

Unconventional Models of Computation through Non-Standard Logic Circuits

Juan C. Agudelo¹ and Walter Carnielli²

¹ Ph.D. Program in Philosophy/Logic

IFCH and Group for Applied and Theoretical Logic- CLE
State University of Campinas - UNICAMP, Brazil.

² IFCH and Group for Applied and Theoretical Logic- CLE
State University of Campinas - UNICAMP, Brazil
SQIG - IT, Portugal.

{juancarlos,carniell}@cle.unicamp.br

Abstract. The classical (boolean) circuit model of computation is generalized via polynomial ring calculus, an algebraic proof method adequate to non-standard logics (namely, to all truth-functional propositional logics and to some non-truth-functional logics). Such generalization allows us to define models of computation based on non-standard logics in a natural way by using ‘hidden variables’ in the constitution of the model. *Paraconsistent circuits* for the paraconsistent logic *mbC* (and for some extensions) are defined as an example of such models. Some potentialities are explored with respect to computability and computational complexity.

1 Introduction

The classical notion of *algorithm* is founded in the model of automated machines introduced by Turing in [18], now known as Turing machines, and in its equivalent theories (λ -definability, partial recursive functions, uniform families of boolean circuits, and so on). Surprising connections between classical logic and classical computation have been established, as the equivalence between the well-known ‘halting problem’ and the undecidability of first-order logic (see [18], [3] and [2]), and the relationship between computational complexity and expressibility in logic (see [15, sec. 2] for a survey). The advent of so many non-classical logics challenge us to think, prompted by the above connections, in the possibilities of logic relativization of the notion of computability and in its conceivable advantages. In this spirit we presented in [1] a model of ‘paraconsistent Turing machines’, a generalization of Turing machines through a paraconsistent logic, and proved that some characteristics of quantum computation can be simulated by this kind of machines. In particular, we have showed how this model can be used to solve the so-called Deutsch-Jozsa problem in an efficient way. In this paper we propose another generalization of a classical model of computation (boolean circuits), defining unconventional models of logic circuits where gate operations are defined in accordance with adequate semantics for non-classical

propositional logics. The conspicuous unconventional peculiarities of our approach are represented by the use of non-truth functional logics to express logic circuitry (instead of a physical or biological approach) and by the significance of such logical standpoint in the foundations of the notion of computability. Some initial inquires about advantages of this logic relativized notion of computability are also addressed here.

A *boolean circuit* is basically a finite collection of input variables and logic gates acyclically connected, where input variables can take values in $\{0, 1\}$ (0 representing the truth value *false* and 1 representing the truth value *true*), and each gate performs a boolean operation (e.g. *AND*, *OR*, *NOT*). Boolean circuits can be viewed as computing *boolean functions* $f: \{0, 1\}^n \rightarrow \{0, 1\}$. A particular computation is performed by establishing the values of the input variables and reading the result of the computation as the output at the final gate (the gate with no output connections to any other gate).³ It is clear that the classical (boolean) circuit model of computation is based on classical propositional logic, considering that gates operate in accordance with functional definitions of the classical logic connectives. The fact that uniform boolean families of circuits are a model for Turing machines clarifies much of the contents of celebrated Cook's theorem in [10]; in an analogous way, the *L*-circuits introduced in Section 3 could shed light in certain aspects of computation related to non-standard logics.

In order to generalize boolean circuits, it is necessary to specify the input-output alphabet (i.e. the set of values allowed for input variables and for outputs of gates) as well as the gate operations. An obvious generalization of boolean circuits to truth-functional many-valued logics would consist in considering the set of truth values as the input-output alphabet, and defining logic gate operations as the functional definitions of the corresponding logic operation. In this way, a logic circuit based in a many-valued logic with truth values set A would compute functions of the form $f: A^n \rightarrow A$.

In the case of infinite-valued logics, despite technical difficulties for implementation of infinitely many distinguishable symbols (for inputs and outputs), the above generalization seems to be adequate for every many-valued logics; however, the specification of gate operations is not obvious for logics without truth-functional semantics.

The original logic relativization of boolean circuits that we propose here is obtained via the *polynomial ring calculus* (*PRC*) introduced in [6], which is an algebraic proof method basically consisting on the translation of logic formulas into polynomials and transforming deductions into polynomial operations. As shown in [6], *PRC* is a mechanizable proof method particularly apt for all finitely-many-valued logics and for several non-truth-functional logics as well, provided that they can be characterized by two-valued *dyadic semantics* (see [5]). The generalization of boolean circuits we are going to propose takes advantage of the features of *PRC*, allowing to define logic gate operations through polynomial operations, and restricting input-output values to finite sets. Interesting

³ This definition can be easily extended to compute functions of the form $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$, allowing more than one final gate.

examples of our generalization are presented by exhibiting *PRC* for the paraconsistent logic *mbC* and for some extensions, while exploring some potentialities of this model with respect to computability and computational complexity.

We present a summary of the *PRC* proof method and some illustrative examples following [6], before presenting the promised generalization of boolean circuits.

2 Polynomial Ring Calculus

PRC consists in translating logic formulas into polynomials over the finite (Galois) fields $GF(p^n)$ (where p is a prime number and n is a natural number).⁴ *PRC* defines rules to operate with polynomials. The first group of rules, the *ring rules*, correspond to the ring properties of addition and multiplication: addition is associative and commutative, there is a ‘zero’ element and all elements have ‘addition inverse’; multiplication is associative, and there is a ‘one’ element and multiplication distributes over addition. The second group of rules, the *polynomial rules*, establishes that the addition of an element x exactly p^n times can be reduced to the constant polynomial 0; and that elements of the form $x^i \cdot x^j$ can be reduced to $x^k \pmod{q(x)}$, for $k \equiv i + j \pmod{p^n - 1}$ and $q(x)$ a convenient primitive polynomial (i.e. and irreducible polynomial of degree n with coefficients in \mathbb{Z}_p). Two inference metarules are also defined, the *uniform substitution*, which allows to substitute some variable in a polynomial by another polynomial (in all occurrences of the variable), and the *Leibniz rule*, which allows to perform substitutions by ‘equivalent’ polynomials.

The tip to define a *PRC*, for an specific logic, is to specify translation of formulas into polynomials in such way that they mimic the conditions of an adequate (correct and complete) class of valuations for the logic in question. We will illustrate this point with some examples.

First, let us consider Classical Propositional Logic (*CPL*). Denoting by *For* the set of well formed formulas of *CPL*, and using Greek letters as metavariables for formulas, a *CPL* valuation is a function $v: For \rightarrow \{0, 1\}$ subject to the following conditions (considering *For* over the alphabet $\{\wedge, \vee, \neg\}$):

$$v(\varphi \wedge \psi) = 1 \text{ iff } v(\varphi) = 1 \text{ and } v(\psi) = 1; \quad (1)$$

$$v(\varphi \vee \psi) = 1 \text{ iff } v(\varphi) = 1 \text{ or } v(\psi) = 1; \quad (2)$$

$$v(\neg\varphi) = 1 \text{ iff } v(\varphi) = 0. \quad (3)$$

In this case, *For* could be translated to polynomials in the polynomial ring $\mathbb{Z}_2[X]$ (i.e. polynomials with coefficients in the field \mathbb{Z}_2 and variables in the set

⁴ The number p is usually called the *characteristic* of the field. The characteristic of any finite field is necessarily a prime number, see for example [16].

$X = \{x_1, x_2, \dots\}$). The translation function $*$: $For \rightarrow \mathbb{Z}_2[X]$ is defined by:

$$p_i^* = x_i \text{ if } p_i \text{ is a propositional variable;} \quad (4)$$

$$(\varphi \wedge \psi)^* = \varphi^* \cdot \psi^*; \quad (5)$$

$$(\varphi \vee \psi)^* = \varphi^* \cdot \psi^* + \varphi^* + \psi^*; \quad (6)$$

$$(\neg\varphi)^* = \varphi^* + 1. \quad (7)$$

Polynomial rules in this case establish that $x + x$ can be reduced to 0 and that $x \cdot x$ can be reduced to x . Assigning values in $\{0, 1\}$ to variables in X , it can be easily shown that:

$$(\varphi \wedge \psi)^* = 1 \text{ iff } \varphi^* = 1 \text{ and } \psi^* = 1; \quad (8)$$

$$(\varphi \vee \psi)^* = 1 \text{ iff } \varphi^* = 1 \text{ or } \psi^* = 1; \quad (9)$$

$$(\neg\varphi)^* = 1 \text{ iff } \varphi^* = 0. \quad (10)$$

which means that the translation $*$ characterizes all *CPL*-valuations. Consequently, using $\varphi(p_1, \dots, p_k)$ to express that φ is a formula with propositional variables in $\{p_1, \dots, p_k\}$, we have the following theorem:

Theorem 1. $\vdash_{CPL} \varphi(p_1, \dots, p_k)$ iff $\varphi^*(a_1, \dots, a_k) = 1$ for every $(a_1, \dots, a_k) \in \mathbb{Z}_2^k$.

And by theorem 2.3 in [6], we have:

Theorem 2. $\vdash_{CPL} \varphi$ iff φ^* reduces by *PRC* rules to the constant polynomial 1.

As an example of a proof of $\varphi \vee \neg\varphi$ in *CPL* using *PRC* (taking into account Theorem 2):

$$\begin{aligned} (\varphi \vee \neg\varphi)^* &= \varphi^* \cdot (\neg\varphi)^* + \varphi^* + (\neg\varphi)^* \\ &= \varphi^* \cdot (\varphi^* + 1) + \varphi^* + \varphi^* + 1 \\ &= \varphi^* \cdot \varphi^* + \varphi^* + \varphi^* + \varphi^* + 1 \\ &= \varphi^* + \varphi^* + \varphi^* + \varphi^* + 1 \\ &= 1 \end{aligned}$$

Now, let us consider the paraconsistent logic *mbC*, a fundamental logic in the hierarchy of Logics of Formal Inconsistency (*LFI*s). Albeit not a finite valued logic, *mbC* can be characterized by a non-truth-functional two-valued valuation semantics (cf. [7], Sections 3.2 and 3.3). In this case, valuations are subject to the following conditions (considering *For* over the alphabet $\{\wedge, \vee, \rightarrow, \neg, \circ\}$, where \circ denotes the ‘consistency’ operator):

$$v(\varphi \wedge \psi) = 1 \text{ iff } v(\varphi) = 1 \text{ and } v(\psi) = 1; \quad (11)$$

$$v(\varphi \vee \psi) = 1 \text{ iff } v(\varphi) = 1 \text{ or } v(\psi) = 1; \quad (12)$$

$$v(\varphi \rightarrow \psi) = 1 \text{ iff } v(\varphi) = 0 \text{ or } v(\psi) = 1; \quad (13)$$

$$v(\neg\varphi) = 0 \text{ implies } v(\varphi) = 1; \quad (14)$$

$$v(\circ\varphi) = 1 \text{ implies } v(\varphi) = 0 \text{ or } v(\neg\varphi) = 0. \quad (15)$$

Note that implications in conditions (14) and (15) hold in one direction only, the other direction being ‘indeterminate’. The way offered in [6] to mimic such non-determination via polynomials is the introduction of new variables (outside the set of variables for propositional variables) in the translations from formulas into polynomials. In this paper, we will dub such new variables ‘hidden variables’.⁵ The simple but powerful strategy of introducing hidden variables is the key idea for making *PRC* apt for a wide class of non-classical logics, including non-truth-functional and infinite-valued logics. Using this strategy, formulas in *mbC* can be translated into polynomials in the polynomial ring $\mathbb{Z}_2[X]$, defining the translation function $*$: $For \rightarrow \mathbb{Z}_2[X]$ by:

$$p_i^* = x_i \text{ if } p_i \text{ is a propositional variable;} \quad (16)$$

$$(\varphi \wedge \psi)^* = \varphi^* \cdot \psi^*; \quad (17)$$

$$(\varphi \vee \psi)^* = \varphi^* \cdot \psi^* + \varphi^* + \psi^*; \quad (18)$$

$$(\varphi \rightarrow \psi)^* = \varphi^* \cdot \psi^* + \varphi^* + 1; \quad (19)$$

$$(\neg\varphi)^* = \varphi^* \cdot x_\varphi + 1; \quad (20)$$

$$(\circ\varphi)^* = (\varphi^* \cdot (x_\varphi + 1) + 1) \cdot x_{\varphi'}; \quad (21)$$

where x_φ and $x_{\varphi'}$ are hidden variables.⁶

For paraconsistent logics extending *mbC* where the formula $\circ\varphi$ is equivalent to the formula $\neg(\varphi \wedge \neg\varphi)$ (such as C_1 , *mCil*, *Cil*, *Cile*, *Cila*, *Cilae* and *Cilo*, see [7]; here we will refer to these logics as *mbC*⁺ logics), the translations of $\circ\varphi$ and $\neg(\varphi \wedge \neg\varphi)$ must be equivalent and dependent of the same variables. For *mbC*, the translation of $\circ\varphi$ is $(\circ\varphi)^* = (\varphi^* \cdot (x_\varphi + 1) + 1) \cdot x_{\varphi'}$, and the translation of $\neg(\varphi \wedge \neg\varphi)$ is $(\neg(\varphi \wedge \neg\varphi))^* = \varphi^* \cdot (\varphi^* \cdot x_\varphi + 1) \cdot x_{\varphi \wedge \neg\varphi} + 1$. The only variables that do not match in these translations are $x_{\varphi'}$ and $x_{\varphi \wedge \neg\varphi}$. Then, for *mbC*⁺ logics, setting these variables as equal and taking into account that both translations must be equivalent, we have that $x_{\varphi'} = x_{\varphi \wedge \neg\varphi} = 1$, and as consequence $(\circ\varphi)^* = \varphi^* \cdot (x_\varphi + 1) + 1$.

Some valuation semantics introduce conditions over schemes of formulas with more than one logic operator. For instance, paraconsistent logics with the axiom $\neg\neg\varphi \rightarrow \varphi$ must have the following clause in its valuation semantic (cf. [7, p. 58]):

$$v(\neg\neg\varphi) = 1 \text{ implies } v(\varphi) = 1. \quad (22)$$

This kind of condition translate in ‘polynomial conditions’, i.e. relations between polynomials that must be considered in any polynomial reduction. For *mbC*, using the *PRC* presented above, the translation of the formula $\neg\neg\varphi$ is $(\neg\neg\varphi)^* = (\varphi^* \cdot x_\varphi + 1) \cdot x_{\neg\varphi} + 1$. For a logic extending *mbC*, with a valuation semantic

⁵ In [6] such variables are called ‘quantum variables’, by resemblance with the ‘hidden variables’ theories of quantum mechanics.

⁶ In the translation defined to *mbC* in [6] a different rule to translate $\circ\varphi$ is presented, but such translation does not permit that $(\circ\varphi)^*$ and φ^* take simultaneously the value 0, while the semantic valuation for *mbC* permits $v(\circ\varphi) = v(\varphi) = 0$. Our definition fix this problem.

including (22), the following polynomial condition must be taken into account:

$$(\varphi^* \cdot x_\varphi + 1) \cdot x_{\neg\varphi} = 0 \text{ implies } \varphi^* = 1. \quad (23)$$

3 Generalizing Boolean Circuits via Polynomial Ring Calculus

Having presented the *PRC*, it is now easy to generalize boolean circuits to a wide extent of non-classical logics:

Definition 1 (*L*-circuit). *Let L be a propositional logic provided with a PRC over the finite field F . An L -circuit is a finite directed acyclic graph $C = (V, E)$, where V is the set of nodes and E the set of edges. Nodes without incoming edges are called inputs of the circuit, and are denoted by variables (x_1, x_2, \dots) or by constants (elements of F). Other nodes are called logic gates, and correspond to logic operators of L . A logic gate evaluates the polynomial associated to the corresponding logic operator and perform polynomial reductions in accordance with the *PRC* for L . The gate without output connections to any other gate gives the output of the circuit.*

It is to be noted that any logic gate has at most one outgoing edge, and that the number of incoming edges of a logic gate is determined by the arity of the corresponding logic operator.

With this definition of *L*-circuit and the *PRC* presented to *CPL* in the previous section, it is easy to see that *CPL*-circuits behave just in the same way as boolean circuits, as it would have to be expected. More interesting cases of *L*-circuits are obtained when we consider the paraconsistent logic *mbC* and its extensions, and their respective *PRC*s. For instance, the *mbC*-circuit for the formula $\neg p_1 \wedge \neg p_2$ (graphically represented by the Figure 1) shows how ‘hidden variables’ appear in the process of computation, giving place to ‘indeterminism’, an interesting characteristic not present in boolean circuits.⁷

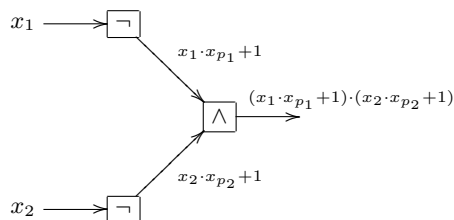


Fig. 1. *mbC*-circuit for the formula $\neg p_1 \wedge \neg p_2$

⁷ Probabilistic circuits are defined by introducing probabilistic input variables, but such variables are independent of the logic operations (see [9, Sect. 2.12]). In other words, indeterminism in probabilistic circuits is introduced by allowing indeterminism in the inputs, but not generated by logic operations as in our *L*-circuits.

This *mbC*-circuit, when variables x_1 and x_2 take both the value 0, produces the output 1 in a deterministic way. But when x_1 takes the value 1, the output depends on the hidden variable x_{p_1} ; and when x_2 takes the value 1, the output depends on the hidden variable x_{p_2} . Hidden variables, like input variables, have to take values in F (the field used in the *PRC*, in this case \mathbb{Z}_2), and we could assume that values to hidden variables are randomly assigned. Then, we could distinguish between *deterministic* and *non-deterministic L*-circuits:

Definition 2 (Deterministic and non-deterministic L-circuit). An *L*-circuit is deterministic if the reduced polynomial of the output gate does not contain hidden variables, otherwise the *L*-circuit is non-deterministic.

Note that there may be deterministic *L*-circuits with hidden variables within the circuit. For example, the *mbC*-circuit for the formula $p_1 \vee \neg p_1$, graphically represented by the following figure:

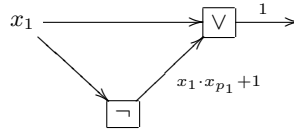


Fig. 2. *mbC*-circuit for the formula $p_1 \vee \neg p_1$

It is also important to take into account that non-deterministic *L*-circuits could behave deterministically for some specific inputs, like the *mbC*-circuit for the formula $\neg p_1 \wedge \neg p_2$ presented above.

Now, having defined the concept of *L*-circuit, it is opportune to discuss its potentialities. Main characteristics of any model of computation concerns its computability power (the class of functions the model of computation can express) and its computational complexity. Classically, it is possible to show that to every boolean function whose inputs have a fixed length, there is a boolean circuit that computes such function (see for example [17, p. 79, prop. 4.3]). Non-computable functions arise when we consider functions whose inputs have arbitrary length. In this case it is necessary to define a *family of circuits*, which is an infinite denumerable set of circuits $\{C_i\}_{i \in \mathbb{N}}$ where the circuit C_i computes inputs of length i . To accept a family of circuits as an effective procedure we have to impose the restriction that such a family of circuits should be *uniformly generated*, i.e. described by a Turing machine. Usually, uniform generation also imposes computational complexity restrictions; it establishes, for example, that the circuit must be generated by a polynomial time Turing machine, i.e. the time taken to generate the circuit C_i must be a polynomial expression on i . A uniformly generated family of circuits is called a *uniform family* of circuits. It can be proven that uniform families of boolean circuits are equivalent (with respect to computability and computational complexity) to Turing machines (see [4]).

The next two sections explore potentialities of uniform families of L -circuits as models of computation, first with respect to computability and then with respect to computational complexity, considering only to the particular case of paraconsistent logics.

3.1 Potentialities with Respect to Computability

Because paraconsistent logics are deductively weaker than CPL (or a version of CPL in a different signature, see [7, Sect. 3.6]), we could think that the class of functions computed by uniform families of paraconsistent L -circuits would be narrower than functions computed by uniform families of boolean circuits, but we will show that this is not the case for mbC .

Theorem 3. *To every function computed by a uniform family of boolean circuits there is a uniform family of mbC -circuits that computes such function.*

Proof. Let the mapping $t_1: For \rightarrow For^\circ$ (where For is the set or formulas of CPL and For° is the set or formulas of mbC) defined as:

1. $t_1(p) = p$, for any propositional variable p ;
2. $t_1(\varphi \# \psi) = t_1(\varphi) \# t_1(\psi)$, if $\# \in \{\wedge, \vee, \rightarrow\}$;
3. $t_1(\neg\varphi) = \sim t_1(\varphi)$, where \sim is defined in mbC as $\sim\varphi = \varphi \rightarrow (\psi \wedge (\neg\psi \wedge \circ\psi))$.

t_1 conservatively translates CPL within mbC (cf. [7, p. 47]), that is, for every $\Gamma \cup \{\varphi\} \subseteq For$:

$$\Gamma \vdash_{CPL} \varphi \text{ iff } t_1(\Gamma) \vdash_{mbC} t_1(\varphi),$$

where $t_1(\Gamma) = \{t_1(\psi) : \psi \in \Gamma\}$.

Using t_1 , the uniform family of boolean circuits could be algorithmically translated to an equivalent uniform family of mbC -circuits, just by changing the *NOT* gates for subcircuits corresponding to the formula $\varphi \rightarrow (\psi \wedge (\neg\psi \wedge \circ\psi))$. It can be checked that $(\varphi \rightarrow (\psi \wedge (\neg\psi \wedge \circ\psi)))^* = \varphi^* + 1$, which is the same polynomial as in classical negation. \square

The previous theorem can be generalized to several other paraconsistent logics, that is, to those where a conservatively translation function from CPL can be defined taking into account that such translation function must be effectively calculated. In the other direction, the existence of uniform families of boolean circuits to every uniform family of L -circuits (for any logic L provided with PRC) is guaranteed by the classical computability of roots for polynomials over finite fields. Then, the L -circuits model does not invalidate Church-Turing's thesis.

Generalizing PRC to infinite fields could be a way to define models of 'hyper-computation' (i.e. models that permit the computation of non Turing-machine computable functions, see [11]). For the case where the field is the rational numbers with usual addition and multiplication, the output of a certain subclass of L -circuits would be Diophantine polynomials, and thus the problem of determining when L -circuits have 0 as output will be equivalent to the problem

of determining when Diophantine equations have solutions, a well-known classically unsolvable problem. However, in spite of the possibility of finding a clever way (using hidden variables and polynomial operations) to determine 0 outputs of L -circuits, hypercomputation would be attained at the cost of introducing an infinite element into the model, which seems to be opposed to the intuitive idea of ‘computable’ (but most models of hypercomputation, nevertheless, are endowed with some kind of infinite element—see [12] and [13] for critiques of hypercomputation).

3.2 Potentialities with Respect to Computational Complexity

Computational complexity potentialities can be thought in both directions too, that is: are there uniform families of boolean circuits more efficient than uniform families of paraconsistent L -circuits? And vice versa? The first question has an immediate answer, but the other is not obvious. We will next show the answer to the first question, and subsequently some ideas related to the second.

Theorem 4. *To every function computed by a polynomial size uniform family of boolean circuits there is a polynomial size uniform family of mbC-circuits that computes such function.*

Proof. The translation t_1 , defined in the proof of Theorem 3 adds only a polynomial number of gates to the circuit (4 gates for every *NOT* gate). \square

As in the case of Theorem 3, Theorem 4 can be generalized to several other paraconsistent logics, those to where a conservative translation function from *CPL* can be defined (taking into account that such translation function must be effectively calculated and add at most a linear number of gates to the circuits).

The other direction is left as an open problem, but we will show some analogies between *quantum circuits* (see [8]) and paraconsistent circuits that could be valuable in looking for a positive answer.

The quantum circuit model of computation is a generalization of the boolean circuits that depart from a quantum mechanics perspective. In a brief description of such model of computation, we can say that classical *bits* are substituted by *qubits* (or *quantum bits*). Qubits, in a similar way than classical bits, can take values $|0\rangle$ and $|1\rangle$ (where $|0\rangle$ and $|1\rangle$ represent bases vectors in a two dimensional Hilbert space), but the radical difference is that qubits admit also *superpositions* (or linear combinations) of such values, following the rules of the description of a quantum system in quantum mechanics. Superposition of values are usually referred to as *superposition states*. In some interpretations of quantum mechanics (as in the many-worlds interpretations), the states involved in a superposition state are interpreted as coexisting, and then we can think that qubits can take simultaneously the values $|0\rangle$ and $|1\rangle$, with probabilities associated to each value. In quantum circuits, boolean gates are substituted by *quantum gates*, which are unitary operators over qubits, also in agreement with quantum mechanics. A relevant quantum gate is the so-called *Hadamard*

gate, which transforms base states into perfect superposition states (states where probabilities are the same to any base state). Because quantum gates are also linear operators, the application of a gate to a superposition state is equivalent to the simultaneous application of the gate to every base state in the superposition. This characteristic allows the circuit to compute in parallel, in what is called *quantum parallelism*. The problem is that when a measurement is performed, only one of the superposition states is obtained with a probability given by the coefficients of the linear combination. Then, quantum algorithms must take advantage (before performing the measurement) of global characteristics of the functions computed in parallel.

We propose that outputs depending on hidden variables in L -circuits correspond to superposition states, since the indeterminism introduced by hidden variables allows us to simulate indeterminism of superposition states measurements. Under this assumption, gates corresponding to \neg and \circ in mbC and mbC^+ logics, setting 1 as input, could be used to simulate the Hadamard gate (because in mbC , $(\neg\varphi)^*(1) = x_\varphi + 1$ and $(\circ\varphi)^*(1) = x_\varphi \cdot x_{\varphi'}$, and in mbC^+ logics we have that $(\circ\varphi)^*(1) = x_\varphi$). To illustrate how \neg and \circ gates could be used in mbC -circuits and mbC^+ -circuits, we will demonstrate a relevant theorem:

Theorem 5. *Let φ a formula in mbC and $\varphi[p_i/\psi]$ the formula obtained from φ by uniformly replacing p_i by ψ . Then, $\varphi^* = f(x_1, \dots, x_i, \dots, x_n)$ implies that $(\varphi[p_i/\neg p_i])^*(x_1, \dots, 1, \dots, x_n) = f(x_1, \dots, x_{p_i} + 1, \dots, x_n)$.*

Proof. By induction on the complexity of φ . □

Theorem 5 shows that, for mbC , by uniformly replacing in a formula φ the propositional variable p_i with the formula $\neg p_i$, and setting 1 to the input variable x_i in the corresponding mbC -circuit, we have that the output polynomial of the mbC -circuit is equal to the polynomial for φ replacing the input variable x_i with $x_{p_i} + 1$.

A similar theorem can be established for mbC^+ logics, uniformly replacing the propositional variable p_i with the formula $\circ p_i$. In this case, the output polynomial is equal to the polynomial for φ replacing the input variable x_i with the hidden variable x_{p_i} . This result could be used to compute satisfiability of CPL formulas in a non-deterministic way, taking the translation of a CPL formula to a mbC^+ logic (with a conservative translation), by uniformly replacing all propositional variables p_i with $\circ p_i$ and setting 1 to all input variables.

For example, the translation of the CPL formula $\varphi = p_1 \wedge p_2$ to a mbC^+ formula (using the translation t_1 above) is the same formula $\varphi = p_1 \wedge p_2$. Using the PRC for mbC , and the translation of $\circ\varphi$ to polynomials in mbC^+ logics already present above, we have that $\varphi^* = x_1 \cdot x_2$ and $(\varphi[p_1/\circ p_1, p_2/\circ p_2])^*(1, 1) = x_{p_1} \cdot x_{p_2}$. Note that the only thing we did was to replace input variables with hidden variables.

Obviously, for any CPL formula φ with propositional variables in $\{p_1, \dots, p_n\}$, if φ is satisfiable, then $(t_1(\varphi)[p_1/\circ p_1, \dots, p_n/\circ p_n])^*(1, \dots, 1) = 1$ for some assignment of values to hidden variables. An interesting question is whether there exists any method, in accordance with PRC , that allows to eliminate hidden

variables in such a way that polynomials distinct to the constant polynomial 0 reduces to the constant polynomial 1. For some paraconsistent logics that we have explored, this seems to be impossible, but we also do not have a definite negative result (the possibility of polynomial conditions, see last paragraph of Section 2, could be the key to obtain positive or negative results). We also have to take into account that *PRCs* can be defined over distinct fields, and for other (not necessarily paraconsistent) non-standard logics.

4 Final Comments

This paper proposes a natural way to generalize boolean circuit model of computation to models based on non-standard logics. This opens possibilities to integrate innovative concepts of non-standard logics into computational theory, shedding light into the question of how depending on logic abstract notions of computability and algorithmic complexity could be. Moreover, the generalization proposed here has the advantage of being flexible enough to be almost immediately adapted to a wide range of non-standard logics.

Cook's theorem (cf. [10]) states that any NP-problem can be converted to the satisfiability problem in *CPL* in polynomial time. The proof shows, in a constructive way, how to translate a Turing machine into a set of *CPL* formulas in such a way that the machine outputs '1' if, and only if, the formulas are consistent. As mentioned in the introduction, a model of paraconsistent Turing machines presented in [1] was proved to solve Deutsch-Jozsa problem in an efficient way. We conjecture that a similar result as Cook's theorem can be proven to paraconsistent Turing machines. In this way, paraconsistent circuits could be shown to efficiently solve Deutsch-Jozsa problem. Consequences of this approach would be the definition of 'non-standard' complexity classes relative to such unconventional models of computation founded over non-classical logics.

Finally, another relevant question that we do not tackle in this paper refers to physical viability: are there any physical implementations to the hidden-variable model of computation presented here? We think that hidden-variable theories of quantum mechanics (see [14]) could give a positive response to this question.

Acknowledgements

This research was supported by FAPESP- Fundação de Amparo à Pesquisa do Estado de São Paulo, Brazil, Thematic Research Project grant 2004/14107-2. The first author was also supported by a FAPESP scholarship grant 05/05123-3, and the second by a CNPq (Brazil) Research Grant 300702/2005-1 and by an EU-FEDER grant via CLC (Portugal).

References

1. Juan C. Agudelo and Walter Carnielli. Quantum algorithms, paraconsistent computation and Deutsch's problem. In Bhanu Prasad, editor, *Proceedings of the 2nd*

- Indian International Conference on Artificial Intelligence*, pages 1609–1628, Pune, India, 2005.
2. George Boolos and Richard Jeffrey. *Computability and logic*. Cambridge University Press, 3rd. edition, 1989.
 3. J. Richard Büchi. Turing-machines and the Entscheidungsproblem. *Mathematische Annalen*, 148:201–213, 1962.
 4. Chris Calabro. Turing Machine vs. RAM Machine vs. Circuits. Lecture notes, available at <http://http://www-cse.ucsd.edu/classes/fa06/cse200/ln2.ps>.
 5. Marcelo E. Coniglio Carlos Caleiro, Walter Carnielli and João Marcos. Two’s company: “the humbug of many logical values”. In Jean-Yves Beziau, editor, *Logica Universalis*, pages 169–189. Birkhäuser Verlag, Basel, Switzerland. Preprint available at <http://ws1c.math.ist.utl.pt/ftp/pub/CaleiroC/05-CCCM-dyadic.pdf>.
 6. Walter A. Carnielli. Polynomial ring calculus for many-valued logics. In B. Werner, editor, *Proceedings of the 35th International Symposium on Multiple-Valued Logic*, pages 20–25. IEEE Computer Society, 2005. Preprint available at *CLE e-Prints* vol 5, n. 3, 2005: http://www.cle.unicamp.br/e-prints/vol_5,n_3,2005.html.
 7. Walter A. Carnielli, Marcelo E. Coniglio, and João Marcos. Logics of Formal Inconsistency. In D. Gabbay and F. Guentner, editors, *Handbook of Philosophical Logic*, volume 14. Kluwer Academic Publishers, 2nd edition, 2005. In print. Preprint available at *CLE e-Prints* vol 5, n. 1, 2005: http://www.cle.unicamp.br/e-prints/vol_5,n_1,2005.html.
 8. Isaac L. Chuang and Michael A. Nielsen. *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2000.
 9. Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Springer-Verlag, 2002.
 10. Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.
 11. Jack Copeland. Hypercomputation. *Minds and machines*, 12:461–502, 2002.
 12. Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: Life and Legacy of a Great Thinker*, pages 195–212. Springer, Berlin, 2004.
 13. Martin Davis. Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178:4–7, 2004.
 14. Marco Genovese. Research on hidden variable theories: A review of recent progresses. *Physics Reports*, 413:319–396, 2005.
 15. Josef Y. Halpern, Robert Harper, Neil Immerman, Phokion G. Kolaitis, Moshe Y. Vardi, and Victor Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7(2):213–236, 2001.
 16. Nathan Jacobson. *Basic Algebra I*. New York: W. H. Freeman and Company, 2nd edition, 1985.
 17. Christos Papadimitriou. *Computational Complexity*. Adisson-Wesley, 1994.
 18. Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, pages 230–265, 1936. A correction, *ibid*, vol 43. 1936-1937. págs. 544 - 546.