# An Algebra of Switching Networks

Andrey Mokhov
Schools of CS and EEE, Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk

*Is it right, that regardless of the existence of the already elaborated algebra of logic, the specific algebra of switching networks should be considered as a utopia?*
Paul Ehrenfest, 1910

**Abstract**

A switch, mechanical or electrical, is a fundamental building element of any digital computation system. Having two possible states, namely On and Off, it naturally lends itself to Boolean reasoning. Furthermore, a network of interconnected switches can compute any Boolean function, which was formally demonstrated by Shannon in his thesis back in 1937. A sound theory of switching networks was soon founded on the basis of two mathematical constructs: Boolean algebra and graph theory — the former to describe a network functionality, and the latter to describe its structure. A functionality therefore could be deduced from a given structure via *analysis* and a structure could be created for a specified functionality via *synthesis*.

In this work a unified *algebra of switching networks* is introduced. As the name suggests, the elements of the algebra are switching networks, rather than just Boolean functions (as in Boolean algebra) or vertices and edges (as in graph theory). This allows one to express both functionality and structure of switching networks by the same mathematical language and brings in new methods of system composition, which are of paramount importance for modern system design and development heavily relying on the reuse of components and interfaces. Using the rules of the algebra one can optimise the structure of a switching network for different non-functional criteria, such as size, energy efficiency, and reliability, to name but a few, by preserving the specified functional requirements.

# 1 Introduction

The dawn of computer engineering was marked by manual design at the level of basic switching elements, such as electromechanical relays. The elements were large and expensive by today's standards, thus each one had to be accounted for. Prior to the seminal work by Shannon [15], design of relay networks was a trial and error process on graphs and required a great deal of ingenuity. Shannon demonstrated that Boolean algebra of logic [5][14] can be used to reason about functionality of relay networks and described the first synthesis methods [15][16] that liberated designers from routine exploration of possible network structures. Huffman [7], Hohn and Schissler [6], as well as many other researchers contributed to the theory of switching networks over the next decades.

Since relays were very unreliable due to mechanical switching, the issue of network robustness was particularly important at that time. Large switching networks could only be built by providing a certain degree of structural redundancy; it became clear that having only functional specification of a network in terms of a system of Boolean equations is not enough and other, non-functional, requirements have to be considered. The advent of much more reliable switching elements, such as vacuum tubes and transistors, changed the course of the switching theory, however. The research activity shifted towards automation of large- and very-large-scale integrated (VLSI) circuits [3][2][11][14].

As semiconductor technology marched forward, the cost of manufacturing a single transistor rapidly tended towards zero and in the last decade it became essentially free. By that time, manual low-level design had been abandoned, and engineers began thinking and designing systems in terms of higher-level *components* (first, logic gates, then arithmetic units, and now even whole IP cores!) and their *configurations* rather than in terms of transistors and electrical circuits. Hidden under the multiple layers of abstraction, the switching networks were no longer synthesised directly; instead, they were structurally put together to allow as much component reuse as possible and to avoid extremely expensive re-design, re-thinking, verification and test.

An unavoidable consequence of the shrinking size of a transistor was the growing uncertainty of its characteristics, leading to lower robustness and shorter life-span of large transistor networks. Individual reliability as well as collective power dissipation of transistors became dominant problems that could not be solved at the level of system components. Once again designers were forced to consider non-functional aspects of basic switching elements and their networks.

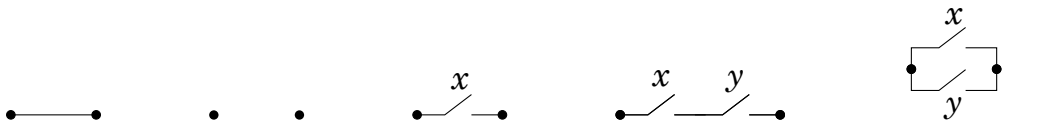
Today's newly emerging switching technologies bring new non-functional characeristics into consideration, for example, *photonic circuits* [10] use optical switches that are more energy-efficient for long distances; *MEMS (micro-electro-mechanical systems) switches* [9] are slower than MOSFET transistors but do not leak current and have other unique characteristics making them attractive in the context of *energy modulated computing* [17]. All evidence suggests that methodologies for formal reasoning on non-functional properties of switching networks are of immediate need in the microelectronic industry.

This paper introduces a new *algebra of switching networks* with the aim of providing a mathematical instrument for reasoning on both functional and non-functional (i.e. structural) properties of a network of interconnected Boolean switches. A network is represented by an algebraic expression, and the axioms of the algebra ensure that any permissible rewriting of the expression preserves the network functionality; the network structure, however, can be changed arbitrarily, thereby allowing exploration of the non-functional design space. The problem of *analysis* therefore corresponds to rewriting a given expression into a certain normal form, while the problem of *synthesis* translates to that of simplifying an algebraic expression subject to a set of non-functional constraints.

The paper is organised as follows. Section 2 gives a background on switching networks and Boolean algebra. The algebra of switching networks is then defined axiomatically in Section 3. Examples of algebraic representation and transformation of switching networks are presented in Section 4, which is followed by conclusions in Section 5.

## 2 Switching networks

This section introduces *switching networks*, their graphical representation, and principles of Boolean analysis, first studied in this form by Shannon [15].

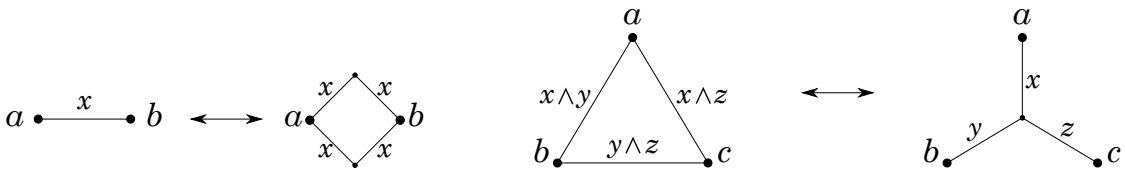



(a) $f_a = 1$ (b) $f_b = 0$ (c) $f_c = x$ (d) $f_d = x \wedge y$ (e) $f_e = x \vee y$

Figure 1: Switching networks and their connectivity functions

Structurally, switching networks consist of *nodes* and *switches* that establish an electrical connection between two particular nodes when being On, and break the connection when being Off. Figure 1 shows several simple networks and their *connectivity functions*,[1] which evaluate to 1 whenever the two nodes are connected. The first two networks correspond to the basic cases when a switch is either always On (the connectivity function in this case is $f_a = 1$) or always Off ($f_b = 0$). The next case has the simplest non-trivial connectivity function $f_c = x$, i.e. the network is connected if (and only if) $x = 1$. A series connection of two switches yields $f_d = x \wedge y$, where Boolean operator $x \wedge y$ represents the logical proposition 'x and y'; intuitively, the two nodes are connected only when both switches are On. Finally, the last case represents a parallel connection of two switches that yields $f_e = x \vee y$, where Boolean operator $x \vee y$ represents the logical proposition 'x or y'.

Using this natural Boolean interpretation of switching networks, one can write algebraic expressions corresponding to connectivity functions between any pair of nodes in a given network, thus performing its *Boolean analysis*. The inverse task of finding a switching network realising a giving set of connectivity functions is called *synthesis*. See [6] for an extensive review of these two fundamental problems.




(a) Increasing network robustness (b) Delta-wye transformation

Figure 2: Examples of functionally equivalent transformations (note that switches are represented by simple edges annotated with their connectivity functions)

Separation of structure (which is represented graphically by a switching network itself) and functionality (captured by systems of Boolean equations) leads to inability to formally reason about functional correctness of structural transformations. For example, consider two switching networks shown in Figure 2(a). The leftmost network is just a simple switch with a connectivity function $f_{ab} = x$. The rightmost network is an ingenious transformation of the former aimed at increasing its robustness. Assume that any switch can fail, i.e. become permanently On or Off, with a certain probability p. Then, in order for the transformed network to fail, at least two switches must fail, which is a much more rare event happening with a probability proportional to $p^2$. Note

[1] The function of connectivity is also known under the names of *hindrance function* [16] and *transmission function* [7].

that the transformed network has the same connectivity function $f'_{ab} = (x \wedge x) \vee (x \wedge x) = x$, thus the networks are *functionally equivalent*. However, the only way to prove the correctness of this simple structural transformation is to compute the connectivity functions and directly compare them, which is a very time-consuming process in general and cannot be routinely performed in modern networks comprising of billions of switches.

Figure 2(b) shows another example of the same problem — the famous delta-wye transformation, which preserves all the pairwise connectivity functions between the nodes a, b and c, but reduces the overall structural complexity of the network.

The aim of this paper is to develop a formal approach for reasoning about structural transformations that preserve functionality of a switching network. The next section introduces the approach, while Section 4 demonstrates its application to the examples discussed above.

# 3 Algebra of switching networks

In this section a new algebra of switching networks is introduced. The algebra builds on the formalisms of *Conditional Partial Order Graphs* [13] and *Parameterised Graphs* [12] and inherits their basic principle: it achieves a compact representation of multiple graphs by overlaying them in the form of a graph annotated with Boolean conditions. It should be noted that graphs with conditions have been historically used to represent switching networks (see, for example, works by Shannon [15][16] and Bryant [3][2]), however, structural operations on graphs were never axiomatised and functional correction of each structural transformation had to be separately proved in an ad hoc manner, therefore not suitable for automation.

## 3.1 Operations

A natural model for a *switching network* is an undirected graph $N = (V, E)$ [8], where a set of *vertices* $V$ corresponds to the network *nodes*, and a set of *edges* $E \subseteq V \times V$ models a state of switches connecting the nodes. We assume that all vertices belong to a fixed universe $\mathcal{V}$. For any pair $u, v \in V$ of network nodes, if there is a *closed switch* between them (i.e., the nodes are connected) then an unordered pair of vertices $(u, v)$ belongs to the set of edges, which will be denoted as $uv \in E$ for brevity. Otherwise, if there is an *open switch* between them or no switch at all (i.e., the nodes are disconnected), then the corresponding edge is missing: $uv \notin E$. It is assumed that a node is connected to itself, so there is a self-loop $vv \in E$ for any $v \in V$. An *empty network* $(\emptyset, \emptyset)$ will be further denoted by $\varepsilon$, and the *singleton networks* $(\{v\}, \{(v, v)\})$ simply by $v$, for any $v \in V$.

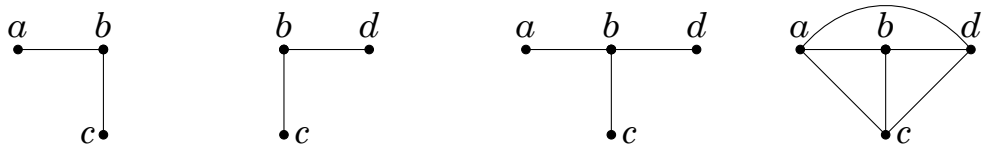
Let $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$ be two switching networks; here $V_1$ and $V_2$, as well as $E_1$ and $E_2$, are not necessarily disjoint. The following three basic operations on switching networks are defined (in the order of increasing precedence):

Overlay: $N_1 + N_2 \stackrel{\text{df}}{=} (V_1 \cup V_2, E_1 \cup E_2)$.

Connection: $N_1 \multimap N_2 \stackrel{\text{df}}{=} (V_1 \cup V_2, E_1 \cup E_2 \cup V_1 \times V_2)$.

Condition: $[1]N \stackrel{\text{df}}{=} N$ and $[0]N \stackrel{\text{df}}{=} \varepsilon$.

In other words, the *overlay* $+$ and *connection* $\multimap$ are binary operations on networks with the following semantics: $N_1 + N_2$ is a network obtained by *overlaying* networks $N_1$ and $N_2$, i.e. it contains the union of their nodes and edges, while network $N_1 \multimap N_2$ contains the union plus the edges connecting every node from network $N_1$ to every node from network $N_2$. One can observe that any non-empty network can be obtained by successively applying the operations $+$ and $\multimap$ to the singleton networks. An example in Figure 3 illustrates these operations.




(a) Network $N_1$ (b) Network $N_2$ (c) Overlay $N_1 + N_2$ (d) Network $N_1 — N_2$

Figure 3: Example of operations on switching networks (self-loops are not shown)

Given a network N, the unary *condition* operations can either preserve it (*true condition* [1]N) or nullify it (*false condition* [0]N). These two operations are not particularly useful until one considers replacing a Boolean constant 0 or 1 with a Boolean variable or a general predicate, say, x, resulting in an expression [x]N, whose value depends on the value of *parameter* x. This subtle conceptual step (which is akin going from arithmetic to algebraic expressions) brings up a new algebra with interesting properties, capable of describing both functionality and structure of a switching network.

## 3.2 Axioms

By successive application of the operations defined above, one can construct *expressions* using the empty network $\varepsilon$, the singleton networks $v \in V$, and Boolean predicates (parameters for the conditional operations) as the basic building elements. A natural question immediately arises: given two expressions, how to decide if they represent networks with the same functionality?

Semantically, two switching networks are considered to be *equivalent* if they have the same set of nodes, and the connectivity function between any pair of nodes in the first network equals that of the corresponding pair in the second network. Alternatively, one can define two networks $N_1 = (V_1, E_1)$ and $N_2 = (V_2, E_2)$ to be equivalent iff their *transitive closures* $N_1^* = (V_1, E_1^*)$ and $N_2^* = (V_2, E_2^*)$ coincide $N_1^* = N_2^*$. For example, the two networks in Figures 3(c) and 3(d) are equivalent because all their nodes are connected to each other either directly or transitively via intermediate nodes.

*The algebra of switching networks* is a tuple $\langle \mathcal{N}, +, —, [0], [1] \rangle$, where $\mathcal{N}$ is a set of switching networks whose nodes are picked from some universe $\mathcal{V}$ and the operations are as described above. The equivalence relation can be abstractly defined by the following set of *axioms*:

- $+$ and $—$ are commutative and associative operations:
  - $p + q = q + p$ and $p — q = q — p$
  - $(p + q) + r = p + (q + r)$ and $(p — q) — r = p — (q — r)$
- $\varepsilon$ is an identity of $—$: $p — \varepsilon = p$
- $—$ distributes over $+$: $p — (q + r) = p — q + p — r$
- decomposition: $p — q — r = p — q + p — r + q — r$
- closure: if $q \neq \varepsilon$ then $p — q + q — r = p — q + p — r + q — r$
- condition: $[0]p = \varepsilon$ and $[1]p = p$

One can easily check that the axioms are satisfied at the semantic level of switching networks introduced in the previous subsection, thus the algebra is *sound*. *Completeness* follows from the existence of the canonical form introduced in the next subsection. *Minimality* of the set of axioms can be proved by enumerating the fixed-size models of the algebra with the help of the Alg tool [1]: It turns out that removing any of the axioms leads to a different number of non-isomorphic models of a particular size, implying that all the axioms are necessary. Hence the following result holds.

**Theorem 1** (Soundness, Minimality and Completeness). *The set of axioms of the algebra of switching networks is sound, minimal and complete.*

The following useful equalities can be proved from the axioms (the equalities involving conditions can be reduced to the axioms by case analysis on the values of Boolean parameters):

- $\varepsilon$ is an identity of $+$: $p + \varepsilon = p$
- $+$ is idempotent: $p + p = p$
- absorption: $p + p — q = p — q$
- conditional overlay: $[x](p + q) = [x]p + [x]q$
- conditional connection: $[x](p — q) = [x]p — [x]q$
- AND-condition: $[x \wedge y]p = [x][y]p$
- OR-condition: $[x \vee y]p = [x]p + [y]p$

Finally, for the sake of convenience a ternary operation called a *switch* is introduced as a combination of the three basic operations:

$$p \overset{x}{—} q \overset{\text{df}}{=} p + q + [x](p — q)$$

As its name suggests, the operation corresponds to a switch connecting two networks $p$ and $q$, which is On when $x = 1$ and Off when $x = 0$.

**Example 2.** The following simple algebraic expressions represent the switching networks shown in Figure 1: $N_a = a — b$, $N_b = a + b$, $N_c = a \overset{x}{—} b$, $N_d = a \overset{x \wedge y}{—} b$ and $N_e = a \overset{x \vee y}{—} b$.

The next subsection describes the notion of the canonical form of an algebraic expression and demonstrates the process of Boolean analysis in the algebraic context.

## 3.3 Canonical form

Any algebraic expression representing a switching network (further called a *switching expression* for short) can be rewritten in the *canonical form* as stated by the following proposition. This result can be proved by adapting the canonical form of *Transitive Parameterised Graphs* presented in [12].

**Proposition 3** (Canonical form). *Any switching expression can be rewritten in the following canonical form:*

$$\left( \sum_{v \in V} [f_v]v \right) + \left( \sum_{\substack{u,v \in V \\ u \neq v}} [f_{uv}](u — v) \right),$$

*where:*

1. *$V$ is a subset of singleton graphs that appear in the original expression;*
2. *for all $v \in V$, $f_v$ are canonical forms of Boolean expressions and are distinct from 0;*
3. *for all $u, v \in V$, $u \neq v$, $f_{uv}$ are canonical forms of Boolean expressions such that $f_{uv} \Rightarrow f_u \wedge f_v$ (this requirement ensures that a switch cannot appear without its nodes);*
4. *for all $u, v, w \in V$, $f_{uv} \wedge f_{vw} \Rightarrow f_{uw}$ (the transitivity requirement, i.e. if nodes $u$ and $v$ are connected and so are nodes $v$ and $w$ then nodes $u$ and $w$ must also be connected).*

Intuitively, the canonical form corresponds to a graphical representation of a given switching expression: each node $v \in V$ is represented as a vertex with condition $f_v$, and each pair of nodes is connected by a switch representing the connectivity function $f_{uv}$ between the nodes specified by the original expression. The process of constructing the canonical form of an expression directly corresponds to Boolean analysis of the corresponding network, in particular, the obtained matrix $(f_{uv})$ is called the *switching matrix* in the context of classical Boolean analysis [6].

Let's use the canonical form to formally prove that switching networks in Figures 3(c) and 3(d) are equivalent. The two switching expressions are:

$$\begin{array}{rcccc} s_1 & = & N_1 + N_2 & = & (a \text{---} b + b \text{---} c) + (b \text{---} c + b \text{---} d) \\ s_2 & = & N_1 \text{---} N_2 & = & (a \text{---} b + b \text{---} c) \text{---} (b \text{---} c + b \text{---} d) \end{array}$$

The canonical form of the first expression is derived by opening the brackets, using the rule of idempotence, and applying the closure axiom to add all the transitive edges:

$$\begin{array}{rcl} s_1 & = & (a \text{---} b + b \text{---} c) + (b \text{---} c + b \text{---} d) \\ & = & a \text{---} b + b \text{---} c + b \text{---} c + b \text{---} d \\ & = & a \text{---} b + b \text{---} c + b \text{---} d \\ & = & a + b + c + d + a \text{---} b + b \text{---} c + b \text{---} d + a \text{---} c + a \text{---} d + c \text{---} d. \end{array}$$

The canonical form of the second expression is derived with the help the decomposition axiom that helps to decompose chained connections into simpler ones, and by using an observation that $\text{---}$ is idempotent for singleton networks, i.e. $v \text{---} v = v$ for any $v \in \mathcal{V}$:

$$\begin{array}{rcl} s_2 & = & (a \text{---} b + b \text{---} c) \text{---} (b \text{---} c + b \text{---} d) \\ & = & a \text{---} b \text{---} b \text{---} c + b \text{---} c \text{---} b \text{---} c + a \text{---} b \text{---} b \text{---} d + b \text{---} c \text{---} b \text{---} d \\ & = & a \text{---} b \text{---} c + b \text{---} c + a \text{---} b \text{---} d + b \text{---} c \text{---} d \\ & = & a + b + c + d + a \text{---} b + b \text{---} c + b \text{---} d + a \text{---} c + a \text{---} d + c \text{---} d. \end{array}$$

One can see that the canonical forms of $s_1$ and $s_2$ coincide, thus proving that the original networks are equivalent. In fact, it is far more easier to prove equivalence of $s_1$ and $s_2$ without going through the canonical form (i.e. without doing the complete Boolean analysis), which is one of the main advantages of the presented algebraic approach:

$$\begin{array}{rcl} s_1 & = & (a \text{---} b + b \text{---} c) + (b \text{---} c + b \text{---} d) \\ & = & (a + c) \text{---} b + b \text{---} (c + d) \\ & = & (a + c) \text{---} b \text{---} (c + d) \\ & = & (a \text{---} b + b \text{---} c) \text{---} (b \text{---} c + b \text{---} d) \\ & = & s_2. \end{array}$$

## 3.4 Node contraction

In many cases a relaxed notion of equivalence if useful, for example, to prove that the networks in Figure 2 are equivalent it is necessary to remove auxiliary nodes from one of the networks [4]. This subsection describes a procedure, called *node contraction*, that performs such a removal and thereby establishes a relaxed notion of equivalence.

Consider a switching expression $s$ and a node $t$ that appears in the expression. *Node contraction* produces a new expression $s'$ which is free from $t$ but preserves the connectivity functions for all pairs of vertices $u \neq t$ and $v \neq t$. This is formally denoted as: $s' = s \setminus t$.

**Proposition 4** (Node contraction). *Let $s$ be a switching expression with the following canonical representation:*

$$s = \left( \sum_{v \in V} [f_v] v \right) + \left( \sum_{\substack{u,v \in V \\ u \neq v}} [f_{uv}] (u \text{---} v) \right),$$

*and* $t \in V$ *be any node appearing in the expression. Then the node contraction* $s \setminus t$ *has the following canonical form:*

$$s \setminus t = \left( \sum_{v \in V \setminus t} [f_v]v \right) + \left( \sum_{\substack{u,v \in V \setminus t \\ u \neq v}} [f_{uv}](u \text{—} v) \right).$$

In other words, all the terms corresponding to node $t$ are dropped. Note that due to the transitivity requirement of the canonical form, this transformation does not affect any connectivity function in the resulting network.

Node contraction is a *confluent* transformation, i.e. the order of contractions does not matter:

$$s \setminus t_1 \setminus t_2 = s \setminus t_2 \setminus t_1.$$

This allows one to generalise contraction to sets of nodes: $s \setminus T$. For example, $s \setminus \{t_1, t_2\} = s \setminus t_1 \setminus t_2$.

The next section demonstrates application of node contraction to checking functional equivalence of the networks with auxiliary nodes.

# 4 Examples

First, consider the networks shown in Figure 2(b), which represent the 'delta' $\Delta$ and the 'wye' $Y$ in the well-known delta-wye transformation. The rightmost network $N_R$ contains an auxiliary node, which will be denoted as $t$; the leftmost network $N_L$ has no auxiliary nodes. Algebraic representations of the networks are given below:

$$\begin{array}{rcl} N_L & = & a \overset{xy}{\text{—}} b + a \overset{xz}{\text{—}} c + b \overset{yz}{\text{—}} c \\ N_R & = & a \overset{x}{\text{—}} t + b \overset{y}{\text{—}} t + c \overset{z}{\text{—}} t \end{array}$$

The canonical form of the leftmost network is:

$$\begin{array}{rcl} N_L & = & a \overset{xy}{\text{—}} b + a \overset{xz}{\text{—}} c + b \overset{yz}{\text{—}} c \\ & = & a + b + c + [xy](a \text{—} b) + [xz](a \text{—} c) + [yz](b \text{—} c) \end{array}$$

The canonical form of the rightmost network is more complicated due to the transitive edges:

$$\begin{array}{rcl} N_R & = & a \overset{x}{\text{—}} t + b \overset{y}{\text{—}} t + c \overset{z}{\text{—}} t \\ & = & a+b+c+t+[x](a \text{—} t)+[y](b \text{—} t)+[z](c \text{—} t)+[xy](a \text{—} b)+[xz](a \text{—} c)+[yz](b \text{—} c) \end{array}$$

Now, according to Proposition 4, node contraction $N_R \setminus t$ has the following canonical form (all terms containing $t$ are removed):

$$N_R \setminus t = a + b + c + [xy](a \text{—} b) + [xz](a \text{—} c) + [yz](b \text{—} c)$$

which coincides with the canonical form of $N_L$, thus proving equivalence of the two networks. Note that the original expressions $a \overset{xy}{\text{—}} b + a \overset{xz}{\text{—}} c + b \overset{yz}{\text{—}} c$ and $a \overset{x}{\text{—}} t + b \overset{y}{\text{—}} t + c \overset{z}{\text{—}} t$ perfectly capture the structure of the networks despite having the same functionality, and by using the algebraic transformations one can transform one of them into another for the purpose of optimising a non-functional criteria, such as, for example, the overall complexity of the network in terms of the number of switches, thereby performing a form of *synthesis*.

Finally, consider the transformation for increasing robustness of a single switch by replacing it with four identical switches connected in a bridge structure, as shown in Figure 2(a). The leftmost and rightmost networks are specified algebraically by expressions $N_L$ and $N_R$, respectively:

$$\begin{array}{rcl} N_L & = & a \overset{x}{\text{—}} b \\ N_R & = & (a + b) \overset{x}{\text{—}} (t_1 + t_2) \end{array}$$

where $T = \{t_1, t_2\}$ represents the set of auxiliary signals. $N_R$ can be rewritten into the canonical form by opening brackets and deducing the transitive connections using the closure axiom:

$$\begin{aligned} N_R &= (a+b)\overset{x}{—}(t_1+t_2) \\ &= a\overset{x}{—}t_1 + a\overset{x}{—}t_2 + b\overset{x}{—}t_1 + b\overset{x}{—}t_2 \\ &= a+b+t_1+t_2+[x](a—t_1)+[x](a—t_2)+[x](b—t_1)+[x](b—t_2)+[x](a—b)+[x](t_1—t_2) \end{aligned}$$

Now one can obtain a contracted network $N_R \setminus T = a + b + [x](a—\!—b)$, which is equivalent to $N_L$, as expected.

### 4.1 Structural and functional compositions

One particularly unique and useful feature of the algebra is *compositionality*. Consider two systems A and B. Their *structural composition* corresponds to connecting their common interface wires together obtaining a larger system C. Algebraically this is represented simply as $C = A+B$. While this type of composition is usually handled well by other methods, our approach also allows to perform *functional composition* of systems, which is a lot more difficult to handle.

For example, if one wants to describe a system C that delivers functionality A under a certain condition x, and functionality B under condition $\overline{x}$ then the corresponding algebraic representation is $C = [x]A+[\overline{x}]B$. Importantly, if the two functionalities are similar, one can simplify the resulting expression. For instance, $[x]A + [\overline{x}]A = [x \vee \overline{x}]A = A$, i.e., if the functionalities coincide one can algebraically prove it and remove the condition x altogether.

Finally, the axiomatic definition of switching network equivalence allows a designer to substitute a part of an expression A with an equivalent part B without any additional checks of the resulting system global properties. As long as the local equivalence $A = B$ holds, it is guaranteed that the rest of the system is not affected by the substitution.

Algebraic compositionality opens way for new methodologies and techniques for system optimisation in various aspects, such as latency, power consumption, reliability, etc. by performing local provable transformations of an expression representing an entire system.

## 5 Conclusions

This paper discusses the glorious past of the theory of switching networks, whose roots can be traced back to the beginning of the XX century. Today, the theory forms the backbone of any computation system, however, it is hidden by multiple layers of abstraction and largely forgotten; little or no development is going on at present: it is believed that all the useful facts about switching networks have already been discovered.

This work is an attempt to revive the old theory by introducing a new mathematical construct – an algebra of switching networks – that unifies the notions of function and structure of a computation system that were always separated. The algebra is specified axiomatically, and the soundness, minimality and completeness of the resulting sets of axioms are proved. The transformations for the canonical form derivation and node contraction are developed and demonstrated on a set of examples.

**Acknowledgement**
The author would like to thank Marc Riedel for introduction to the world of Shannon's switching networks, Victor Khomenko for inspiration to creating new algebrae, Alex Yakovlev for continued contribution to the theory of conditional graphs, as well as all the colleagues at Newcastle University. This research was supported by the Epsrc grant EP/J008133/1 (TrAmS-2).

## References


[1] A. Bizjak and A. Bauer. Alg *User Manual*, Faculty of Mathematics and Physics, University of Ljubljana, 2011.

[2] R. E. Bryant. Algorithmic aspects of symbolic switch network analysis. *IEEE Transactions on CAD of Integrated Circuits*, 6:618–633, 1987.

[3] R. E. Bryant. Boolean analysis of MOS circuits. *IEEE Transactions on Computer-aided Design*, 6:634–649, 1987.

[4] W. Chen. Boolean matrices and switching nets. *Mathematics Magazine*, pages 1–8, January 1966.

[5] L. Couturat. The Algebra of Logic. *Open Court Publishing Company*, 1914.

[6] F. E. Hohn and L. R. Schissler. Boolean Matrices and Combinational Circuit Design. *Bell Systems Technical Journal*, (34):177–202, 1955.

[7] D.A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954.

[8] Art Lew. *Computer Science: A Mathematical Introduction.* Prentice-Hall, 1985.

[9] T.-J. K. Liu, D. Markovic, V. Stojanovic, and E. Alon. MEMS Switches for Low-Power Logic. *IEEE Spectrum*, April 2012.

[10] A. D. McAulay. *Optical computer architectures: the application of optical concepts to next generation computers*. John Wiley & Sons, Inc., 1991.

[11] C. Mead and L. Conway. *Introduction to VLSI systems*. Addison-Wesley, 1980.

[12] A. Mokhov, V. Khomenko, A. Alekseyev, and A. Yakovlev. Algebra of Parameterised Graphs. Technical Report CS-TR-1307, School of Computing Science, Newcastle University, 2011. URL: http://www.cs.ncl.ac.uk/publications/trs/abstract/1307; accepted for ACSD'12 conference.

[13] A. Mokhov and A. Yakovlev. Conditional Partial Order Graphs: Model, Synthesis and Application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

[14] J. E. Savage. *Models of Computation – Exploring the Power of Computing*. Addison-Wesley, 1998.

[15] C. E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Transactions of the American Institute of Electrical Engineers*, 57:713–723, 1938.

[16] C. E. Shannon. The Synthesis of Two-Terminal Switching Circuits. *Bell Systems Technical Journal*, 28:59–98, 1948.

[17] Alex Yakovlev. Energy-modulated computing. In *Design Automation and Test in Europe (DATE) conference*, pages 1340–1345, 2011.