulas [RW00], the isomorphism problem for Boolean formulas [Rei03], the question to determine the lexicographically maximal satisfying assignment of a given propositional formula and the question whether in this assignment (or the lexicographically minimal satisfying assignment) a particular variable is set to true [RV00], the question to determine the minimal satisfying assignment of a given propositional formula in coordinate-wise partial order [KK01].

An interesting question from a complexity point of view is also to determine, given some Boolean function f, in which class of Post's lattice it falls optimally. We turn to this question in the final section of this survey.

3 The Meta-Problem: The Complexity of the Clones

The function computed by a B-circuit C is per definition in $[B]$, but that does not mean that there is no other set B' of boolean functions with $[B'] \subsetneq [B]$ such that $f_C \in [B']$. For example the circuit C given by $\neg(\overline{x_1} \vee \overline{x_2})$ uses gates $\vee$ and $\neg$, and $[\{\vee, \neg\}] = \mathrm{BF}$, but f_C is even in E_2 since it is equal to $x_1 \wedge x_2$. Simply looking at the gates of a circuit C obviously is not sufficient to find the smallest class containing f_C. Is there a tractable way to solve this question? We show that the answer is "no" in most cases.

First we introduce the membership problem formally:

Problem:	MEM(B)
Input:	A $\{\wedge, \vee, \neg\}$–circuit C
Question:	Is f_C in B?

Theorem 3.1 [Böh01]. *If* $B \in \{\mathrm{R}_2, \mathrm{R}_1, \mathrm{R}_0, \mathrm{BF}\}$ *then* $\mathrm{MEM(B)} \in \mathrm{P}$*; in all other cases* MEM(B) *is* coNP*-complete.*

Proof. If $B = \mathrm{BF}$ the claim is obvious; if $B \in \{\mathrm{R}_1, \mathrm{R}_0, \mathrm{R}_2\}$ we only have to compute $C(0, \dots, 0)$, or $C(1, \dots, 1)$, or both to decide whether or not $C \in \mathrm{MEM(B)}$.

On the other hand, looking at the clones in Fig. 1 we see that all defining conditions can be checked by a coNP calculation, hence $\mathrm{MEM(B)} \in \mathrm{coNP}$ for all B. Let TAUT be the set of all tautological $\{\vee, \wedge, \neg\}$-circuits, i. e., the set of circuits that return 1 on every input. This problem is coNP-complete. For a $\{\vee, \wedge, \neg\}$-circuit $C(x_1, \dots, x_n)$ let

$$C' =_{\mathrm{def}} \left((\overline{C(x_1, \dots, x_n)} \vee \overline{C(y_1, \dots, y_n)}) \wedge \overline{z} \right) \vee u.$$

We show that $C \in \mathrm{TAUT}$ if and only if $f_{C'}$ is linear, monotonic, self-dual, and 1-separating of level 2. First, let $C \in \mathrm{TAUT}$. That means, that $C(a_1, \dots, a_n) = 1$ for all $a_1, \dots, a_n \in \{0, 1\}$. Therefore $C' \equiv u$, i. e. C' is a projection to one variable and that means that $f_{C'} \in \mathrm{I}_2$. A look at Fig. 2 shows us that every function from I_2 is linear, monotonic, self-dual, and 1-separating of level 2.

If on the other hand $C \notin \mathrm{TAUT}$ then there is a tuple $\alpha = (a_1, \dots, a_n) \in \{0, 1\}^n$ such that $C(\alpha) = 0$. Let us define $\overline{\alpha} =_{\mathrm{def}} (\overline{a_1}, \dots, \overline{a_n})$. Now make the following observations:

- Assume C' is linear. Then C' can be described by a formula $c_0 \oplus c_1 x_1 \oplus \dots \oplus c_n x_n \oplus c_{n+1} y_1 \oplus \dots \oplus c_{2n} y_n \oplus c_{2n+1} z \oplus c_{2n+2} u$ where c_i is a constant for $1 \leq i \leq 2n+2$. Since $C'(\alpha, \alpha, 0, 0) = 1 = C'(\alpha, \alpha, 0, 1)$ obviously $c_{2n+2} = 0$. But on the other hand we have $0 = C'(\alpha, \alpha, 1, 0) \neq C'(\alpha, \alpha, 1, 1) = 1$ and so $c_{2n+2} = 1$ which is a contradiction.
- C' is not self-dual, since $C'(\alpha, \alpha, 0, 0) = C'(\overline{\alpha}, \overline{\alpha}, 1, 1)$.
- C' is not monotonic, since $1 = C'(\alpha, \alpha, 0, 0) > C'(\alpha, \alpha, 1, 0) = 0$.
- Finally, C' is not 1-separating of degree 2, since $C'(\alpha, 0^{n+2}) = 1 = C'(0^n, \alpha, 0, 0)$.

Now look at Post's lattice: Note that every clone $B \notin \{\mathrm{R}_2, \mathrm{R}_1, \mathrm{R}_0, \mathrm{BF}\}$ is a subset of M, D, L, S_1^2, or S_0^2. For the first four cases we gave a reduction from TAUT to MEM(B) above. If B is a subclone of S_0^2, note that dual(B) is a subclone of S_1^2. Since for all B, $\mathrm{MEM(B)} \leq_{\mathrm{m}}^{\mathrm{log}} \mathrm{MEM(dual(B))}$ via the Duality Principle (see Sect. 1.3), coNP-hardness for the remaining classes follows. □

References

[BCRV04] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks: Constraint satisfaction problems. Technical Report 318, Fachbereich Mathematik und Informatik, Universität Würzburg, 2004.

[Böh01] E. Böhler. On the relative complexity of Post's classes. Technical Report 286, Fachbereich Mathematik und Informatik, Universität Würzburg, 2001.

[Coo71] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the Association for Computing Machinery*, 18:4–18, 1971.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, New York, 1995.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Gol77] L. M. Goldschlager. The monotone and planar circuit value problems are log-space complete for P. *SIGACT News*, 9:25–29, 1977.

[JGK70] S. W. Jablonski, G. P. Gawrilow, and W. B. Kudrjawzew. *Boolesche Funktionen und Postsche Klassen*, volume 6 of *Logik und Grundlagen der Mathematik.* Friedr. Vieweg & Sohn and C. F. Winter'sche Verlagsbuchhandlung, Braunschweig and Basel, 1970.

[KK01] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability in Post's lattice. Unpublished notes, 2001.

[Lad75] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):12–20, 1975.

[Lev73] L. A. Levin. Universal sorting problems. *Problemi Peredachi Informatsii*, 9(3):115–116, 1973. English translation: *Problems of Information Transmission*, 9(3):265–266.

[Lew79] H. R. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.

[Pip97] N. Pippenger. *Theories of Computability.* Cambridge University Press, Cambridge, 1997.

[Pos41] E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

[Rei03] S. Reith. On the complexity of some equivalence problems for propositional calculi. In *Proceedings 28th International Symposium on Mathematical Foundations of Computer Science*, pages 632–641, 2003.

[RV00] S. Reith and H. Vollmer. Optimal satisfiability for propositional calculi and constraint satisfaction problems. In *Proceedings 25th International Symposium on Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 640–649. Springer Verlag, 2000.

[RW00] S. Reith and K. W. Wagner. The complexity of problems defined by Boolean circuits. Technical Report 255, Institut für Informatik, Universität Würzburg, 2000. To appear in *Proceedings International Conference Mathematical Foundation of Informatics*, Hanoi, Oct. 25–28, 1999.

[Sze86] Á. Szendrei. *Clones in Universal Algebra.* Séminaire de mathématiques supérieures, NATO Advanced Study Institute. Les Presses de l'Université de Montréal, Montréal, 1986.

[Ugo88] A. B. Ugolnikov. Closed Post classes. *Izvestiya VUZ. Matematika*, 32:79–88 (131–142), 1988.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

[Zve00] I. E. Zverovich. Characterizations of closed classes of Boolean functions in terms of forbidden subfunctions and Post classes. Technical Report 17-2000, Rutgers Center for Operations Research, Rutgers University, 2000.