

A Theory of Cartesian Arrays

(with Applications in Quantum Circuit Verification)

Yu-Fang Chen¹[0000–0003–2872–0336], Philipp Rümmer^{2,3}[0000–0002–2733–7098]()
and Wei-Lun Tsai¹

¹ IIS, Academia Sinica, Taiwan

² University of Regensburg, Germany

³ Uppsala University, Sweden

Abstract. We present a theory of Cartesian arrays, which are multi-dimensional arrays with support for the projection of arrays to sub-arrays, as well as for updating sub-arrays. The resulting logic is an extension of Combinatorial Array Logic (CAL) and is motivated by the analysis of quantum circuits: using projection, we can succinctly encode the semantics of quantum gates as quantifier-free formulas and verify the end-to-end correctness of quantum circuits. Since the logic is expressive enough to represent quantum circuits succinctly, it necessarily has a high complexity; as we show, it suffices to encode the k -color problem of a graph under a succinct circuit representation, an NEXPTIME-complete problem. We present an NEXPTIME decision procedure for the logic and report on preliminary experiments with the analysis of quantum circuits using this decision procedure.

1 Introduction

There has been extensive research on logics to reason about array data-types in programs. Arrays can concisely represent the values of an unbounded number of memory locations, and have been successfully applied to verify industrial-scale programs [11,29,15]. An array formula encoding the semantics of a program path is typically linear in the number of program statements. Much of the existing work focuses on one-dimensional arrays and uses nesting to handle the case of multiple dimensions.

This paper studies a logic called *Cartesian Array Logic (CaAL)*, in which multi-dimensional arrays are treated as first-class citizens. The motivation for designing this logic comes from developing a tailor-made theory for reasoning about *quantum circuits* or *programs*, which need a fundamentally different representation of states than classical programs. *Quantum states* exist in a *superposition* of classical states. Figure 1 gives an example of a 5-qubit quantum state, which can be interpreted as a probability distribution over 2^5 classical states; every classical state, which can be seen as a string of n bits, is associated with a probability of being observed.

Current SMT-based solutions for reasoning about quantum programs [3] encode program paths to a Satisfiability Modulo Theories (SMT) formula over the theory of real numbers. For a n -qubit quantum program, the direct encoding uses 2^n variables

$ 00000\rangle$	69%
$ 00001\rangle$	1%
...	
$ 11111\rangle$	1%

Fig. 1: A quantum state.

to represent the execution of a quantum circuit, one variable per classical state. The formula representing a quantum circuit is exponential in the circuit size.

In the Cartesian Array Logic designed in this paper, one can instead encode an n -qubit quantum state as an array $s : (\mathbb{B}^n \Rightarrow \mathbb{C})$ that maps each classical state to a complex number c encoding the probability of this classical state being observed. The squared absolute value $|c|^2$ is the probability that the complex number c encodes. *Quantum gates*, the basic operating units of a *quantum circuit*, can be viewed as functions that transform one quantum state to another. We show that CaAL can concisely encode the semantics of quantum gates, so that a path formula becomes linear in the circuit size. The semantics of a quantum circuit is the composition of the gate encodings.

Structure of the Paper. The syntax and formal semantics of the CaAL logic will be given in Section 2. In the same section, we show that this logic is quite expressive, it can easily encode the satisfiability problem of a quantified Boolean formula (QBF). We show that deciding the logic is, in fact, NEXPTIME-hard by a polynomial reduction from the k -color problem of a succinct circuit representation of graphs [23]. As an application, in Section 3, we show that the logic can concisely encode the semantics of *quantum circuits*, using \mathbb{B}^n as the index type and \mathbb{C} as the value type. In Section 4, we present a decision procedure for CaAL, extending the classical approach of read-over-write propagation used for arrays. In the worst case, our procedure might perform an exponential number of such propagations; hence, if the underlying logic can be decided in NP, our logic can be decided in NEXPTIME. The preliminary experimental results of applying this decision for quantum circuit verification can be found in Section 5.

Contributions of the paper are (i) a new array logic, CaAL, with native support for multi-dimensional arrays; (ii) the proof the satisfiability problem of CaAL is NEXPTIME-hard; (iii) a linear encoding of the semantics of quantum circuits in CaAL; (iv) an NEXPTIME decision procedure for CaAL without nested array sorts; and (v) a preliminary evaluation of our approach using standard quantum circuits.

Related Work on Verification of Quantum Circuits. Although quantum states can be naturally represented as arrays, the connection between array theories and quantum circuit verification is novel, to the best of our knowledge. In the past, people have considered automated quantum circuit verification based on automata [7], various types of equivalence checking [1,9,19,33], abstract interpretation [34,24], and model checking [13,21,32]. However, techniques based on satisfiability modulo theories (SMT) are still lacking. The closest work to ours is a symbolic execution and verification framework of quantum circuits [3]. The work encodes quantum circuit verification problems into SMT with the theory of real numbers, using variables in trigonometric functions, e.g., $\sin x$, which might lose precision in corner cases. As mentioned, their approach requires 2^n variables to encode a n -qubit circuit in the worst case. As far as we know, our work is the first SMT-based approach that allows a precise and succinct encoding and verification of quantum circuits.

Related Work on Array Theories. There is a large body of research on array decision procedures for SMT, going back to the 1980s, and most SMT solvers implement at least

the theory of extensional arrays (with operations *read* and *write/store*) in our paper, as standardized in SMT-LIB [2]. Stump et al. [29] presented a decision procedure for this theory and formed the basis for many later procedures. An extension of the theory, called Combinatorial Array Logic (CAL), with functions for *constant arrays* and for the *point-wise extension of functions* was presented by De Moura et al. [11]. CAL served as the main inspiration for our work and is in this paper extended further by adding *projections* and *updates of sub-arrays*. An extension of CAL with *cardinality constraints* was presented by Raya et al. [25]. Christ et al. [8] present an algorithm for the theory of arrays where lemmas are created lazily based on weak equivalences; this method was later extended to handle *constant arrays* [20].

There are also many more generalized decision procedures for arrays. For instance, Ganesh et al. [16] focus on the combined theory of arrays and bit-vectors and present a decision procedure based on pre-processing, bit-blasting, and linear arithmetic solving. Brummayer et al. present a decision procedure for the same theory that introduces lemmas lazily, guided by congruence closure [6]. An extended array theory tailored to software, including operations `memset` and `memcpy`, was presented by Falke et al. [12]. More recently, several theories of finite arrays were proposed. Bonacina et al. [5] extend the standard theory of arrays with an abstract notion of length, and present a decision procedure based on the CDSAT framework. Wang et al. [31] consider a logic extending CAL with a length function, as well as operations for concatenation, slicing, and repetition of arrays, and identify a decidable fragment. Sheng et al. [27] propose a theory of sequences that combines the standard array operations with a length function, concatenation, and slicing. All those logics cannot directly encode quantum circuits in a similar style as CaAL, however, since no projection operation is available.

2 A Theory of Cartesian Arrays

2.1 Preliminaries

We work in the setting of multi-sorted first-order logic with equality; see, e.g., [18]. A signature is a tuple $\Sigma = (\Sigma^S, \Sigma^F, \Sigma^P)$ consisting of a set Σ^S of sorts, a set Σ^F of function symbols, and a set Σ^P of predicates. Predicates and functions have fixed arity and argument sorts, and functions have a fixed result sort. Given a signature Σ and a set \mathcal{X} of sorted variables, we define the usual notions of Σ -terms, Σ -atoms, Σ -literals, Σ -formulas, and Σ -sentences. Formulas are evaluated over Σ -structures $M = (D, I)$ that interpret every sort $\sigma \in \Sigma^S$ as a non-empty domain $I(\sigma) \subseteq D$, predicates $p \in \Sigma^P$ as relations $I(p)$, and functions $f \in \Sigma^F$ as set-theoretical functions $I(f)$. We slightly abuse notation; we assume that also variables $x \in \mathcal{X}$ are mapped to values $I(x)$ by M . The evaluation of terms, formulas, etc., is defined as is common; the equality symbol $=$ is assumed to be interpreted as the equality relation on D . A theory T over Σ is a set of Σ -sentences. A Σ -formula ϕ is called T -satisfiable if there is a Σ -structure M satisfying both the T -axioms and ϕ .

2.2 Definition of the Theory of Cartesian Arrays

Cartesian arrays are introduced in the context of a base signature Σ_B and a base Σ_B -theory T_B , which provides the index and value sorts for arrays. The signature $\Sigma_{\text{CaAL}} =$

Table 1: Operations included in Σ_{CaAL}^F for each sort $(\sigma^n \Rightarrow \tau)$.

$\cdot[\cdot, \dots, \cdot] : (\sigma^n \Rightarrow \tau) \times \sigma^n \rightarrow \tau$	Reading of array values.
$store : (\sigma^n \Rightarrow \tau) \times \sigma^n \times \tau \rightarrow (\sigma^n \Rightarrow \tau)$	Updating of array values.
$K : \tau \rightarrow (\sigma^n \Rightarrow \tau)$	Construction of constant arrays.
$map_f : (\sigma^n \Rightarrow \tau_1) \times \dots \times (\sigma^n \Rightarrow \tau_k) \rightarrow (\sigma^n \Rightarrow \tau)$	Point-wise extension of base function $f : \tau_1 \times \dots \times \tau_k \rightarrow \tau$.
$proj_k : (\sigma^n \Rightarrow \tau) \times \sigma \rightarrow (\sigma^{n-1} \Rightarrow \tau)$	For $n > 1$ and $k \in \{1, \dots, n\}$, projection to $n-1$ of the indexes.
$arrayStore_k : (\sigma^n \Rightarrow \tau) \times \sigma \times (\sigma^{n-1} \Rightarrow \tau) \rightarrow (\sigma^n \Rightarrow \tau)$	For $n > 1$ and $k \in \{1, \dots, n\}$, update of a sub-array.

$(\Sigma_{\text{CaAL}}^S, \Sigma_{\text{CaAL}}^F, \Sigma_{\text{CaAL}}^P)$ of CaAL is then defined as follows. The set of sorts is the least set Σ_{CaAL}^S such that (i) $\Sigma_B^S \subseteq \Sigma_{\text{CaAL}}^S$, and (ii) $\sigma, \tau \in \Sigma_{\text{CaAL}}^S$ and $n \in \mathbb{N}_{>0}$ imply $(\sigma^n \Rightarrow \tau) \in \Sigma_{\text{CaAL}}^S$. A sort $(\sigma^n \Rightarrow \tau)$ is an array sort of arity n with index sort σ and value sort τ .

The set Σ_{CaAL}^F includes Σ_B^S , as well as the operations listed in Table 1 for every array sort $(\sigma^n \Rightarrow \tau)$. The operators $\cdot[\cdot, \dots, \cdot]$ and $store$ are the functions for reading from and writing to arrays, as in the standard theory of arrays. K and map_f correspond to the functions introduced in CAL [11]; in particular, any base function $f \in \Sigma_B^F$ is lifted to an operator on arrays using map_f . The operators $proj$ and $arrayStore$ are specific to our theory CaAL, and can be used to project an n -dimensional array to an $(n-1)$ -dimensional sub-array by fixing the value of the k 'th index, and to update the corresponding portion of the original array, respectively. The set Σ_{CaAL}^P coincides with Σ_B^P . Semantics is defined by the axiom schemata in Table 2.

Example 1. We illustrate the use of two-dimension arrays $s, s' : (\mathbb{B}^2 \Rightarrow \mathbb{C})$ to encode two-qubit quantum states. Suppose that s represents the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, and $s' = X_2(s)$ is the quantum state after applying an X gate (the quantum version of a ‘‘not’’-gate) on the 2nd qubit of s . The matrix representations of s and s' are as follows; note that the results of $x_2 = 0$ and $x_2 = 1$ are swapped in s and s' .

$$s = \begin{smallmatrix} & x_1=0 & x_1=1 \\ x_2=0 & \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \\ x_2=1 & & \end{smallmatrix}, \quad s' = \begin{smallmatrix} & x_1=0 & x_1=1 \\ x_2=0 & \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 \end{pmatrix} \\ x_2=1 & & \end{smallmatrix}.$$

The projection $proj_1(s, k)$ maps the matrix s to its k 'th column vector, specifically the column with $x_1 = k$. In CaAL, we can construct s' from s as $s' = arrayStore_2(arrayStore_2(K(0), 1, proj_2(s, 0)), 0, proj_2(s, 1))$. To compute the sum of the two matrices, we use $map_+(s, s')$, which is also utilized for other quantum gate operations.

Several extensions of the theory of Cartesian arrays are possible but beyond the scope of this paper. Those include (i) arrays with multiple different index sorts, as opposed to

Table 2: Axioms of the Theory of Cartesian Arrays. As shorthand notation, we write $\bar{i} : \sigma^n$ for a vector of n index variables $i_1 : \sigma, \dots, i_n : \sigma$.

$\forall a : (\sigma^n \Rightarrow \tau), \bar{i} : \sigma^n, x : \tau.$	(1)
$store(a, \bar{i}, x)[\bar{i}] = x$	
$\forall a : (\sigma^n \Rightarrow \tau), \bar{i} : \sigma^n, \bar{j} : \sigma^n, x : \tau.$	(2)
$\bar{i} = \bar{j} \vee store(a, \bar{i}, x)[\bar{j}] = a[\bar{j}]$	
$\forall a, b : (\sigma^n \Rightarrow \tau). \exists \bar{i} : \sigma^n.$	(3)
$a = b \vee a[\bar{i}] \neq b[\bar{i}]$	
$\forall x : \tau, \bar{i} : \sigma^n.$	(4)
$K(x)[\bar{i}] = x$	
$\forall a_1 : (\sigma^n \Rightarrow \tau_1), \dots, a_k : (\sigma^n \Rightarrow \tau_k), \bar{i} : \sigma^n.$	(5)
$map_f(a_1, \dots, a_k)[\bar{i}] = f(a_1[\bar{i}], \dots, a_k[\bar{i}])$	
$\forall a : (\sigma^n \Rightarrow \tau), \bar{i} : \sigma^n.$	(6)
$proj_k(a, i_k)[i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_n] = a[\bar{i}]$	
$\forall a : (\sigma^n \Rightarrow \tau), b : (\sigma^{n-1} \Rightarrow \tau), \bar{i} : \sigma^n.$	(7)
$arrayStore_k(a, i_k, b)[\bar{i}] = b[i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_n]$	
$\forall a : (\sigma^n \Rightarrow \tau), b : (\sigma^{n-1} \Rightarrow \tau), \bar{i} : \sigma^n, j : \sigma.$	(8)
$j = i_k \vee arrayStore_k(a, j, b)[\bar{i}] = a[\bar{i}]$	

just n copies of the same index sort σ ; and (ii) a theory that also includes point-wise extensions of predicates.

2.3 Complexity of Satisfiability in CaAL

We now study the hardness of satisfiability of quantifier-free CaAL formulas. The quantified Boolean formula problem (QBF) generalizes the Boolean satisfiability problem by allowing *existential* and *universal* quantifiers to be applied to variables. Its satisfiability problem is *PSPACE-complete* [28]. Without loss of generality, we can assume that QBF formulas are in *prenex normal form* $Q_1 x_1. Q_2 x_2. \dots Q_n x_n. \phi$, which consists of a Boolean formula ϕ over n Boolean variables x_1, \dots, x_n , and a prefix of quantifiers $Q_1, Q_2, \dots, Q_n \in \{\forall, \exists\}$.

To reduce the satisfiability problem of QBF to CaAL, we assume that the base theory provides a sort \mathbb{B} with the standard operations. This sort will be used for both index and values. An array $\text{toCaAL}(\phi) : (\mathbb{B}^n \Rightarrow \mathbb{B})$ encoding the semantics of ϕ is defined recursively as follows:

- $\text{toCaAL}(x_k) = arrayStore_k(K(0), 1, K(1))$.
- $\text{toCaAL}(\neg\phi) = map_{\neg}(\text{toCaAL}(\phi))$.
- $\text{toCaAL}(\phi_1 \wedge \phi_2) = map_{\wedge}(\text{toCaAL}(\phi_1), \text{toCaAL}(\phi_2))$.

Observe that $\text{arrayStore}_k(K(0), 1, K(1))[i_1, \dots, i_k, \dots, i_n] = i_k$, and note that the size of $\text{toCaAL}(\phi)$ is linear in the size of ϕ . We can construct a CaAL formula that is equisatisfiable with $Q_1 x_1 \cdots Q_n x_n. \phi$ as follows:

$$\begin{aligned} \text{QElim}(Q_1 x_1 \cdots Q_n x_n. \phi) = \\ (q_1[0] \odot_1 q_1[1]) \wedge \bigwedge_{i=2}^n q_{i-1} = \text{map}_{\odot_i}(\text{proj}_i(q_i, 0), \text{proj}_i(q_i, 1)) \wedge q_n = \text{toCaAL}(\phi) \end{aligned}$$

where $\odot_i = \wedge$ when $Q_i = \forall$, and $\odot_i = \vee$ otherwise. Note that the QBF formula $Q_1 x_1 \cdots Q_n x_n. \phi$ is valid if and only if the CaAL formula $\text{QElim}(Q_1 x_1 \cdots Q_n x_n. \phi)$ is satisfiable.

Theorem 1. *The satisfiability problem of CaAL over \mathbb{B} is PSPACE-hard.*

This lower bound can be improved, however. The k -colorability problem for graphs with succinct circuit representation is NEXPTIME-complete [23]. This problem can be reduced to the satisfiability problem of CaAL in polynomial time as well.

Consider an undirected graph with 2^n nodes, and let $\phi(\bar{x}, \bar{x}')$ be a Boolean circuit encoding the edge relation of the graph: $\phi(\bar{x}, \bar{x}')$ evaluates to true whenever there is an edge $(\bar{x}) \rightarrow (\bar{x}')$ in the graph. The k -colorability of the graph can be characterized as the following formula, where $c : (\mathbb{B}^n \rightarrow \mathbb{N})$ is an array representing the color of each node:

$$\forall \bar{x}, \bar{x}' : \mathbb{B}^n. \phi(\bar{x}, \bar{x}') \rightarrow c[\bar{x}] \neq c[\bar{x}'] \wedge c[\bar{x}] < k \wedge c[\bar{x}'] < k.$$

In a similar way as for QBF, we encode ϕ as an array formula ϕ' of linear size, in which $a_\phi : (\mathbb{B}^n \times \mathbb{B}^n \Rightarrow \mathbb{B})$ is an array variable representing the edge relation. We then create two intermediate arrays $a, b : (\mathbb{B}^n \times \mathbb{B}^n \Rightarrow \mathbb{N})$ and use the following formula in CaAL to encode the relation $\forall \bar{x}, \bar{x}' : \mathbb{B}^n. a[\bar{x}, \bar{x}'] = c[\bar{x}] \wedge b[\bar{x}, \bar{x}'] = c[\bar{x}']$:

$$\begin{aligned} \text{EqColor}(a, b, c) \equiv \\ a = a_n \wedge c = a_0 \wedge \bigwedge_{j=1}^n \text{proj}_{j+n}(a_j, 0) = \text{proj}_{j+n}(a_j, 1) = a_{j-1} \wedge \\ b = b_n \wedge c = b_0 \wedge \bigwedge_{j=1}^n \text{proj}_j(b_j, 0) = \text{proj}_j(b_j, 1) = b_{j-1} \end{aligned}$$

Then we encode the k -color problem with the following CaAL formula:

$$\begin{aligned} \phi' \wedge \text{EqColor}(a, b, c) \wedge \text{map}_f(a_\phi, a, b) = K(1) \\ \text{where } f(e, \text{col1}, \text{col2}) \equiv e \rightarrow (\text{col1} \neq \text{col2} \wedge \text{col1} < k \wedge \text{col2} < k). \end{aligned}$$

Theorem 2. *The satisfiability problem of CaAL is NEXPTIME-hard.*

3 Array Semantics of Quantum Circuits

As an application, we show that CaAL can encode the semantics of quantum circuits. Below, we only give a short overview of quantum circuits and define notations; for more details, see, e.g., the textbook of Nielsen and Chuang [22].

In a n -qubit quantum, a state is a *superposition* of *computational basis states* $\{|j\rangle \mid j \in \{0, 1\}^n\}$. For example, for a system with three qubits x_1 , x_2 , and x_3 , the computational basis state $|101\rangle$ (in Dirac notation) denotes a state in which both x_1 and x_3 are set to 1, and x_2 is set to 0. A n -qubit quantum state s is then denoted as a formal sum $\sum_{j \in \{0, 1\}^n} c_j \cdot |j\rangle$, where $c_0, c_1, \dots, c_{2^n - 1} \in \mathbb{C}$ are *complex probability amplitudes* satisfying the constraint that $\sum_{j \in \{0, 1\}^n} |c_j|^2 = 1$. Intuitively, $|c_j|^2$ is the probability that when we measure the quantum state s in the computational basis, we obtain the basis state $|j\rangle$. The constraint $\sum_{j \in \{0, 1\}^n} |c_j|^2 = 1$ states that probabilities need to sum up to 1 for all computational basis states.

We can record a quantum state as an array that maps a computational basis state to its complex probability amplitudes. The state s is represented as an array $s : (\mathbb{B}^n \Rightarrow \mathbb{C})$ satisfying $s[j] = c_j$ for all $j \in \{0, 1\}^n$; slightly abusing notation, we denote both the state and the array by s .

3.1 Quantum Circuits

A *quantum circuit* consists of a sequence of *quantum gates*. Each quantum gate defines a specific transformation of quantum states. For example, the Pauli- X gate (the quantum version of classical “not” gate) on the k -th qubit transforms a state s to s' satisfying $\forall i \in \{0, 1\}^{k-1}, b \in \{0, 1\}, j \in \{0, 1\}^{n-k} : s'[ibj] = s[i\bar{b}j]$, i.e., it negates the k -th index bit.

Another example is the Pauli- Z gate on the k -th qubit, which transforms a state s to s' satisfying $\forall i \in \{0, 1\}^{k-1}, b \in \{0, 1\}, j \in \{0, 1\}^{n-k} : s'[ibj] = \text{ite}(b, -1 \cdot s[i\bar{b}j], s[i\bar{b}j])$. Here, probability amplitudes are multiplied with -1 when b is 1, and are unchanged otherwise.

A H gate, or Hadamard gate, on the k -th qubit transforms a state s to s' satisfying $\forall i \in \{0, 1\}^{k-1}, b \in \{0, 1\}, j \in \{0, 1\}^{n-k} :$

$$s'[ibj] = \text{ite}(b, \frac{s[i0j] - s[i1j]}{\sqrt{2}}, \frac{s[i0j] + s[i1j]}{\sqrt{2}}).$$

Notice that the amplitude of a basis state of s' is affected by the amplitude of two basis states of s , enabling a more diverse superposition. The division with $\sqrt{2}$ is for normalizing the probability sum.

A more advanced class of gates are multiple-qubit gates. The CX gate (“controlled- X ”) on the control qubit c and target qubit t applies an X gate to t when c is 1, and is identity otherwise. Formally, assuming $c < t$, the gate transforms a state s to s' satisfying $\forall i_1 \in \{0, 1\}^{c-1}, b_c \in \{0, 1\}, i_2 \in \{0, 1\}^{t-c-1}, b_t \in \{0, 1\}, i_3 \in \{0, 1\}^{n-t} :$

$$s'[i_1 b_c i_2 b_t i_3] = \text{ite}(b_c, s[i_1 b_c i_2 \bar{b}_t i_3], s[i_1 b_c i_2 b_t i_3]).$$

The Toffoli gate CCX (“controlled-controlled- X gate”) has two control qubit c, d and applies the X gate to the target qubit t only when $c = d = 1$.

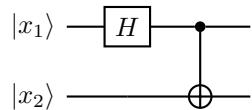


Fig. 2: The EPR circuit, consisting of an H and a CX gate with control qubit (•) and target qubit (⊕).

Table 3: Semantics of quantum gates in Cartesian array logic. We use s and s' to denote the quantum state before and after executing the circuit.

Gate	Formula
X_k	$\text{proj}_k(s', 0) = \text{proj}_k(s, 1) \wedge$ $\text{proj}_k(s', 1) = \text{proj}_k(s, 0)$
Y_k	$\text{proj}_k(s', 0) = \text{map}_{*(-\omega^2)} \text{proj}_k(s, 1) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{*(\omega^2)} \text{proj}_k(s, 0)$
Z_k	$\text{proj}_k(s', 0) = \text{proj}_k(s, 0) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{*(-1)} \text{proj}_k(s, 1)$
S_k	$\text{proj}_k(s', 0) = \text{proj}_k(s, 0) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{*(\omega^2)} \text{proj}_k(s, 1)$
T_k	$\text{proj}_k(s', 0) = \text{proj}_k(s, 0) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{*(\omega)} \text{proj}_k(s, 1)$
H_k	$\text{proj}_k(s', 0) = \text{map}_{(+)}/\sqrt{2}(\text{proj}_k(s, 0), \text{proj}_k(s, 1)) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{(-)}/\sqrt{2}(\text{proj}_k(s, 0), \text{proj}_k(s, 1))$
$Rx(\frac{\pi}{2})_k$	$\text{proj}_k(s', 0) = \text{map}_{(+(-\omega^2) * .)}/\sqrt{2}(\text{proj}_k(s, 0), \text{proj}_k(s, 1)) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{((-\omega^2) * . + .)}/\sqrt{2}(\text{proj}_k(s, 0), \text{proj}_k(s, 1))$
$Ry(\frac{\pi}{2})_k$	$\text{proj}_k(s', 0) = \text{map}_{(-.)/\sqrt{2}}(\text{proj}_k(s, 0), \text{proj}_k(s, 1)) \wedge$ $\text{proj}_k(s', 1) = \text{map}_{(+.)/\sqrt{2}}(\text{proj}_k(s, 0), \text{proj}_k(s, 1))$
$CX_{c,t}$	$\text{proj}_c(s', 0) = \text{proj}_c(s, 0) \wedge$ $\text{proj}_t(\text{proj}_c(s', 1), 0) = \text{proj}_t(\text{proj}_c(s, 1), 1) \wedge$ $\text{proj}_t(\text{proj}_c(s', 1), 1) = \text{proj}_t(\text{proj}_c(s, 1), 0)$
$CZ_{c,t}$	$\text{proj}_c(s', 0) = \text{proj}_c(s, 0) \wedge$ $\text{proj}_t(\text{proj}_c(s', 1), 0) = \text{proj}_t(\text{proj}_c(s, 1), 0) \wedge$ $\text{proj}_t(\text{proj}_c(s', 1), 1) = \text{map}_{*(-1)} \text{proj}_t(\text{proj}_c(s, 1), 1)$
$CCX_{c,d,t}$	$\text{proj}_c(s', 0) = \text{proj}_c(s, 0) \wedge$ $\text{proj}_d(s', 0) = \text{proj}_d(s, 0) \wedge$ $\text{proj}_t(\text{proj}_d(\text{proj}_c(s', 1), 1), 0) = \text{proj}_t(\text{proj}_d(\text{proj}_c(s, 1), 1), 1) \wedge$ $\text{proj}_t(\text{proj}_d(\text{proj}_c(s', 1), 1), 1) = \text{proj}_t(\text{proj}_d(\text{proj}_c(s, 1), 1), 0)$

We have introduced enough quantum gates to define the EPR circuit (Fig. 2), named after Einstein, Podolsky, and Rosen for constructing the Bell state, i.e., a 2-qubit circuit converting a basis state $|00\rangle$ to a maximally entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Starting from a state s (represented s that maps 00 to 1 and others to 0, the circuit first applies H on the first qubit x_1 (denoted H_1 in this paper) to produce the quantum state s' with $s'[00] = s'[10] = \frac{1}{\sqrt{2}}$ and $s'[11] = s'[01] = 0$. Then a $CX_{1,2}$ converts it further to s'' with $s''[00] = s''[11] = \frac{1}{\sqrt{2}}$ and $s''[01] = s''[10] = 0$. Notice that $CX_{1,2}$ converts $|10\rangle$ to $|11\rangle$, i.e., when x_1 is 1, it negates x_2 .

Note on complexity. Simulation of a quantum circuit is bounded-error quantum polynomial time (BQP) hard, a complexity class that is incomparable with NP, as it can compute exactly the probability amplitudes of a quantum state after executing a circuit. We will show that the Cartesian array logic can encode the semantics of quantum circuits, so

one can also use the logic for quantum circuit simulation. Hence, exponential time is the best deterministic algorithm we can hope for when solving CaAL formulas.

3.2 Interpretation of Quantum Gates

We show the encoding of quantum gates in CaAL in Table 3. Notice that this gate set includes several universal gates (e.g., H , CX , and T [10]) that can approximate *any quantum gate* to an arbitrary precision requirement. Arbitrary degree rotation can also be supported using the theory of reals as the base theory. This paper presents a precise encoding that only requires a theory of integers. In the figure, we use s and s' to denote the quantum states (encoded as arrays) before and after executing a quantum gate. To encode $s' = X_k(s)$, negating the k -th qubit, we use $\text{proj}_k(s', 0) = \text{proj}_k(s, 1) \wedge \text{proj}_k(s', 1) = \text{proj}_k(s, 0)$: index $k = 0$ in s' equals the case of $k = 1$ in s . The handling of Z , S , and T gates is similar, using the *map* function to multiply the array values with different constants. Note that here we use ω to represent $e^{\frac{\pi i}{4}} = \cos \frac{\pi}{4} + i \sin \frac{\pi}{4} = \frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}$, the unit vector that is at an angle of 45° to the positive real axis in the complex plane. Later we will show that this representation allows a precise algebraic representation of complex numbers using a five-tuple of integers. Observe that $\omega^4 = -1$. The Y gate combines the two constructions; it negates the k -th index qubit and multiplies each projection with different constant coefficients. For the H , $Rx(\frac{\pi}{2})$, and $Ry(\frac{\pi}{2})$ gates, we use a binary *map* function to update the amplitudes. For the controlled gates, we use the projection function to classify the cases according to the control bits and apply the X or Z gate only when all controlled bits are 1.

Example 2. We use CaAL to verify the correctness of the EPR circuit Fig. 2: the circuit transforms the state $|00\rangle$ to $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. For this, the initial state of the circuit is encoded as an array expression, the H and CX gates are encoded according to Table 3, and the intended final state of the circuit is represented as a negated equation:

$$\begin{aligned}
 s_0 &= \text{store}(K(0), (0, 0), 1) \\
 \wedge \text{proj}_1(s_1, 0) &= \text{map}_{(+.)/\sqrt{2}}(\text{proj}_1(s_0, 0), \text{proj}_1(s_0, 1)) \quad \left. \right\} s_1 = H_1(s_0) \\
 \wedge \text{proj}_1(s_1, 1) &= \text{map}_{(-.)/\sqrt{2}}(\text{proj}_1(s_0, 0), \text{proj}_1(s_0, 1)) \\
 \wedge \text{proj}_1(s_2, 0) &= \text{proj}_1(s_1, 0) \\
 \wedge \text{proj}_2(\text{proj}_1(s_2, 1), 0) &= \text{proj}_2(\text{proj}_1(s_1, 1), 1) \quad \left. \right\} s_2 = CX_{1,2}(s_1) \\
 \wedge \text{proj}_2(\text{proj}_1(s_2, 1), 1) &= \text{proj}_2(\text{proj}_1(s_1, 1), 0) \\
 \wedge s_2 &\neq \text{store}(\text{store}(K(0), (1, 1), \frac{1}{\sqrt{2}}), (0, 0), \frac{1}{\sqrt{2}})
 \end{aligned}$$

The formula is unsatisfiable if and only if the EPR circuit correctly performs the transformation.

Representation of complex numbers. To achieve accuracy with no loss of precision, in this paper, when working with \mathbb{C} , we use a subset of the complex numbers that the

Table 4: Tableau proof rules of the decision procedure for CaAL.

idx	$\frac{a = \text{store}(b, \bar{i}, v)}{v = a[\bar{i}]}$	$\text{K} \Downarrow \frac{a = K(v) \quad w = a'[\bar{i}] \quad a \sim a'}{v = w}$
$\text{store} \Downarrow$	$\frac{a = \text{store}(b, \bar{i}, v) \quad w = a'[\bar{j}] \quad a \sim a'}{\bar{i} = \bar{j} \quad \quad w = b[\bar{j}]}$	
$\text{store} \Uparrow$	$\frac{a = \text{store}(b, \bar{i}, v) \quad w = b'[\bar{j}] \quad b \sim b'}{\bar{i} = \bar{j} \quad \quad w = a[\bar{j}]}$	
$\text{map} \Downarrow$	$\frac{a = \text{map}_f(b_1, \dots, b_m) \quad w = a'[\bar{i}] \quad a \sim a'}{w = f(b_1[\bar{i}], \dots, b_m[\bar{i}])}$	
$\text{map} \Uparrow$	$\frac{a = \text{map}_f(b_1, \dots, b_m) \quad w = b'[\bar{i}] \quad b' \sim b_k \text{ for some } k \in \{1, \dots, m\}}{a[\bar{i}] = f(b_1[\bar{i}], \dots, b_{k-1}[\bar{i}], w, b_{k+1}[\bar{i}], \dots, b_m[\bar{i}])}$	
$\text{proj} \Downarrow$	$\frac{a = \text{proj}_k(b, j) \quad w = a'[\bar{i}] \quad a \sim a'}{w = b[i_1, \dots, i_{k-1}, j, i_k, \dots, i_{n-1}]}$	
$\text{proj} \Uparrow$	$\frac{a = \text{proj}_k(b, j) \quad w = b'[\bar{i}] \quad b \sim b'}{j \neq i_k \quad \quad w = a[i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n]}$	
$\text{arrayStore} \Downarrow$	$\frac{a = \text{arrayStore}_k(b, j, c) \quad w = a'[\bar{i}] \quad a \sim a'}{j = i_k \wedge w = c[i_1, \dots, i_{k-1}, i_{k+1}, \dots] \quad \quad j \neq i_k \wedge w = b[\bar{i}]}$	
$\text{arrayStore} \Uparrow_1$	$\frac{a = \text{arrayStore}_k(b, j, c) \quad w = b'[\bar{i}] \quad b \sim b'}{j = i_k \quad \quad w = a[\bar{i}]}$	
$\text{arrayStore} \Uparrow_2$	$\frac{a = \text{arrayStore}_k(b, j, c) \quad w = c'[\bar{i}] \quad c \sim c'}{w = a[i_1, \dots, i_{k-1}, j, i_k, \dots, i_{n-1}]}$	
ext	$\frac{a : (\sigma^n \Rightarrow \tau) \quad b : (\sigma^n \Rightarrow \tau)}{a = b \quad \quad \exists \bar{i} : \sigma^n. a[\bar{i}] \neq b[\bar{i}]}$	$\text{freshIdx} \frac{i_1, \dots, i_k : \sigma}{\exists j : \sigma. j \neq i_1 \wedge \dots \wedge j \neq i_k}$
read	$\frac{a : (\sigma^n \Rightarrow \tau) \quad \bar{i} : \sigma^n}{\exists v : \tau. v = a[\bar{i}]}$	$\text{readCong} \frac{v = a[\bar{i}] \quad w = b[\bar{j}] \quad a \sim b}{\bar{i} \neq \bar{j} \quad \quad \bar{i} = \bar{j} \wedge v = w}$

following algebraic encoding can express (cf. [35,30,7]):

$$\left(\frac{1}{\sqrt{2}}\right)^k (a + b\omega + c\omega^2 + d\omega^3), \quad (9)$$

where $a, b, c, d, k \in \mathbb{Z}$. A complex number is then represented by a five-tuple (a, b, c, d, k) . Although the considered set of numbers is only a small subset of \mathbb{C} , it is closed under the operations needed to encode quantum gates, and it can arbitrarily closely approximate any complex number. For this, note that $(a, 0, c, 0, k)$ represents $\frac{1}{\sqrt{2^k}}(a + c\omega^2) = \frac{a}{\sqrt{2^k}} + \frac{ci}{\sqrt{2^k}}$, and pick suitable a, c , and k . The representation is also sufficient to describe a set of quantum gates that can implement universal quantum computation (Table 3).

4 A Decision Procedure for Cartesian Arrays

We now present a decision procedure for quantifier-free CaAL. Our calculus is an extension of the calculus for CAL [11] with rules for the *proj* and *arrayStore* operations. For the sake of presentation, we use the setting of analytic tableaux [14], although the same proof rules can be used also in a model-constructing calculus [11].

As a simplifying assumption, in this section we furthermore require that the index sorts σ of an array sort ($\sigma^n \Rightarrow \tau$) represent *infinite* domains. This assumption can be lifted in the same way as for CAL [11], but the details are orthogonal to the task of supporting the new array operations.

4.1 Preliminaries

A tableau [14] is a finite tree growing downwards, in which each node is labelled with a formula, the root is labelled with the formula to be refuted, and the children of each node are derived from the formulas on the branch leading to the node using one of the available proof rules. We assume a tableau calculus equipped with a set of standard rules [14]: (i) α - and β -rules for eliminating Boolean connectives \wedge, \vee ; (ii) δ -rules for eliminating existential quantifiers \exists ; (iii) rules for reasoning about positive and negative equalities $x = y$ between variables, which include rules for closing proof branches; (iv) rules implementing a decision procedure for the base theory T_B .

Our calculus operates on *flat* formulas, which are formulas in which functions f only occur in equations $y = f(\bar{x})$ in positive positions, i.e., underneath an even number of negations, with y, \bar{x} being variables. Every formula can be converted to a flat formula by introducing a linear number of new variables.

We define proof rules using the following notation:

$$\text{rule } \frac{\phi_1 \quad \phi_2 \quad \cdots \quad \phi_k}{\psi_1 \quad | \quad \cdots \quad | \quad \psi_m}$$

The rule is applicable if the premises ϕ_1, \dots, ϕ_k occur on a proof branch, and has the effect of expanding the tableau: the proof branch is split into m new branches, to which the formulas ψ_1, \dots, ψ_m , respectively, are appended.

In the premises of a rule, we frequently include assumptions $x \sim y$ that require that the equality $x = y$ follows from positive equalities between variables on the proof branch. We also use premises $x : \sigma$, stating that x is a variable of sort σ occurring on the proof branch.

4.2 Proof Rules

The rules of our calculus are shown in Table 4. The rules *idx*, $K \Downarrow$, *store* \Downarrow , *store* \Uparrow , $\text{map} \Downarrow$, $\text{map} \Uparrow$ coincide with the rules used for CAL [11], and define the semantics of the operators K , *store*, and *map*. Extensionality is implemented by the rule *ext*, which can be applied for any two array variables a, b of the same type occurring on a branch.

The semantics of *proj* and *arrayStore* is defined, in a similar way as for *store*, by upward and downward propagation of array reads. Since $\text{arrayStore}_k(b, j, c)$ combines

two arrays b, c into a single new array, downward propagation has to route reads either to b or to c . Upward propagation from c is always possible, while reads on b can only be propagated if they are not overwritten by c .

For sake of presentation, we write the conclusion in the rules $\text{map}\Downarrow$, $\text{map}\Uparrow$, and ext in non-flat form, and assume that the transformation to a flat formula happens implicitly by adding existentially quantified variables representing the sub-terms.

Congruence reasoning is necessary only for array reads, and implemented using the rule readConq . For simplicity, in our formulation the rule splits over the cases $\bar{i} \neq \bar{j}$ and $\bar{i} = \bar{j}$, and effectively searches for an arrangement of the index variables satisfying a formula. An actual implementation could rely on equality propagation being performed by a theory combination procedure.

As one of the more tricky points, the completeness of the calculus sometimes requires new array reads to be generated. This aspect is covered by the rules ϵ_{\neq} and ϵ_{δ} in CAL [11], which are rules that can, however, not directly be used in our setting of multi-dimensional arrays. To obtain completeness, our calculus sometimes has to construct reads by combining different index variables occurring on a branch, and sometimes invent index values that are distinct from all indexes occurring in a formula. The introduction of corresponding new reads is handled by the rules freshIdx and read .

Example 3. Consider arrays $a, b : (\mathbb{Z}^2 \Rightarrow \mathbb{Z})$, and the formulas

$$\text{proj}_1(a, i) = K(42) \wedge \text{proj}_2(a, j) = K(43) \quad (10)$$

$$a = K(42) \wedge b = \text{store}(a, (i, i), 43) \wedge \text{proj}_1(b, i) = K(43) \quad (11)$$

Both formulas are unsatisfiable, but cannot be refuted using the rules discussed so far. In (10), no reads $a[\dots]$ exist, so that no propagations can be performed by any of the rules. It is necessary to identify the constraints on the value $a[i, j]$ as contradictory. The rule read can be used to introduce a new formula $\exists v. v = a[i, j]$ on a proof branch, after which the rules $\text{proj}\Uparrow$ and $\text{K}\Downarrow$ can be applied.

To show that (11) is unsatisfiable, we need to consider a point (i, j) with $j \neq i$ and derive that $a[i, j] = b[i, j] = 42$, and contradicting $\text{proj}_1(b, i) = K(43)$. The introduction of a fresh index value j (different from i) is handled by the rule freshIdx , which relies on the index sort σ representing an infinite domain. Once the existence of an index $j \neq i$ has been asserted, the rule read can be used to introduce an equation $v = a[i, j]$, and the contraction be derived.

4.3 Correctness and Complexity

Theorem 3. *The presented tableau calculus is sound and complete for flat quantifier-free CaAL formulas: there is a closed tableau for a formula ϕ if and only if ϕ is unsatisfiable.*

Proof. Soundness: As usual, we identify each proof branch with the conjunction of its formulas and a tableau with the disjunction of its proof branches. It can be shown that the tableau before expansion using a proof rule is equi-satisfiable to the tableau before the expansion, modulo the array axioms in Table 2.

Completeness: We make the simplifying assumption that ϕ only contains arrays with (infinite) index sort σ and value sort τ , and in particular that array sorts are not nested. Completeness for the general case follows by recursively applying model construction.

Consider then the systematic construction of a tableau for a formula ϕ by exhaustively applying proof rules under the following restrictions: (i) regularity, i.e., rules are only applied if they lead to new formulas being added to each generated branch; (ii) rule `freshIdx` can only be applied once on a branch, only after `ext` has been applied to all pairs a, b of array variables on the branch, and choosing i_1, \dots, i_k as the set of all variables of sort σ on the branch.

Observe that this systematic application of rules terminates: the calculus never introduces new array variables so that only finitely many applications of `ext` are possible. Note that `ext` and `freshIdx` are the only rules introducing new index variables. Since `freshIdx` is applied at most once on a branch, the set of index variables is bounded, and there is only a bounded number of array reads $v = a[\bar{i}]$.

Assume now that a tableau for ϕ cannot be closed, i.e., has at least one branch B that cannot be closed, although all possible rule applications have been performed. We extract a model of ϕ from B . Suppose that $M_T = (D_T, I_T)$ is a model that interprets the non-array-variables (including index variables), satisfying all literals on B that do not contain array variables, and denote the equivalence class of an array variable a on B by $[a] = \{b \mid a \sim b\}$. Extending I_T , we construct an interpretation I with $I((\sigma^n \Rightarrow \tau)) = I_T(\sigma)^n \rightarrow I_T(\tau)$ being a function space, and the theory functions $\cdot[\cdot]$, `store`, K , map_f , `proj` and `arrayStore` having their expected meaning. I is constructed in such a way that all array literals on B are satisfied; the satisfaction of compound formulas on B , and in particular of ϕ , then follows like in the standard Hintikka construction [14].

The interpretation $I(a)$ of an array variable $a : (\sigma^n \Rightarrow \tau)$ is derived from the array reads on $[a]$ occurring on B . The main difficulty is to consistently interpret the (infinitely many) elements of the array that are not mentioned explicitly on B . For this, denote the index variable introduced by the unique `freshIdx` application on B by ϵ , and observe that its value $I_T(\epsilon)$ is distinct from the value of all other index variables. We will use values read from $I_T(\epsilon)$ -locations as default values for the arrays. Let

$$R_a = \{(\langle I_T(i_1), \dots, I_T(i_n) \rangle, I_T(v)) \mid v = b[\bar{i}] \text{ occurs on } B \text{ and } a \sim b\}$$

be the set of array reads for $a : (\sigma^n \Rightarrow \tau)$. The relation R_a describes a non-empty, consistent (but partial) valuation of the array elements, due to the exhaustive application of rules `read` and `readCong`.

The gaps in R_a will be filled with default values introduced by ϵ . For this, we define a precedence ordering $\preceq \subseteq I_T(\sigma)^* \times I_T(\sigma)^*$ over index vectors; intuitively, $\bar{c} \preceq \bar{d}$ if \bar{c} and \bar{d} agree in all components, unless $d_k = I_T(\epsilon)$, which is interpreted as don't-care:

$$\langle c_1, \dots, c_k \rangle \preceq \langle d_1, \dots, d_m \rangle \text{ iff } k = m \text{ and } \forall i \in \{1, \dots, k\} : c_i = d_i \vee d_i = I_T(\epsilon)$$

The value of array variable $I(a) \in I((\sigma^n \Rightarrow \tau))$ is then:

$$I(a) = \left\{ (\bar{c}, x) \mid \begin{array}{l} (\bar{d}, x) \in R_a, \text{ where } \bar{c} \preceq \bar{d} \\ \text{and for all } (\bar{d}', x') \in R_a : \text{if } \bar{c} \preceq \bar{d}' \text{ then } \bar{d} \preceq \bar{d}' \end{array} \right\}$$

To see that $I(a)$ is functionally consistent, note that whenever (\bar{d}, x) and (\bar{d}', x') exist in R_a such that $\bar{c} \preceq \bar{d}$ and $\bar{c} \preceq \bar{d}'$, then there is also some $(\bar{d}'', x'') \in R_a$ such that $\bar{c} \preceq \bar{d}'' \preceq \bar{d}, \bar{d}'$. This is because the rule `read` has been applied exhaustively.

It remains to be shown that I satisfies all array literals. By construction, equations $a = b$ will be satisfied. To see that equations $v = a[\bar{i}]$ hold, note that $I(a) \supseteq R_a$. Equations $a \neq b$ are satisfied due to the exhaustive application of `ext`: there has to be some vector \bar{i} of index variables such that $a[\bar{i}] \neq b[\bar{i}]$.

All other array literals are positive equations of the form $x = f(\bar{y})$, and hold because exhaustive propagation of read atoms was performed. As an example, consider an equation $a = \text{proj}_k(b, j)$; it has to be shown that $I(a) = \{(\langle c_1, \dots, c_{k-1}, c_{k+1}, \dots, c_n \rangle, x) \mid (\bar{c}, x) \in I(b), c_k = I_T(j)\}$. Observe that $R_a = \{(\langle c_1, \dots, c_{k-1}, c_{k+1}, \dots, c_n \rangle, x) \mid (\bar{c}, x) \in R_b, c_k = I_T(j)\}$ due to the rules `proj` \Downarrow and `proj` \Uparrow . Consider then a point $(\bar{c}, x) \in I(a)$, defined by $(\bar{d}, x) \in R_a$, and the corresponding index vectors $\bar{c}' = \langle c_1, \dots, c_{k-1}, I_T(j), c_k, \dots, c_{n-1} \rangle$ and $\bar{d}' = \langle d_1, \dots, d_{k-1}, I_T(j), d_k, \dots, d_{n-1} \rangle$ in R_b , and show that $(\bar{c}', x) \in I(b)$ is defined by $(\bar{d}', x) \in R_b$. \square

The proof of the theorem highlights the restrictions necessary to obtain a decision procedure for CaAL: all rules should be applied under the condition of regularity, and the rule `freshIdx` has to be restricted to at most one application per branch, and only after applications of `ext` have been performed.

To evaluate runtime, like in the proof of Theorem 3 we make the assumption that there are no nested array sorts, i.e., index and value sorts are themselves not arrays. To avoid degenerate cases when evaluating runtime, we assume that a formula ϕ cannot be smaller than the maximum arity of occurring array variables. We then get:

Lemma 1. *The satisfiability problem of quantifier-free CaAL formulas ϕ without nested array sorts is in NEXPTIME, assuming that the satisfiability problem of the base theory is in NP.*

Proof. This follows from the proof of Theorem 3. On every branch, the rule `ext` can be applied at most quadratically often, and the number of index variables occurring on a branch is polynomial in the size of the input formula ϕ . The number of distinct read atoms $v = a[\bar{i}]$ that can be introduced on a branch, and therefore the number of rule applications altogether is then polynomially bounded by the number of variables in ϕ , and exponentially bounded in the maximum arity of array variables in ϕ . After exhaustive application of the rules in Table 4, solving an at most exponential number of base theory formulas (with at most exponential size) on a branch is in NEXPTIME. \square

4.4 Optimizations

The calculus and decision procedure are primarily designed with simplicity in mind, rather than focusing on practical efficiency. Although the procedure's complexity may not be reduced below NEXPTIME, incorporating various optimizations can yield significant practical improvements. Two obvious improvements to be considered are: (i) The detection of **linear array variables**, which are essentially variables that are assigned to at most once in array literals [11]. It is enough to perform upward propagation (rules \Uparrow)

Table 5: Experimental results. We list the **circuit** name, the number of **qubits** and **gates** in the circuit, the verification **result**, and the execution **time**.

circuit	qubits	gates	result	time	circuit	qubits	gates	result	time
H^2	1	2	OK	3.1s	H^2 (bug)	1	2	bug	3.0s
BV	1	3	OK	3.2s	BV (bug)	1	3	bug	3.3s
BV	2	5	OK	6.4s	BV	5	13	OK	1m59.0s
BV	3	8	OK	16.8s	BV	6	15	OK	9m13s
BV	4	10	OK	43.2s	BV	7	18	OK	50m54s
Grover _{Single-Comp}	2	17	OK	5.2s	Grover _{Single-Comp}	4	85	OK	51.7s
Grover _{All-Comp}	2	17	OK	6.8s	Grover _{All-Comp}	4	85	OK	3m53s
Grover _{Single-Iter}	1	9	OK	3.2s	Grover _{All-Iter}	1	9	OK	3.8s
Grover _{Single-Iter}	2	15	OK	4.9s	Grover _{All-Iter}	2	15	OK	14.2s
Grover _{Single-Iter}	3	21	OK	8.4s	Grover _{All-Iter}	3	21	OK	37.9s
Grover _{Single-Iter}	4	27	OK	17.1s	Grover _{All-Iter}	4	27	OK	4m51s
Grover _{Single-Iter}	5	33	OK	46.9s	Grover _{All-Iter}	5	33	OK	57m2s

only for non-linear variables. (ii) The **restriction of the number of reads** introduced using the rule `read`. In practice, only a few of the generated equations are actually needed to ensure completeness. Instead of generating all possible reads eagerly, a procedure could focus on the other rules first, and only introduce additional reads when it is detected that default values are missing for some sub-arrays. We believe that other refinements presented in [11] can be carried over to our decision procedure as well.

5 Preliminary Experimental Result

We have implemented the decision procedure proposed for CaAL, the encoding of quantum gates using array operations, and of complex numbers as five-tuples of integers in the SMT solver Princess [26]. The implementation is still a proof of concept and largely unoptimized, so that the results reported in this section should be considered preliminary. We evaluate the performance of CaAL based on a set of benchmarks for quantum circuit verification. All experiments were conducted on a server with an AMD EPYC 7742 64-core processor (1.5 GHz), 1,152 GiB of RAM, and a 1 TB SSD running Ubuntu 20.04.5 LTS but were run with only one core for the sake of fairness. Files to reproduce the experiment can be found in <https://zenodo.org/record/7970588>. The experimental results are shown in Table 5. Specifically, we tested four different verification problems with different circuit sizes.

- H^2 : Two consecutive H gates equal to identity.
- BV: The (complex) amplitudes of the output quantum state from a Bernstein-Vazirani’s [4] circuit have no imaginary parts.
- Grover_{XXX-Comp}: The Grover’s [17] circuit has a probability of 90% to find the correct answer.
- Grover_{XXX-Iter}: Each Grover iteration [17] increases the possibility of finding the correct answer.

For Grover’s algorithm, XXX=Single means we check the correctness of the circuit against a specific oracle, and XXX=All means we check against all possible oracles. We manually injected two bugs (by altering one gate) into two examples to demonstrate bug-catching capability. With a timeout of 60min, our implementation can analyze circuits with at most 7 qubits and at most 85 gates, which are still relatively small circuits. Analyzing the results, we discovered that, in particular, the H gates used to create a superposition state at the beginning of a circuit are challenging for the array decision procedure, as they lead to an exponential number of array reads being created.

6 Conclusions

We have presented CaAL, an expressive logic of extensional arrays, with operations for reading and storing values, creating constant arrays, a point-wise extension of functions on array values to arrays, projection of arrays, and updating array slices. We have established that checking the satisfiability of quantifier-free CaAL formulas is NEXPTIME-complete, for a base theory in NP and non-nested arrays. The root cause for the complexity of CaAL (as opposed to the NP complexity of CAL and the standard theory of arrays) is that formulas can be constructed in which a cell in one array has dependencies to an exponential number of cells in another array. In our decision procedure, such situations lead to an exponential number of reads generated during propagation. High degrees of dependency are typical, however, for quantum circuits.

We believe that CaAL is a suitable framework for reasoning about quantum circuits. Due to the expressiveness of the logic, the encoding of quantum gates becomes remarkably succinct and elegant (Table 3), and easily understandable both for researchers in quantum circuit verification and people in automated reasoning. While theoretically optimal, we consider the decision procedure proposed for CaAL only as a first step: the high complexity of CaAL implies that brute-force approaches like saturation are unlikely to scale to interesting instances. As future work, we therefore plan to explore the use of abstraction methods and of more succinct array representations in the decision procedure, thus making it possible to exploit the highly structured nature of typical quantum circuits in the solving process. We also plan to investigate whether interesting fragments of CaAL with lower complexity can be identified.

Acknowledgements. This work has been partially funded by the Swedish Research Council (VR) under grant 2018-04727, the Swedish Foundation for Strategic Research (SSF) under the project WebSec (Ref. RIT17-0011), the Wallenberg project UPDATE, and the NSTC QC project under Grant no. NSTC 111-2119-M-001-004- and 112-2119-M-001-006-.

References

1. Amy, M.: Towards large-scale functional verification of universal quantum circuits. In: Quantum Physics and Logic (2018)
2. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017), available at www.SMT-LIB.org
3. Bauer-Marquart, F., Leue, S., Schilling, C.: symqv: Automated symbolic verification of quantum programs. In: 25th International Symposium on Formal Methods. Springer (2023)
4. Bernstein, E., Vazirani, U.V.: Quantum complexity theory. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA. pp. 11–20. ACM (1993). <https://doi.org/10.1145/167088.167097>, <https://doi.org/10.1145/167088.167097>
5. Bonacina, M.P., Graham-Lengrand, S., Shankar, N.: CDSAT for nondisjoint theories with shared predicates: Arrays with abstract length. In: Déharbe, D., Hyvärinen, A.E.J. (eds.) Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, August 11-12, 2022. CEUR Workshop Proceedings, vol. 3185, pp. 18–37. CEUR-WS.org (2022), <https://ceur-ws.org/Vol-3185/paper9712.pdf>
6. Brummayer, R., Biere, A.: Lemmas on demand for the extensional theory of arrays. *J. Satisf. Boolean Model. Comput.* **6**(1-3), 165–201 (2009). <https://doi.org/10.3233/sat190067>, <https://doi.org/10.3233/sat190067>
7. Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W., Yen, D.: An automata-based framework for verification and bug hunting in quantum circuits (2023). <https://doi.org/10.48550/arxiv.2301.07747>, <https://arxiv.org/abs/2301.07747>, to appear at PLDI 2023
8. Christ, J., Hoenicke, J.: Weakly equivalent arrays. In: Lutz, C., Ranise, S. (eds.) Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wrocław, Poland, September 21-24, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9322, pp. 119–134. Springer (2015). https://doi.org/10.1007/978-3-319-24246-0_8, https://doi.org/10.1007/978-3-319-24246-0_8
9. Coecke, B., Duncan, R.: Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics* **13**(4), 043016 (apr 2011). <https://doi.org/10.1088/1367-2630/13/4/043016>, <https://doi.org/10.1088%2F1367-2630%2F13%2F4%2F043016>
10. Dawson, C.M., Nielsen, M.A.: The Solovay-Kitaev algorithm. *arXiv preprint quant-ph/0505030* (2005)
11. De Moura, L., Bjørner, N.: Generalized, efficient array decision procedures. In: 2009 Formal Methods in Computer-Aided Design. pp. 45–52. IEEE (2009)
12. Falke, S., Merz, F., Sinz, C.: Extending the theory of arrays: memset, memcpy, and beyond. In: Cohen, E., Rybalchenko, A. (eds.) Verified Software: Theories, Tools, Experiments - 5th International Conference, VSTTE 2013, Menlo Park, CA, USA, May 17-19, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8164, pp. 108–128. Springer (2013). https://doi.org/10.1007/978-3-642-54108-7_6, https://doi.org/10.1007/978-3-642-54108-7_6
13. Feng, Y., Yu, N., Ying, M.: Model checking quantum Markov chains. *J. Comput. Syst. Sci.* **79**(7), 1181–1198 (2013). <https://doi.org/10.1016/j.jcss.2013.04.002>, <https://doi.org/10.1016/j.jcss.2013.04.002>
14. Fitting, M.C.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, 2nd edn. (1996)

15. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Computer Aided Verification: 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007. Proceedings 19. pp. 519–531. Springer (2007)
16. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 519–531. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_52, https://doi.org/10.1007/978-3-540-73368-3_52
17. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 212–219. ACM (1996). <https://doi.org/10.1145/237814.237866>, <https://doi.org/10.1145/237814.237866>
18. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press (2009)
19. Hietala, K., Rand, R., Hung, S.H., Wu, X., Hicks, M.: Verified optimization in a quantum intermediate representation. arXiv preprint arXiv:1904.06319 (2019)
20. Hoenicke, J., Schindler, T.: Solving and interpolating constant arrays based on weak equivalences. In: Enea, C., Piskac, R. (eds.) Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11388, pp. 297–317. Springer (2019). https://doi.org/10.1007/978-3-030-11245-5_14, https://doi.org/10.1007/978-3-030-11245-5_14
21. Mateus, P., Ramos, J., Sernadas, A., Sernadas, C.: Temporal Logics for Reasoning about Quantum Systems, p. 389–413. Cambridge University Press (2009). <https://doi.org/10.1017/CBO9781139193313.011>
22. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, USA, 10th edn. (2011)
23. Papadimitriou, C.H., Yannakakis, M.: A note on succinct representations of graphs. Information and control **71**(3), 181–185 (1986)
24. Perdrix, S.: Quantum entanglement analysis based on abstract interpretation. In: International Static Analysis Symposium. pp. 270–282. Springer (2008)
25. Raya, R., Kuncak, V.: NP satisfiability for arrays as powers. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation - 23rd International Conference, VMCAI 2022, Philadelphia, PA, USA, January 16-18, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13182, pp. 301–318. Springer (2022). https://doi.org/10.1007/978-3-030-94583-1_15, https://doi.org/10.1007/978-3-030-94583-1_15
26. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: Proceedings, 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning. LNCS, vol. 5330, pp. 274–289. Springer (2008)
27. Sheng, Y., Nötzli, A., Reynolds, A., Zohar, Y., Dill, D.L., Grieskamp, W., Park, J., Qadeer, S., Barrett, C.W., Tinelli, C.: Reasoning about vectors using an SMT theory of sequences. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13385, pp. 125–143. Springer (2022). https://doi.org/10.1007/978-3-031-10769-6_9, https://doi.org/10.1007/978-3-031-10769-6_9
28. Sipser, M.: Introduction to the theory of computation. ACM Sigact News **27**(1), 27–29 (1996)
29. Stump, A., Barrett, C.W., Dill, D.L., Levitt, J.: A decision procedure for an extensional theory of arrays. In: Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. pp. 29–37. IEEE (2001)

30. Tsai, Y., Jiang, J.R., Jhang, C.: Bit-slicing the Hilbert space: Scaling up accurate quantum circuit simulation. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 439–444. IEEE (2021). <https://doi.org/10.1109/DAC18074.2021.9586191>, <https://doi.org/10.1109/DAC18074.2021.9586191>
31. Wang, Q., Appel, A.W.: A solver for arrays with concatenation. *J. Autom. Reason.* **67**(1), 4 (2023). <https://doi.org/10.1007/s10817-022-09654-y>, <https://doi.org/10.1007/s10817-022-09654-y>
32. Xu, M., Fu, J., Mei, J., Deng, Y.: Model checking QCTL plus on quantum Markov chains. *Theor. Comput. Sci.* **913**, 43–72 (2022). <https://doi.org/10.1016/j.tcs.2022.01.044>, <https://doi.org/10.1016/j.tcs.2022.01.044>
33. Xu, M., Li, Z., Padon, O., Lin, S., Pointing, J., Hirth, A., Ma, H., Palsberg, J., Aiken, A., Acar, U.A., et al.: Quartz: superoptimization of quantum circuits. In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 625–640 (2022)
34. Yu, N., Palsberg, J.: Quantum abstract interpretation. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 542–558 (2021)
35. Zulehner, A., Wille, R.: Advanced simulation of quantum computations. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **38**(5), 848–859 (2019). <https://doi.org/10.1109/TCAD.2018.2834427>, <https://doi.org/10.1109/TCAD.2018.2834427>