The Logic in Computer Science Column

by

Yuri Gurevich

Computer Science and Engineering
University of Michigan, Ann Arbor, MI 48109, USA
gurevich@umich.edu

CIRCUITS: AN ABSTRACT VIEWPOINT

Andreas Blass and Yuri Gurevich

Abstract

Our primary purpose is to isolate the abstract, mathematical properties of circuits — both classical Boolean circuits and quantum circuits — that are essential for their computational interpretation. A secondary purpose is to clarify the similarities and differences between the classical and quantum situations.

The general philosophy in this note is to include the mathematically essential aspects of circuits but to omit any of the additional structures that are usually included for convenience. We shall, however, retain the assumption that circuits are finite; this assumption does no harm to the applicability of our approach and is necessary for some of our work.

One of the endearing things about mathematicians is the extent to which they will go to avoid doing any real work.

— Matthew Pordage

1 Introduction

As we worked on Circuit Pedantry [1], we tried to figure out the appropriate level of abstraction for Boolean and quantum circuits. There is a natural tendency in the sciences, but especially in mathematics, to abstract away as many details as possible. This celebrated tendency is fruitful but it may also be fraught with troubles of various kinds.

The story goes that Plato defined man as featherless biped, abstracting from man's many other properties, but Diogenes plucked the feathers from a cock and brought it to Plato saying: "Here's your man."

A more recent example is Cantor's definition of sets. Cantor imposed no restrictions on what the elements of a set can be or how they are collected into a

whole. The notion of set simplified mathematical analysis, enabled the development of logic and topology, etc. But it also led to paradoxes.

Here is a more pedestrian example which is closer to Circuit Pedantry. A finite matrix can be defined as an indexed set $\{e_p : p \in R \times C\}$ where R and C are finite sets indexing rows and columns. Normally the rows are linearly ordered and so are the columns. But the notion of finite matrix makes perfect sense without those orderings. That is, until you want to draw a matrix on a whiteboard or on paper.

Still, there are advantages in dealing with sets and indexed sets rather than linearly ordered sets. Think of relational databases where relational tuples are unordered, which simplifies theory [8] and improves practice [6] by eliminating a most important source of implementation dependence.

The set-based approach supports the most expressive polynomial-time computation model in the literature where machines do not distinguish between isomorphic structures [3, 4, 7]. Some complexity-theoretic advantages of the set-based approach are demonstrated in [5].

In Circuit Pedantry, we restrained our own tendency to abstract and adopted the traditional approach. In that approach, the input nodes are ordered in an arbitrary way and — in the case of quantum circuits (and balanced Boolean circuits) — one fixes a bijection between incoming and outgoing edges for every gate. As a result, there are definite timelines from the input nodes to output nodes. This allowed us to use traditional circuit diagrams as in, for example, [9] and hopefully to make that paper more readable.

But before we restrained our tendency to abstract, we indulged it for a little while. There is elegance and mathematical utility in the more abstract view. We are using the abstract view in our forthcoming paper on Quantum Circuits with Classical Channels [2] and we illustrate it here.

Our primary purpose in the present paper is to isolate the abstract, mathematical properties of circuits — both classical Boolean circuits and quantum circuits — that are essential for their computational interpretation. A secondary purpose is to clarify the similarities and differences between the classical and quantum situations.

Our general philosophy in this note is to include the mathematically essential aspects of circuits but to omit any of the additional structures that are usually included for convenience. We shall, however, retain the assumption, satisfied in theoretical as well as applied work, that circuits are finite. This assumption does no harm to the applicability of our approach and is necessary for some of our

work.

In the rest of this introduction, we describe how we want to view circuits. Precise details will be given in later sections.

Inputs and outputs, whether of a whole circuit or of a single gate, will be families (of Boolean values or of quantum states, usually qubits and possibly entangled) indexed by some finite sets. It is customary to index inputs and outputs by natural numbers, thereby imposing a linear ordering on the inputs and another linear ordering on the outputs. When the number of inputs equals the number of outputs, we thereby obtain a particular bijection between the inputs and outputs. Although such bijections are useful for drawing circuits, we shall see that none of this customary extra structure — numerical indexes, linear orders, particular bijections — is essential for mathematical purposes; indeed none of this structure will appear in the formal development below. Any gate G will have a finite set ι_G of input labels and a finite set o_G of output labels, but there will be no additional structure or assumptions on these sets. (We use the Greek letters iota and omicron for input and output of gates, in order to keep i available for other uses.)

In the case of Boolean gates, the whole input will be an ι_G-indexed family of Boolean values, i.e., an element of $\{0,1\}^{\iota_G}$, and the output will be an element of $\{0,1\}^{o_G}$. In the case of quantum gates, the input and output will be vectors in $Q^{\otimes \iota_G}$ and $Q^{\otimes o_G}$, respectively, where Q is our basic Hilbert space, usually $\mathbb{C}^2$, the state space for a qubit.

Similarly, the circuit as a whole will have input nodes indexed by a finite set I and output nodes indexed by another finite set O, with no additional structure or assumptions. The input to such a circuit will be in $\{0,1\}^I$ in the Boolean case or $Q^{\otimes I}$ in the quantum case. The output will be in $\{0,1\}^O$ in the Boolean case or $Q^{\otimes O}$ in the quantum case.

The connections between gates, inputs, and outputs will also be described in what we believe to be the simplest reasonable way. Wherever a value (Boolean or quantum) is needed, there will be a pointer to a provider for that value. A value is needed at each input position of a gate and at each output node of the whole circuit. Potential providers for these values are the circuit's input nodes and the gates' output positions. We call the places where a value is needed "consumers" and the places where a value can be obtained "producers". So the wiring of our circuits will be given by a "provider" function π from consumers to producers, giving for each consumer c a producer $\pi(c)$ expected to supply the value needed by c.

2 Preliminaries

Throughout this paper we shall need to work with families indexed by arbitrary finite sets, in contexts where indexing by natural numbers is more common and provides a specific ordering for the elements of the family. This preliminary section is devoted to describing how our more general sort of indexing works and fixing our notation for it.

Convention 1. Throughout this paper, index sets are assumed to be finite.

An I-indexed *family* is a function x with domain I. The usual notations for the value of x at i are $x(i)$ and x_i. The family itself is usually written $\langle x_i : i \in I\rangle$ or $\langle x_i\rangle_{i\in I}$. The x_i's are called the *elements* or *components* of the family.

When the index set I is $\{1, 2, \ldots, n\}$, one may write such an indexed family as $\langle x_1, x_2, \ldots, x_n\rangle$ and call it an *n-tuple*.

We use the notation $\bigsqcup_{i\in I} A_i$ for the *disjoint union* of a family $\langle A_i : i \in I\rangle$ of sets, defined as

$$\bigsqcup_{i\in I} A_i = \{\langle i, a\rangle : i \in I \text{ and } a \in A_i\}.$$

In other words, we replace all the sets A_i by pairwise disjoint, bijective copies, $\{i\} \times A_i$, and then we take the union of those copies. If the A_i's are themselves pairwise disjoint, then we could have just taken their union without copying, and we may tacitly identify that union with the official disjoint union defined above. Even when the A_i's are not disjoint, we may tacitly identify elements a of A_i with the corresponding elements $\langle i, a\rangle$ of $\bigsqcup_{i\in I} A_i$, relying on the context to provide the correct i.

When the index set I is $\{1, 2, \ldots, n\}$, one may write the disjoint union as

$$\bigsqcup_{i\in\{1,2,\ldots,n\}} A_i = \bigsqcup_{i=1}^{n} A_i = A_1 \sqcup A_2 \sqcup \cdots \sqcup A_n.$$

In particular, we have the binary operation $\sqcup$ as in $A_1 \sqcup A_2$.

The *(Cartesian) product* of a family $\langle A_i : i \in I\rangle$ is defined as the collection of those I-indexed families whose elements are taken from the corresponding sets A_i. That is,

$$\prod_{i\in I} A_i = \{\langle x_i : i \in I\rangle : (\forall i \in I)\, x_i \in A_i\}.$$

As in the case of disjoint unions, alternative notations may be used when $I =$

$\{1, 2, \ldots, n\}$, namely

$$\prod_{i\in\{1,2,\ldots,n\}} A_i = \prod_{i=1}^{n} A_i = A_1 \times A_2 \times \cdots \times A_n.$$

In particular, we have the binary operation $\times$ as in $A_1 \times A_2$.

Similar conventions apply to the tensor product of vector spaces. The tensor product of an indexed family $\langle V_i : i \in I\rangle$ of vector spaces can be defined as the vector space generated by the elements of $\prod_{i\in I} V_i$, considered as formal symbols, modulo the relations that make the generators linear functions of each component when the other components are held fixed.

The precise definitions will be given in a moment, but let us first give an orienting example with $I = \{1, 2, 3\}$. A typical generator $\langle x_1, x_2, x_3\rangle$ would, in the context of tensor products, often be written as $x_1 \otimes x_2 \otimes x_3$, and a fairly typical relation would be the distributivity equation

$$(x_1 \otimes p \otimes x_3) + (x_1 \otimes q \otimes x_3) = x_1 \otimes (p + q) \otimes x_3.$$

To describe these relations in more detail and in full generality, it is convenient to introduce a bit of notation. If $\langle x_i : i \in I\rangle$ is an indexed family, if $j \in I$, and if q is an arbitrary entity, then we write "$\langle x_i : i \in I\rangle$ but $j \mapsto q$" for the i-indexed family $\langle x'_i : i \in I\rangle$ where $x'_i = x_i$ for all $i \neq j$ but $x'_j = q$. That is, we modify the original famiiy $\langle x_i : i \in I\rangle$ by changing the j-component to q. Then the linearity relations for the tensor product are, first, for all $j \in I$ and all $p, q \in V_j$,

$$(\langle x_i : i \in I\rangle \text{ but } j \mapsto p) + (\langle x_i : i \in I\rangle \text{ but } j \mapsto q) = \\ = \langle x_i : i \in I\rangle \text{ but } j \mapsto p + q,$$

and second, for all $j \in I$, all $q \in V_j$, and all scalars λ,

$$\lambda(\langle x_i : i \in I\rangle \text{ but } j \mapsto q) = \langle x_i : i \in I\rangle \text{ but } j \mapsto \lambda q.$$

We use the notation $\bigotimes_{i\in I} V_i$ for this tensor product. As before, when $I = \{1, 2, \ldots, n\}$, we have the alternative notations

$$\bigotimes_{i\in\{1,2,\ldots,n\}} A_i = \bigotimes_{i=1}^{n} A_i = A_1 \otimes A_2 \otimes \cdots \otimes A_n,$$

and we have the binary operation $\otimes$ as in $A_1 \otimes A_2$.

When people work with numerical index sets and with the binary operations $\sqcup$, $\times$, and $\otimes$, they make extensive (but often tacit) use of the commutative and associative laws (up to canonical isomorphism) for these operations, and the main effect of these laws is to render the numerical indexing irrelevant. In our general indexed context, these laws take on quite different forms. Commutativity in the numerical-indexed context allows one to change the order of the operands, but our operands don't come with an order. Associativity in the numerical context allows one to regard operations on three or more operands as built up from binary operations in various ways, but we have defined the n-ary and in fact I-ary operations directly, not in terms of binary ones. We list below the more general laws governing our more general operations. In each case, the isomorphisms indicated by $\cong$ are obvious and will be referred to as canonical. We leave the routine verifications to the reader.

Suppose $f : I \to J$ is a bijection. Then any J-indexed family, being a function with domain J, can be composed with f to produce an I-indexed family, with the same components but differently indexed. In symbols, composition with f transforms $\langle x_j : j \in J \rangle$ into $\langle x_{f(i)} : i \in I \rangle$. Then we have, for any families $\langle A_j : j \in J \rangle$ of sets and $\langle V_j : j \in J \rangle$ of vector spaces,

$$\bigsqcup_{i \in I} A_{f(i)} \cong \bigsqcup_{j \in J} A_j$$
$$\prod_{i \in I} A_{f(i)} \cong \prod_{j \in J} A_j$$
$$\bigotimes_{i \in I} V_{f(i)} \cong \bigotimes_{j \in J} V_j.$$

That is, up to canonical isomorphisms, re-indexing doesn't change disjoint unions, Cartesian products, and tensor products.

Now suppose $\langle J_i : i \in I \rangle$ is an I-indexed family of index sets J_i, and let K be the disjoint union of all the J_i (remember that elements of K have the form $\langle i, j \rangle$ with $i \in I$ and $j \in J_i$). Then we have, for any K-indexed families $\langle A_{\langle i,j \rangle} : \langle i, j \rangle \in K \rangle$

of sets and $\langle V_{\langle i,j\rangle} : \langle i,j\rangle \in K\rangle$ of vector spaces,

$$\begin{aligned}
\bigsqcup_{\langle i,j\rangle\in K} A_{\langle i,j\rangle} &\cong \bigsqcup_{i\in I}\bigsqcup_{j\in J_i} A_{\langle i,j\rangle}\\
\prod_{\langle i,j\rangle\in K} A_{\langle i,j\rangle} &\cong \prod_{i\in I}\prod_{j\in J_i} A_{\langle i,j\rangle}\\
\bigotimes_{\langle i,j\rangle\in K} V_{\langle i,j\rangle} &\cong \bigotimes_{i\in I}\bigotimes_{j\in J_i} V_{\langle i,j\rangle}.
\end{aligned}$$

We shall sometimes simplify notation by omitting mention of the canonical bijections and isomorphisms above. For example, if $X = A \times C$ and $Y = B \times D$, then we may identify $A \times B \times C \times D$ with $X \times Y$, omitting mention of the canonical isomorphism arising from the bijection $f : I = \{1,2,3,4\} \to J = \bigsqcup_{i\in\{1,2\}}\{1,2\}$ that sends $1,2,3,4$ to $\langle 1,1\rangle, \langle 2,1\rangle, \langle 1,2\rangle, \langle 2,2\rangle$, respectively. In detail, let $A, B, C, D = A_{11}, A_{21}, A_{12}, A_{22}$. Then

$$\begin{aligned}
A \times B \times C \times D = A_{11} \times A_{21} \times A_{12} \times A_{22} &= \prod_{i\in I} A_{f(i)}\\
&\cong \prod_{j\in J} A_j\\
&\cong \prod_{u\in\{1,2\}}\prod_{v\in\{1,2\}} A_{\langle u,v\rangle} = (A \times C) \times (B \times D).
\end{aligned}$$

Consider two I-indexed families of sets $\langle A_i : i \in I\rangle$ and $\langle B_i : i \in I\rangle$ and an I-indexed family of functions $f_i : A_i \to B_i$. These functions f_i induce functions on disjoint unions and Cartesian products

$$\langle i, a\rangle \mapsto \langle i, f_i(a)\rangle : \bigsqcup_{i\in I} A_i \to \bigsqcup_{i\in I} B_i$$

and

$$\langle a_i : i \in I\rangle \mapsto \langle f_i(a_i) : i \in I\rangle : \prod_{i\in I} A_i \to \prod_{i\in I} B_i.$$

Similarly, linear transformations between vector spaces $f_i : V_i \to W_i$ induce a linear transformation of the tensor products, $\bigotimes_{i\in I} V_i \to \bigotimes_{i\in I} W_i$, sending each generator $\langle x_i : i \in I\rangle$ of the former space to the generator $\langle f_i(x_i) : i \in I\rangle$ of the latter. It is easy to check, using the linearity of the f_i's, that this mapping of the generators respects the defining relations of the tensor product and thus gives a well-defined linear transformation of the tensor products.

Remark 2. For category-minded readers, we mention that $\bigsqcup$ and $\prod$ are functors from I-indexed families of sets to sets. In fact, they are the left and right adjoints, respectively, of the functor that sends any set X to the I-indexed family all of whose components are X. Similarly, $\bigotimes$ is a functor from I-indexed families of vector spaces to vector spaces. The canonical isomorphisms indicated earlier are natural isomorphisms in the category-theoretic sense.

3 Boolean Circuits

Definition 3. A *Boolean gate type* is a triple $\langle \iota, o, g \rangle$ consisting of two finite sets ι and o and a function $g : \{0,1\}^\iota \to \{0,1\}^o$. We call ι the set of input labels, o the set of output labels, and g the function of the gate type.

In this section, all gate types under consideration will be Boolean, so we omit "Boolean" and just call them gate types.

People often restrict the labels to be natural numbers. This makes it easier to write elements of $\{0,1\}^\iota$ and $\{0,1\}^o$, but it has no mathematical significance.

Definition 4. A *Boolean circuit* consists of

- a finite set I of *input nodes*,
- a finite set O of *output nodes*,
- a finite set of *gates*,
- an assignment of a gate type (ι_G, o_G, g_G) to each gate G, and
- a *provider* function π as described below.

By *producers* we mean input nodes and triples of the form (G, out, l) where G is a gate of the circuit and l is one of its output labels ($l \in o_G$). We call such a triple an *output port* of the gate G. By *consumers* we mean output nodes and triples of the form (G, in, l) where G is a gate of the circuit and l is one of its input labels ($l \in \iota_G$). We call such a triple an *input port* of the gate G. Producers and consumers are called *nodes* of the circuit.

The provider function π is a function from consumers to producers subject to the following requirement. We say that a gate G is a *direct prerequisite* for another gate H and we write $G \prec H$ if π maps (at least) one of the input ports of H to an output port of G. We require that the relation $\prec$ be acyclic. ◁

To simplify terminology and notation, we shall sometimes refer to providers of a gate when we mean providers of that gate's input ports. Thus, $G \prec H$ if and only if some provider of H is an output port of G.

Notation 5. When G is a gate, we abbreviate $\{\pi(G, \text{in}, l) : l \in \iota_G\}$ as $\pi(G)$.

In view of the assumption that our circuits are finite, the requirement that $\prec$ be acyclic is equivalent to requiring that it be a well-founded relation. We use the word *prerequisite* without "direct" and the notation $\prec^*$ for the transitive closure of $\prec$; thus $\prec^*$ is a strict partial order.

The intuition behind the definition is as follows. Each gate G, given the set ι_G of input labels, the set o_G of output labels, and the Boolean function g_G, reads an input in $\{0,1\}^{\iota_G}$ from its input ports, applies g_G, and puts the result in $\{0,1\}^{o_G}$ at its output ports. The inputs here, at the input ports x of G, are simply retrieved from the corresponding nodes $\pi(x)$ as given by the provider function π. The gate G consumes its inputs and produces its outputs; hence the "producer" and "consumer" terminology. The input nodes of the circuit, the elements of I, can also provide inputs for gates, so they count as producers. The output nodes can retrieve values computed by gates or supplied in the input (as given by π) and exhibit them as the result of the circuit's computation. This intuition is formalized in the following theorem.

Theorem 6. *Let a circuit be given along with an assignment of Boolean values to its input nodes, i.e., an element a of $\{0,1\}^I$. Then there is a unique function C assigning to each node x of the circuit a Boolean value $C(x)$ subject to the following requirements.*

1. *For input nodes x, we have $C(x) = a(x)$.*

2. *For consumers x, we have $C(x) = C(\pi(x))$ (i.e., consumer nodes just retrieve bits from their providers).*

3. *For any gate G, its o_G-tuple of outputs,*

$$l \mapsto C(G, \text{out}, l),$$

is the result of applying its function g_G to its ι_G-tuple of inputs

$$m \mapsto C(G, \text{in}, m).$$

Proof. Clause (2) reduces the problem to defining C on producers. Rewriting clause (3) in terms of producers,

$$(l \mapsto C(G, \text{out}, l)) = g_G(m \mapsto C(\pi(G, \text{in}, m))),$$

we find that this and clause (1) constitute a definition of C (on producers) by recursion on the direct prerequisite relation $\prec$. Since this relation is well-founded, the recursion has a unique solution. □

According to the theorem, any $a \in \{0,1\}^I$ gives rise, via the function C, to a uniquely defined element $b \in \{0,1\}^O$, namely the restriction of C to output nodes. In this way, the given circuit defines a function $\{0,1\}^I \to \{0,1\}^O$, the function *computed* by the circuit.

4 Balanced Boolean Circuits

In preparation for the discussion of quantum circuits, we introduce a special class of Boolean circuits, defined in [1] and designed to be subject to some of the restrictions that become necessary when one moves from the classical world to the quantum world.

Definition 7. A Boolean circuit is *balanced* if all of its gate functions g_G and its provider function π are bijective.

Since bijective functions are invertible, bijectivity of all the gate functions g_G says that the circuit is composed entirely of reversible gates.

Injectivity of the provider function π means that each input bit and each bit produced by a gate can be used (or output) only once. This amounts to saying that the gates and inputs have no fan-out.

Surjectivity of π means that input bits and bits produced by gates must be used, either in computations by subsequent gates or as output from the circuit. They cannot simply be discarded. Intuitively, this seems to be a mild requirement because, if a circuit did discard some of its produced bits, then we could simply regard those bits as additional output. In other words, if π were merely injective and not bijective, we could enlarge O and extend π to map the new elements of O to those producers that were missing from the image of π.

We record some immediate consequences of the definition.

First, if G is a gate in a balanced circuit, then, since $g_G : \{0,1\}^{\iota_G} \to \{0,1\}^{o_G}$ is a bijection, the index sets ι_G and o_G must have the same cardinality; each gate has equally many input as output ports.[1]

Summing that equality over all gates, we find that the total number of gate input ports, which is the number of consumers except for the circuit's output nodes, must equal the total number of gate output ports, which is the number of producers except for the circuit's input nodes.

But the provider function π is also required to be a bijection, so the number of consumers equals the number of producers, without the exceptions. Therefore, the exceptions must match, i.e., the circuit has as many input nodes as output nodes: $|I| = |O|$.

5 Quantum Gates and Circuits

We turn now to the description of circuits for quantum computation. For simplicity and to maintain similarity with the Boolean case discussed in the preceding sections, we make two assumptions about our circuits. First, we assume that the capacity of each connection is a qubit, the quantum analog of a bit, rather than a more complicated quantum system (which would be analogous to transmitting more than one bit, or perhaps an element of some other alphabet, in the Boolean case). Second, we assume that each gate represents a unitary operator; that is, we do not permit more complicated[2] measurements. The second assumption is eliminated in [2].

Under these assumptions, quantum circuits differ from Boolean circuits in the following ways. First, the no-cloning theorem means that at most one consumer can use the output of any one producer, i.e., a producer's output cannot be duplicated to supply multiple consumers. Thus, the provider function π of a quantum circuit is necessarily one-to-one. Furthermore, just as in our earlier discussion of Boolean circuits, we may assume that π is surjective, i.e., that whatever is produced is also consumed; we just treat any unconsumed production as additional

[1] This observation would remain valid if each consumer c received from its provider $\pi(c)$ not a bit but an element of some other, fixed alphabet Σ. But it would not be valid if the alphabet Σ were allowed to be different for different c. For example, if a gate G has hexadecimal inputs and binary outputs, then in order for g_G to be bijective, o_G must have four times as many elements as ι_G.

[2] The general notion of quantum measurement, as defined in, for example [9], allows measurements with only one possible outcome; such a measurement amounts to a unitary operator acting on the state.

output. Thus, we may assume that the provider function π is bijective.

Second, the gate functions g_G in a quantum circuit are not Boolean functions but unitary transformations of Hilbert spaces. Specifically, if ι_G and o_G are, as before, the sets of input and output labels, respectively, of G, then g_G unitarily maps $Q^{\otimes \iota_G}$ to $Q^{\otimes o_G}$, where Q is the one-qubit Hilbert space $Q = \mathbb{C}^2$. Since unitary transformations exist only between Hilbert spaces of equal dimension, we conclude, just as in the balanced Boolean case, that $|\iota_G| = |o_G|$ for every gate G and that therefore also $|I| = |O|$. In these respects, quantum circuits look like balanced Boolean circuits.

Third, and most important, both for the utility of quantum computation and for our work below, is entanglement. In the Boolean case, the inputs to a gate were separate bits, obtained independently from the appropriate providers. In the quantum case, it is usually not the case that a gate's input qubits are independent. They may be entangled with each other and also with other qubits that the gate in question does not directly work with. This entanglement can be seen as the source of the power of quantum computation; it is also the source of some of the complexity in our formal development of the theory.

We now begin the formal development, interspersed with commentary to clarify the underlying intentions.

Notation 8. We use Q to denote the qubit Hilbert space $\mathbb{C}^2$.

Definition 9. A *quantum gate type* is a triple $\langle \iota, o, U \rangle$ consisting of two finite sets ι and o and a unitary transformation $U : Q^{\otimes \iota} \to Q^{\otimes o}$. We call ι the set of input labels, o the set of output labels, and U the operator of the gate type.

As mentioned above, unitarity of U in this definition forces ι and o to have the same cardinality. They need not, however, be the same set, nor need there even be a canonical bijection between them. In many pictures of quantum circuits, a particular bijection would be implicit in the layout of the circuit on the page, but neither the layout nor the bijection is canonical, and neither is relevant in our abstract context.

Definition 10. A *quantum circuit* consists of

- a finite set I of *input nodes*,
- a finite set O of *output nodes*,
- a finite set of *gates*,

- an assignment of a gate type (ι_G, o_G, U_G) to each gate G, and
- a bijective *provider* function π from consumers to producers such that the direct prerequisite relation $\prec$ and therefore also its transitive closure $\prec^*$ are acyclic.

In the last clause of this definition, "consumer", "producer", $\prec$, and $\prec^*$ are to be understood exactly as in the case of Boolean circuits. Thus, the only difference between quantum circuits and balanced Boolean circuits is that each gate G has a unitary operator $U_G : Q^{\otimes \iota_G} \to Q^{\otimes o_G}$ instead of a Boolean bijection $g_G : \{0,1\}^{\iota_G} \to \{0,1\}^{o_G}$. We also carry over from the Boolean case Notation 5 and the terminology "providers of a gate".

The intuition behind the behavior of a quantum circuit is similar in some respects to that for Boolean circuits but quite different in other respects.

As before, a gate G will obtain its input from its providers and act on that input to produce its output. It is, however, important not to misinterpret "retrieve" in our description of the Boolean case, "inputs ... are simply retrieved from the corresponding provider nodes." "Retrieve" must not mean "copy" here because quantum states, unlike classical bits, cannot simply be copied. We should rather regard what is consumed at a gate G to be the same as (not a copy of) what is produced at its provider nodes.

Furthermore, it can be misleading to speak of the state vector on which a gate acts or the state vector that it produces. These states will usually be entangled with other parts of the circuit that are not directly involved with G. Even taking into account that the bits in $\{0,1\}$ of the Boolean situation must be replaced by vectors in Q in the quantum situation, we cannot expect to assign a state vector in Q to each node of the circuit;[3] we cannot expect a direct analog of Theorem 6. Instead of keeping track of separate bits at all the nodes, we must now keep track of the evolution of a global quantum state. Specifically, it makes good sense to speak of the input state where the circuit's computation begins, of the final state after the computation is complete, and of various intermediate states, related to each other by the action of the gates. The following definition serves to describe the contexts in which such a global state makes sense.

Definition 11. A *stage* of a quantum circuit is a set Z of gates closed under direct

[3]We could assign a mixed state to each node by taking a suitable trace of the global state. The trace operation could, however, lose a great deal of information and could, in fact, ruin the usefulness of quantum computation. The reason is that tracing can destroy the entanglement on which quantum computation depends for its power.

prerequisites, i.e., if $x \in Z$ and $y \prec x$ then $y \in Z$. The *exits* of a stage Z are those input nodes in I and output ports of gates G in Z that are not consumed in Z (i.e., are not in $\pi(H)$ for any gate H in Z). We write $\mathrm{Exit}(Z)$ for the set of exits of a stage Z.

The formal notion of stage introduced in this definition is intended to model the informal notion of a stage during a computation, that is, a moment when some gates have already fired and the rest are still waiting to fire. The set Z consists of the gates that have already fired, the "past" of the stage in question; the complementary set of all gates not in Z is the "future" of the stage. The requirement, in the definition, that Z be closed under $\prec$ formalizes the idea that a gate cannot be fired until all its prerequisites have been fired; firing a gate requires the availability of its input. Note that closure under $\prec$ immediately implies closure under $\prec^*$.

The exits of a stage are those producers which have already produced their outputs but have not yet had those outputs consumed. These outputs constitute the information created (or supplied as input) in the past and destined to be consumed in the future.

In terms of typical pictures of circuits, a stage Z can be depicted as a cut through the circuit, separating the gates already fired (those in Z) from the rest of the gates, which still await firing in the future. The circuit's input nodes in I would be depicted as being on the past side of the cut (where Z is) while the output nodes in O are on the future side. The edges in the picture that cross the cut are those whose past ends are in Z (more precisely, these ends are output ports of gates in Z) or I and whose future ends are not. When people use such pictures, they often think in terms of a global state associated to such a cut.

In our abstract picture, we don't directly refer to edges, but our exits correspond to the past ends of the edges crossing the cut (and their pre-images under π correspond to the future ends of those edges).

If one were to actually cut a circuit into a past circuit (input nodes and gates in Z) and a future circuit (output nodes and gates not in Z), then $\mathrm{Exit}(Z)$ would amount to providers for outputs of the past fragment and to inputs for the future fragment. The terminology "exit" is intended to suggest the operation of these nodes in producing output from the Z fragment.

Definition 12. Let Z be a stage of a quantum circuit and let G be a gate not in Z. We say that G is *ready* at Z if all its direct prerequisites (and therefore all its prerequisites) are in Z. In this case, $Z \cup \{G\}$ is also a stage, and we denote it by $Z + G$.

The idea behind this definition is that, after the gates in Z have fired, G is ready to be fired next. Firing it would then bring the computation to the stage $Z + G$. There may, of course, be several gates that are ready at Z, and any one (or more) of them could be fired next.

Notice for future reference that, if a gate G is ready at a stage Z, then

$$\mathrm{Exit}(Z + G) = (\mathrm{Exit}(Z) - \pi(G)) \sqcup \{(G, \mathrm{out}, m) : m \in o_G\},$$

that is, firing G after stage Z adds to the exits the output ports of G and removes the producers that are providers for G. The following proposition is just a reformulation of this observation in a form that will be convenient later.

Proposition 13. *Let Z be a stage, G a gate that is ready at Z, and $R = \mathrm{Exit}(Z) - \pi(G)$. Then*

$$\mathrm{Exit}(Z) = R \sqcup \{\pi(G, \mathrm{in}, l) : l \in \iota_G\}$$

and

$$\mathrm{Exit}(Z + G) = R \sqcup \{(G, \mathrm{out}, m) : m \in o_G\}.$$

We take advantage of this proposition to simplify some of our notation as follows.

Notation 14. Let Z, G, and $R = \mathrm{Exit}(Z) - \pi(G)$ be as in Proposition 13. We identify $\mathrm{Exit}(Z)$ with $R \sqcup \iota_G$ and thereby identify $Q^{\otimes \mathrm{Exit}(Z)}$ with $Q^{\otimes R} \otimes Q^{\otimes \iota_G}$, using the bijection $l \mapsto \pi(G, \mathrm{in}, l)$ for $l \in \iota_G$. Similarly, we identify $Q^{\otimes \mathrm{Exit}(Z+G)}$ with $Q^{\otimes R} \otimes Q^{\otimes o_G}$, using the bijection $m \mapsto (G, \mathrm{out}, m)$ for $m \in o_G$. In other words, we omit mention of those two bijections and the maps they induce on the tensor powers of Q.

It is also worth noting the two extreme cases of stages. The empty set is a stage, the stage at which no gate has yet fired. Only the circuit's input is available at this stage; formally, $\mathrm{Exit}(\varnothing) = I$. The set of all gates is also a stage, the stage after all the gates have fired. Its exits are the providers of the output nodes.

The next theorem formalizes the idea that, once an input state for the circuit is specified in $Q^{\otimes I}$, there is a well-defined global state at each stage, where these global states for different stages are related to each other by the action of the gates. In very abbreviated form, the theorem could be summarized as saying that, given a circuit and an input state, there is a well-defined computation of that circuit on that input. It is the quantum analog of Theorem 6.

Theorem 15. *Let a quantum circuit be given along with an input state vector* $|\psi\rangle \in Q^{\otimes I}$*. Then there is a unique function assigning to each stage* Z *of the circuit a state vector* $C(|\psi\rangle, Z) \in Q^{\otimes \mathrm{Exit}(Z)}$ *subject to the following requirements.*

1. *For the initial stage, we have* $C(|\psi\rangle, \varnothing) = |\psi\rangle$*.*
2. *If* G *is ready at* Z*,then*

$$C(|\psi\rangle, Z + G) = (I_R \otimes U_G)C(|\psi\rangle, Z),$$

where $R = \mathrm{Exit}(Z) - \pi(G)$ *is as in Proposition 13 and* I_R *is the identity operator on* $Q^{\otimes R}$*.*

Requirement (2) in the theorem says, intuitively, that the gate G acts on $C(|\psi\rangle, Z)$ to produce $C(|\psi\rangle, Z + G)$ by applying its operator U_G to the relevant part of this state, which, thanks to the identifications in Notation 14, is $Q^{\otimes \iota_G}$. It does nothing to the rest of $C(|\psi\rangle, Z)$, namely, the part in $Q^{\otimes R}$.

Proof. Fix, for the whole proof, the circuit and the initial state $|\psi\rangle$.

To prove uniqueness of $C(|\psi\rangle, Z)$, we proceed by induction on the cardinality of Z. If this cardinality is 0, then requirement (1) in the theorem ensures uniqueness of $C(|\psi\rangle, \varnothing)$.

Consider now a stage Z that contains at least one gate. Because $\prec^*$ is acyclic, Z must contain a gate G that is not a prerequisite for any other gate in Z. Deletion of G from our stage thus produces another stage, which we call Z' and which, by induction hypothesis, has a uniquely defined $C(|\psi\rangle, Z')$. But then $Z = Z' + G$, and $C(|\psi\rangle, Z) = C(|\psi\rangle, Z' + G)$ is uniquely determined by requirement (2) of the theorem. This completes the induction step and thus completes the uniqueness proof.

It remains to prove the existence part of the theorem. The proof of uniqueness given above implicitly provides a construction that almost proves existence. Specifically, the uniqueness proof obtains $C(|\psi\rangle, Z)$ by removing the gate G to obtain Z' with $Z = Z' + G$ and then acting by $I_R \otimes U_G$ on $C(|\psi\rangle, Z')$. This $C(|\psi\rangle, Z')$ is, of course, obtained in the same way by removing a gate G' to get $Z' = Z'' + G'$, and continuing in the same way until one gets to the empty stage. Starting from $|\psi\rangle = C(|\psi\rangle, \varnothing)$, we apply the gates (or rather their operators $I \otimes U$) in the reverse of the order described above. That is, we have, using the notation R_i for $\mathrm{Exit}(\{G_1, \ldots, G_{i-1}\}) - \pi(G_i)$,

$$C(|\psi\rangle, Z) = (I_{R_k} \otimes U_{G_k}) \circ (I_{R_{k-1}} \otimes U_{G_{k-1}}) \circ \cdots \circ (I_{R_2} \otimes U_{G_2}) \circ (I_{R_1} \otimes U_{G_1})|\psi\rangle$$

for an enumeration $(G_1, G_2, \ldots, G_k)$ of the gates in Z that is *coherent* with $\prec$ in the sense that the prerequisites of any gate appear earlier than the gate itself, i.e., if $G_i \prec G_j$ then $i < j$. (Intuitively, this enumeration is a sequentialization of the circuit, in the sense that G_1 fires first, then G_2, etc.) Note for future reference, that coherence with $\prec$ is the same as coherence with the transitive closure $\prec^*$.

The issue that still needs to be addressed is that there are, in general, several ways choose the gate G in the induction step of the uniqueness proof. We need that all enumerations $(G_1, G_2, \ldots, G_k)$ of the gates in Z coherent with $\prec$ produce the same $C(|\psi\rangle, Z)$.

Once this is done, it will be clear that C so defined satisfies the requirements in the theorem. Indeed, requirement (1) is immediate, being the $k = 0$ case of the definition where no operators act on $|\psi\rangle$. To check requirement (2), it suffices to apply the definition with an arbitrary coherent enumeration for Z and, for $Z + G$, the enumeration obtained by appending G as the last gate.

To show that any two coherent enumerations of Z lead to the same $C(|\psi\rangle, Z)$, we invoke Theorem 32 from [1, §4.2]: If two enumerations of a finite partially ordered set are both coherent with the partial order, then one can be obtained from the other by a sequence of interchanges of two consecutive elements, in such a way that the enumerations at all steps of the process are coherent with the partial order. We apply this result to the finite set Z of gates and the partial order $\prec^*$. We thus find that it suffices to consider two enumerations that differ by interchanging just two consecutive elements.

Suppose, therefore, that one enumeration is $(G_1, \ldots, G_k)$ as above and the other is obtained from it by interchanging G_i and G_{i+1}. These two enumerations give formulas for $C(|\psi\rangle, Z)$ that differ only in two of the factors of the form $I_R \otimes U_G$ that are being composed, and those two factors are adjacent. It is tempting to say that we just need to prove that those factors commute, but the situation is a bit more subtle because each of the two relevant R's depends on the preceding stages.

Let Y be the stage just before either of the two critical gates G_i and G_{i+1} acts, i.e., $Y = \{G_1, \ldots, G_{i-1}\}$. Then $\mathrm{Exit}(Y)$ can be split into three disjoint subsets: the part $\pi(G_i)$ of providers for the input ports of G_i, the analogous part $\pi(G_{i+1})$ for G_{i+1}, and the rest U of $\mathrm{Exit}(Y)$. Formally, we observe that $\pi(G_i)$ and $\pi(G_{i+1})$ are disjoint, because π is bijective, and we define $U = \mathrm{Exit}(Y) - \pi(G_i) - \pi(G_{i+1})$.

Let us consider the action of the critical gates G_i and G_{i+1} with respect to the original enumeration $(G_1, \ldots, G_i, G_{i+1}, \ldots, G_k)$ and, in particular, let us look at the R_i and R_{i+1} at those stages. In our formula for $C(|\psi\rangle, Z)$, we determined R_i by referring to Proposition 13 with the gate G_i and the stage just before the

action of G_i. In our present context, that stage is what we are calling Y, and R_i is therefore $\mathrm{Exit}(Y) - \pi(G_i) = \pi(G_{i+1}) \sqcup U$. Then R_{i+1} is determined by again referring to Proposition 13 but now with the gate G_{i+1} and the stage $Y + G_i$. So R_{i+1} is $\mathrm{Exit}(Y + G_i) - \pi(G_{i+1})$ This R_{i+1} could, a priori, differ from $\mathrm{Exit}(Y) - \pi(G_{i+1}) = \pi(G_i) \sqcup U$, because we now have $\mathrm{Exit}(Y + G_i)$ rather than $\mathrm{Exit}(Y)$. Fortunately, there is no real difference. To see this, first consult Proposition 13 to see that $\mathrm{Exit}(Y + G_i)$ differs from $\mathrm{Exit}(Y)$ only by (1) removal of ports that are in $\pi(G_i)$ and (2) addition of output nodes of G_i. The removals in (1) make no difference for us, because $\pi(G_i)$ is disjoint from $\pi(G_{i+1})$. The additions in (2) also make no difference for the following, less trivial reason. Recall that the enumeration with G_i and G_{i+1} interchanged is also coherent with $\prec$. So we know that $G_i \not\prec G_{i+1}$; no output node of G_i can be the provider for an input node of G_{i+1}. And this is just what we need to ensure that the additions (2) don't matter.

Thus, for the enumeration $(G_1, \ldots, G_i, G_{i+1}, \ldots, G_k)$, we have $R_i = \mathrm{Exit}(Y) - \pi(G_i) = \pi(G_{i+1}) \sqcup U$ and $R_{i+1} = \mathrm{Exit}(Y) - \pi(G_{i+1}) = \pi(G_i) \sqcup U$. An exactly analogous argument gives the same R's for the alternative ordering $(G_1, \ldots, G_{i+1}, G_i, \ldots, G_k)$. That is, the same two operators $I_{R_i} \otimes U_{G_i}$ and $I_{R_{i+1}} \otimes U_{G_{i+1}}$ occur in both versions of the formula for C but in reversed order. So all we still need to prove is that these two operators commute. Fortunately, that is easy. The first is $I_U \otimes U_{G_i} \otimes I_{\pi(G_{i+1})}$ and the second is $I_U \otimes I_{\pi(G_i)} \otimes U_{G_{i+1}}$. These commute because, in each of the three tensor factors, at least one of them is the identity operator. This completes the proof that C is well-defined and thus completes the proof of the theorem. □

The following corollary records information implicit in the proof of Theorem 15, namely that $C(|\psi\rangle, Z)$ is, for each fixed Z, obtained from $|\psi\rangle$ in a uniform and unitary manner.

Corollary 16. *For every quantum circuit and every stage Z, there is a unitary operator $Q^{\otimes I} \to Q^{\otimes \mathrm{Exit}(Z)}$ sending each input state vector $|\psi\rangle$ to the $C(|\psi\rangle, Z)$ described in Theorem 15.*

Proof. The desired unitary operator is the operator

$$(I_{R_k} \otimes U_{G_k}) \circ (I_{R_{k-1}} \otimes U_{G_{k-1}}) \circ \cdots \circ (I_{R_2} \otimes U_{G_2}) \circ (I_{R_1} \otimes U_{G_1}),$$

used in the proof of Theorem 15 and shown there to be independent of the sequentialization of the circuit. □

Given a quantum circuit and an input state $|\psi\rangle \in Q^{\otimes I}$, Theorem 15 provides a complete description of the resulting computation. That includes, in particular, the final result of these computations, namely $C(|\psi\rangle, Z)$ where Z is the set of all the gates in the circuit. But it also includes intermediate results $C(|\psi\rangle, Z)$ for all stages Z, and these tell what happens, step-by-step, in any sequentialization of the computation. Furthermore, even if one does not fully sequentialize the computation but allows several gates to act simultaneously, then, first, any simultaneously acting gates must be incomparable under $\prec^*$ because a gate can act only after its inputs have been produced by its prerequisite gates, and, second, after any part of such a computation has taken place, the gates in that part constitute a stage. Theorem 15 thus provides a well-defined state after that part of the computation.

Remark 17. Theorems 6 and 15 say, in the Boolean and quantum cases respectively, that a circuit and initial data produce a well-defined computation, but the detailed formulations differ significantly. Theorem 6 provides a separate bit $C(x)$ for every node x. We have already explained that it is unreasonable to expect an exact analog for quantum circuits, providing a separate qubit for every node, because the qubits can be entangled. This is why Theorem 15 assigns quantum states not to individual nodes but to (the exits of) whole stages.

In the opposite direction, though, we can easily obtain a Boolean analog of the quantum result. Observe that the definitions of "stage", "exit", "ready", and "$Z + G$" can be applied verbatim to Boolean circuits. Furthermore, Proposition 13 remains correct and we can make identifications like those in Notation 14 for products of sets instead of tensor products of vector spaces. In light of these observations, we can transcribe Theorem 15 to the Boolean situation, obtaining the following corollary of Theorem 6.

Corollary 18. *Let a Boolean circuit be given along with an input $a \in \{0,1\}^I$. Then there is a unique function assigning to each stage Z of the circuit a state $C(a, Z) \in \{0,1\}^{\mathrm{Exit}(Z)}$ subject to the following requirements.*

1. *For the initial stage, we have $C(a, \varnothing) = a$.*
2. *If G is ready at Z, then*

$$C(a, Z + G) = (I_R \times g_G)C(a, Z),$$

where $R = \mathrm{Exit}(Z) - \pi(G)$ is as in Proposition 13 and I_R is the identity function on $\{0,1\}^R$.

Proof. The desired $C(a, Z)$ in this corollary is the function assigning to each $x \in$

Exit(Z) the bit $C(x)$ from Theorem 6. The required properties of C in the present corollary follow easily from the properties of the earlier C in Theorem 6. □

References

[1] Andreas Blass and Yuri Gurevich, "Circuit pedantry," *Bulletin of the European Association for Theoretical Computer Science* 129 October 2019, `https://arxiv.org/abs/1910.06145`

[2] Andreas Blass and Yuri Gurevich, "Quantum ciruits with classical channels," in preparation

[3] Andreas Blass, Yuri Gurevich and Saharon Shelah, "Choiceless polynomial time," *Annals of Pure and Applied Logic* 100 141–187 1999

[4] Andreas Blass, Yuri Gurevich and Saharon Shelah, "On polynomial time computation over unordered structures," *The Journal of Symbolic Logic* 67:3 1093–1125 2002

[5] Andreas Blass, Yuri Gurevich and Jan Van den Bussche, "Abstract state machines and computationally complete query languages," *Information and Computation* 174:1 20–36 2002

[6] C.J. Date and Hugh Darwen, "A Guide to the SQL Standard," 4th edition, Addison-Wesley 1997

[7] Anuj Dawar, David Richerby and Benjamin Rossman, "Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs," *Annals of Pure and Applied Logic* 152:1 31–50 2008

[8] Héctor Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom, "Database systems: The complete book," Prentice Hall 2002

[9] Michael A. Nielsen and Isaac L. Chuang, "Quantum Computation and Quantum Information," 10th Anniversary Edition, Cambridge University Press 2010