Quantum Register Algebra: the mathematical language for quantum computing

J. Hrdina1 · D. Hildenbrand2 · A. Návrat1 · C. Steinmetz4 · R. Alves3 · C. Lavor5 · P. Vašík1 · I. Eryganov1

Received: 11 March 2023 / Accepted: 12 August 2023 / Published online: 31 August 2023
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract
We present Quantum Register Algebra (QRA) as an efficient tool for quantum computing. We show the direct link between QRA and Dirac formalism. We present Geometric Algebra Algorithms Optimizer (GAALOP) implementation of our approach. We demonstrate the ability to fully describe and compute with QRA in GAALOP using the geometric product.

Keywords Quantum computing · Geometric Algebra · Quantum Register Algebra

1 Introduction
In recent years, Geometric Algebra has proved to be a universal tool for different mathematical systems used conventionally in engineering and physics [5, 6, 12, 14, 18]. Geometric Algebra actually unifies the language of linear algebra, quaternions, Dirac and Pauli matrices, as well as Plücke coordinates in a single framework. This means that this single concept can be simply used in interdisciplinary applications. Another advantage, exploited in this paper, is that if it comes to object transformations, both operators and operands are handled as objects of the same algebra, which we found very useful in order to describe quantum computing (QC).

Recent papers show an increasing number of applications of geometric algebras to QC such as the use of Conformal Geometric Algebra (CGA) $\mathbb{G}_{4,1}$ (relativistic case) and $\mathbb{G}_{3,0}$ (non-relativistic case) [4], $\mathbb{G}_{n,n}$ [2], and also complex Clifford algebras as in [8, 9] to construct quantum circuits and describe qubit states, and in [3, 19] to fermionic quantum computation model.

Supported by FAPESP and CNPq and by the Grant No. FSI-S-23-8161.

✉ J. Hrdina
hrdina@fme.vutbr.cz

Extended author information available on the last page of the article
In the following, we present a Geometric Algebra named Quantum Register Algebra (QRA), which has a direct link to Dirac formalism as will be shown later. This paper is a direct follow-up to the article [10] in which the reader can find a detailed description of the correspondence between Dirac notation and geometric algebras. Some of the introductory didactic parts of this work were already published in [10] and presented at the conference ICACGA, 2022. Now, we offer a complete and more advanced paper which can also be used as a tutorial.

Our approach is based on the fact that QRA, i.e., Geometric Algebra $\mathbb{G}_{n+2} = \mathbb{G}_{n+2,0}$, together with the concept of the Witt basis provide a convenient language for writing QC schemes [9]. The expressions are very similar to the Dirac formalism, so they are easy to understand for people who are not familiar with the Geometric Algebra framework. However, compared to abstract Dirac formalism, our quantum realm objects are represented by elements of QRA, being a Geometric Algebra based on simple axioms. Moreover, QRA enables us to perform straightforward computations based on a few intuitive rules.

We are using Geometric Algebra algorithms optimizer (GAALOP) [1, 13] for calculations in QRA. We extend this tool to support QRA as a native algebra based on the definitions presented below. The basic literature that we use to derive our operators and states from conventional Dirac formalism is given, for example, in the books [7, 16].

The paper is structured as follows. In Sect. 2, we see how the complex numbers can be easily identified with Geometric Algebra, which is important in the definition of QRA. In Sect. 3, we briefly present Dirac formalism before we formally define QRA as a subalgebra of $\mathbb{G}_n$ with complex coefficients and also give the identification between Dirac notation and QRA. In Sect. 4, the GAALOP tool is described comprehensively. We present examples for both Clifford and non-Clifford gates, using Swap and Toffoli to attest that QRA can be applied to arbitrary gates. We also provide a compact expression for the Toffoli gate with n-qubits. In Sect. 5, we give the application of QRA to serial and parallel gates. Section 6 presents the GAALOP examples and explains how the Mathematica software simplifies the operations involving tensor products. Before the conclusion in Sect. 8, Sect. 7 illustrates how QRA can be applied in representing generalized SWAP gates. Additionally, in appendix A, we explain how to implement QRA by manually creating the definitions of the basis vectors for the algebra and the Witt basis.

2 From complex numbers to geometric algebras and back again

The algebra of complex numbers $\mathbb{C}$ is an essential tool for quantum computing. Qubits are realized by vectors in a complex vector space $\mathbb{C}^{2^n}$ and quantum logic gates by matrices $2^n \times 2^n$ over complex numbers. In our approach, complex linear algebra is the language for quantum computing. More precisely, we employ the natural concept of complex geometric algebras with the complex numbers as one of their instances. Further, we investigate these concepts in a more detailed way.

Formally, Geometric Algebra $\mathbb{G}_{p,q}$ is an associative, distributive, and unitary algebra over the set of abstract elements $\{e_1, \ldots, e_n\}$ endowed with the following
identities:

\[ e_i e_j = -e_j e_i, \text{ where } i \neq j, \]
\[ e_i^2 = 1, \text{ where } i = 1, \ldots, p, \quad (1) \]
\[ e_i^2 = -1, \text{ where } i = p + 1, \ldots, n, \]

where \( e_i e_j \) stands for the algebra product (also known as the geometric product) of two basis elements. In computer science notation, we understand \( G_{p,q} \) as a vector space where vectors are built as words over the alphabet \( \{e_1, \ldots, e_n\} \), including an empty word, using the following equivalency. Two words are equivalent (they are different representations of the same object) if the first can be rewritten into the second and vice versa with the help of identities \((1)\) and distributivity and associativity properties.

For example, considering Geometric Algebra \( G_{1,1} \), we have the words over the alphabet \( \{e_1, e_2\} \) and identities \( e_1^2 = 1, \ e_2^2 = -1 \) together with anti-commutativity \( e_1 e_2 = -e_2 e_1 \), implying that the basis of the algebra is of the form

\[ \{1, e_1, e_2, e_1 e_2\}. \quad (2) \]

Hence, for example, the word \( e_2 e_1 e_2 e_1 e_2 e_2 \) can be rewritten to \(-e_1\) using the identities \((1)\) in the following way:

\[ e_2 e_1 e_2 e_1 e_2 e_2 = e_2 e_1 e_2 (1)(-1) = e_1 e_2 e_2 = e_1 (-1) = -e_1. \]

We can see that elements of Geometric Algebra \( G_{1,1} \) are linear combinations of the basis \((2)\) elements, i.e.,

\[ G_{1,1} = \{x_1 + x_2 e_1 + x_3 e_2 + x_4 e_1 e_2 | x_i \in \mathbb{R}\} \quad (3) \]

together with the multiplication given by identities \((1)\). For example,

\[
(e_1 + 2e_1 e_2)(1 + e_1 - 3e_2) = e_1 (1 + e_1 - 3e_2) + 2e_1 e_2 (1 + e_1 - 3e_2)
= e_1 + e_1 e_1 - 3e_1 e_2 + 2e_1 e_2 + 2e_1 e_2 e_1 - 6e_1 e_2 e_2
= e_1 + 1 - 3e_1 e_2 + 2e_1 e_2 - 2e_2 e_1 e_1 - 6e_1 (-1)
= 1 + e_1 - e_1 e_2 - 2e_2 + 6e_1
= 1 + 7e_1 - 2e_2 - e_1 e_2 \in G_{1,1}.
\]

In any Geometric Algebra \( G_{p,q} \) such that \( p > 1 \) or \( q > 1 \), we can find a subalgebra isomorphically equivalent to \( \mathbb{C} \) in the following way: if \( p > 1 \), we have two elements \( e, f \in G_{p,q} \) such that \( e^2 = f^2 = 1 \) and \( ef = -fe \), implying that

\[
(ef)^2 = ef ef = -e e f f = -e^2 f^2 = -1
\]

and

\[ \mathbb{C} \cong \tilde{\mathbb{C}} = \{a + b(ef) | a, b \in \mathbb{R}\} \subset G_{p,q}. \]
In the same way, if we have \( q > 1 \), we obtain two elements \( e, f \) such that
\[
e^2 = f^2 = -1 \quad \text{and} \quad ef = -fe,
\]
which implies that
\[
(ef)^2 = efe = -efef = -e^2f^2 = -1(-1)(-1) = -1.
\]
Note that the element \( ef \) commutes with the other basis elements in both cases.

**Remark 1** The complex numbers \( \mathbb{C} \) are in fact isomorphically equivalent to Geometric Algebra \( \mathbb{G}_{0,1} \), where
\[
\mathbb{G}_{0,1} = \{ x_1 + x_2e_1 | x_i \in \mathbb{R} \}, \tag{4}
\]
such that \( e_1^2 = -1 \). The specific concept of the imaginary unit used in QRA is described in Sect. 3.

### 3 Quantum computing in the QRA framework

#### 3.1 Matrices versus Dirac formalism

In this section, we show the direct link between Dirac formalism and Geometric Algebra. Note that quantum computing can be realized in matrices because the Hilbert space of qubit states is of finite dimension. We briefly recall the link between matrices and Dirac formalism [7]:

- the \( n \)-qubit (ket)

\[
|i\rangle = |a_1 \ldots a_{n-1}a_n\rangle, \quad \text{where} \quad a_j \in \{0, 1\}, \ j = 1, \ldots, n, \ \text{and}
\]
\[
i = a_12^{n-1} + \cdots + a_n2^0 \iff \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \text{where} \ 1 \ \text{is on position} \ i + 1
\]

and 0 otherwise.

- the dual \( n \)-qubit (bra)

\[
\langle i | = \langle a_1 \ldots a_{n-1}a_n |, \quad \text{where} \quad a_j \in \{0, 1\}, \ j = 1, \ldots, n, \ \text{and}
\]
\[
i = a_12^{n-1} + \cdots + a_n2^0 \iff (0 \cdots 1 \cdots 0), \quad \text{where} \ 1 \ \text{is on position} \ i+1
\]
and 0 otherwise.

An \( n \)-qubit gate is a matrix \( A = (a_{ij}) \), where \( i, j = 1, \ldots, 2^n \). If we consider a canonical basis of \( \mathbb{R}^{2^n} \), then the matrix \( A \) acts on the basis vectors in such a way that
its element $a_{ij}$ assigns the $j$th element of canonical basis to $i$th element of canonical basis in the same way as Dirac expression $|i - 1\rangle\langle j - 1|$. In other words,

$$
\begin{pmatrix}
   a_{11} & \cdots & a_{1n} \\
   \vdots & \ddots & \vdots \\
   a_{n1} & \cdots & a_{nn}
\end{pmatrix} \iff \sum_{i,j=1}^n a_{ij} |i - 1\rangle\langle j - 1|.
$$

For example, the representations of 1-qubits (ket and bra) are

$|0\rangle \iff \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $|1\rangle \iff \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $\langle 0 | \iff \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\langle 1 | \iff \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

and NOT gate is of the form

$$
\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \iff |0\rangle\langle 1| + |1\rangle\langle 0|.
$$

Similarly, in the case of 2-qubits, we have

$|00\rangle \iff \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, $|01\rangle \iff \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, $|10\rangle \iff \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$, $|11\rangle \iff \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$,

$\langle 00 | \iff \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$, $\langle 01 | \iff \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$, $\langle 01 | \iff \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$, $\langle 11 | \iff \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$,

implying that, for example, CNOT gate is of the form

$$
\begin{pmatrix}
   1 & 0 & 0 & 0 \\
   0 & 1 & 0 & 0 \\
   0 & 0 & 0 & 1 \\
   0 & 0 & 1 & 0
\end{pmatrix} \iff |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|.
$$

3.2 Definition of QRA

Recall that a Geometric Algebra $\mathbb{G}_n = \mathbb{G}_{n,0}$ is a free, associative, unitary algebra over the set of anti-commuting generators $\{e_1, \ldots, e_n\}$, such that $e_i^2 = 1$, $i \in \{1, \ldots, n\}$. Now, we can define the QRA. First, let us consider a Geometric Algebra $\mathbb{G}_{n+2}$ with its generating elements

$$
\{e_1, \ldots, e_n, r_1, r_2\}
$$

together with the following identities

$$
e_1^2 = e_2^2 = \cdots = e_n^2 = r_1^2 = r_2^2 = 1.
$$
Then, we define a bivector $\iota$ as
\[ \iota = r_1 r_2 \]
implying that the set $\tilde{\mathbb{C}} = \{a + b\iota | a, b \in \mathbb{R}\}$ is isomorphic to the algebra $\mathbb{C}$ with $\iota$ playing the role of a complex unit. The set $\tilde{\mathbb{C}}$ is closed with respect to the addition and multiplication. The element $\iota$ is in square equal to $-1$. Indeed,
\[ \iota^2 = r_1 r_2 r_1 r_2 = -r_1^2 r_2^2 = -1, \]
so $\tilde{\mathbb{C}} \cong \mathbb{C}$. Next, we define $\text{QRA}(n)$ as a geometric subalgebra $\mathbb{G}_n$ with the coefficients in $\tilde{\mathbb{C}}$, i.e.,
\[ \text{QRA}(n) = \{a_0 g_0 + \cdots + a_n g_n | a_i \in \tilde{\mathbb{C}}, g_i \in \mathbb{G}_n\} \subset \mathbb{G}_{n+2}. \]
Note that, for any element $g \in \mathbb{G}_n$, we have $\iota g = g \iota$, because $\iota$ is a bivector and $\iota \notin \mathbb{G}_n$.

### 3.3 Relation to Dirac formalism

To use $\text{QRA}$ to model quantum computing, we choose a different basis of $\text{QRA}(n)$ based on the Geometric Algebra $\mathbb{G}_{2n+2}$. This basis is called the Witt basis and it is formed by elements \{ $f_1, f_1^\dagger, \ldots, f_n, f_n^\dagger$ \} satisfying
\begin{align*}
    f_i & = \frac{1}{2}(e_i + \iota e_{i+n}), \quad (7) \\
    f_i^\dagger & = \frac{1}{2}(e_i - \iota e_{i+n}), \quad (8)
\end{align*}
where $\iota = r_1 r_2$. The relation between Clifford and Witt algebras can be found in [15]. The Witt basis can also be used as generators of Clifford algebras to describe the creation and annihilation of operators in quantum mechanics, see [17].

Now, we define an element $I = f_1 f_1^\dagger \cdots f_n f_n^\dagger$. This element satisfies
\begin{align*}
    I^2 & = I, \quad (9) \\
    f_i I & = 0, \quad (10) \\
    f_i f_i^\dagger I & = I, \quad (11)
\end{align*}
which implies that we have a straightforward identification of bra and ket vectors with the elements of $\text{QRA}$ as follows:
\begin{align*}
    \langle a_1 \ldots a_n \rangle & \leftrightarrow I (f_n)^{a_n} \cdots (f_1)^{a_1}, \text{ where } a_i \in \{0, 1\}, \quad (12) \\
    | a_1 \ldots a_n \rangle & \leftrightarrow (f_1^\dagger)^{a_1} \cdots (f_n^\dagger)^{a_n} I, \text{ where } a_i \in \{0, 1\}. \quad (13)
\end{align*}
To show that the (12) and (13) correspondences are correctly defined we prove their following properties

- If \(|a_1 \ldots a_n\rangle\) is a basis then \(<a_1 \ldots a_n|\) is a dual basis. This is based on the straightforward calculation based on identities (20), (21) and (22):

\[
I(f_n)^a_n \ldots (f_1)^a_1 (f_1^\dagger)^b_1 \ldots (f_n^\dagger)^b_n I = 0 \quad \text{if } (a_1 = 1, b_1 = 0) \text{or } (a_1 = 0, b_1 = 1)
\]

If \((a_1 = 1, b_1 = 1)\) or \((a_1 = 0, b_1 = 0)\) we receive

\[
I(f_n)^a_n \ldots (f_1)^a_1 (f_1^\dagger)^a_1 \ldots (f_n^\dagger)^a_n I = I(f_n)^a_n \ldots (f_1)^a_2 (f_1^\dagger)^a_2 \ldots (f_n^\dagger)^a_n I
\]

and the same for \(a_2, b_2, \text{ etc.}\)

- The composition of operators is an operator again. This property is based on the following observation. Each operator can be written as a linear combination of the vectors \(A_i\) and the covectors \(B_j\) as the sum \(\sum_i, j A_j B_j\). Since the vectors and covectors represented in QRA algebra form dual basis (as we proved before), their composition must again be a combination of the vectors the covectors.

For example, let us consider a space of 2-qubit states. Identification (12) for the ket vectors reads

\[
|00\rangle \mapsto (f_1^\dagger)^0 (f_2^\dagger)^0 I = I,
|01\rangle \mapsto (f_1^\dagger)^0 (f_2^\dagger)^1 I = f_2^\dagger I,
|10\rangle \mapsto (f_1^\dagger)^1 (f_2^\dagger)^0 I = f_1^\dagger I,
|11\rangle \mapsto (f_1^\dagger)^1 (f_2^\dagger)^1 I = f_1^\dagger f_2^\dagger I,
\]

implying that the ket vectors in 2-qubit state space are linear combinations of the basis elements \(\{f_1, f_1^\dagger, \ldots, f_n, f_n^\dagger\}\):

\[
|\psi\rangle = (\psi_{00} + \psi_{10} f_1^\dagger - \psi_{11} f_1^\dagger f_2^\dagger) I.
\]

To define quantum gates, we have to identify the bra vectors in a similar way in accordance with (13):

\[
\langle 00| \mapsto I(f_2)^0 (f_1)^0 = I,
\langle 01| \mapsto I(f_2)^0 (f_1^\dagger)^0 = I f_2,
\langle 10| \mapsto I(f_2)^0 (f_1^\dagger)^1 = I f_1,
\langle 11| \mapsto I(f_2)^1 (f_1^\dagger)^1 = -I f_1 f_2,
\]

which implies that the bra vectors in 2-qubit state space are combinations of the Witt basis elements \(\{f_1, f_1^\dagger, \ldots, f_n, f_n^\dagger\}\):

\[
(\psi| = I(\psi_{00} + \psi_{10} f_1 + \psi_{01} f_2 - \psi_{11} f_1 f_2).
\]
To illustrate our approach, we show a design of SWAP gate [16]. Recall that in Dirac notation, SWAP gate is represented by

$$|00\rangle |00\rangle + |01\rangle |10\rangle + |10\rangle |01\rangle + |11\rangle |11\rangle.$$ 

Using identifications (14) and (16), the SWAP gate may be rewritten as

$$\text{SWAP} = |00\rangle |00\rangle + |01\rangle |10\rangle + |10\rangle |01\rangle + |11\rangle |11\rangle = f_1 f_2 f_1 f_2 + f_1 f_2 f_1 f_2$$

(17)

and the T gate (a non-Clifford gate) may be rewritten as

$$T = |00\rangle |00\rangle + e^{-i\frac{\pi}{4}} |11\rangle |11\rangle = f_1 f_2 f_1 f_2 + e^{-i\frac{\pi}{4}} f_1 f_2$$

(18)

In order to present how the SWAP gate acts on 2-qubits, let us mention the rules for calculations with the Witt basis \(\{f_i, f_i^\dagger\}\) elements in the form of a list of properties which can be verified by straightforward computations:

$$\left(f_i\right)^2 = \left(f_i^\dagger\right)^2 = 0,$$

(20)

$$f_i f_j = -f_j f_i, \quad f_i^\dagger f_j^\dagger = -f_j^\dagger f_i^\dagger,$$

(21)

$$f_i f_j^\dagger f_i = f_i, \quad f_i^\dagger f_i f_i^\dagger = f_i^\dagger.$$

(22)

Thus, we obtain the SWAP gate functionality on 2-qubits step by step:

$$\text{SWAP} |\psi\rangle = (f_1 f_2 f_2 f_1 + f_1^\dagger f_2 f_2^\dagger + f_1^\dagger f_1 f_2^\dagger f_2^\dagger) (\psi_00 + \psi_01 f_2^\dagger + \psi_10 f_1^\dagger + \psi_11 f_1^\dagger f_2^\dagger) I$$

$$= f_1 f_2 f_2 f_1 (\psi_00 + \psi_01 f_2^\dagger + \psi_10 f_1^\dagger + \psi_11 f_1^\dagger f_2^\dagger) I \text{ by (20)}$$

$$+ f_2 f_1 (\psi_00 + \psi_01 f_2^\dagger + \psi_10 f_1^\dagger + \psi_11 f_1^\dagger f_2^\dagger) I \text{ by (10) and (20)}$$

$$- f_1 f_2 f_2 f_1 (\psi_00 + \psi_01 f_2^\dagger + \psi_10 f_1^\dagger + \psi_11 f_1^\dagger f_2^\dagger) I \text{ by (10) and (20)}$$

$$+ f_2 f_1 f_2 f_1 (\psi_00 + \psi_01 f_2^\dagger + \psi_10 f_1^\dagger + \psi_11 f_1^\dagger f_2^\dagger) I \text{ by (10) and (20)}$$

$$= (f_1 f_2 f_2 f_1) (\psi_00) I + (f_2 f_1) (\psi_01 f_2^\dagger) I \text{ by (11)}$$

$$+ f_2 f_1 \psi_10 f_1^\dagger I + (f_1 f_2 f_2) (\psi_11 f_1^\dagger f_2^\dagger) I \text{ by the last step by (11)}$$

$$= \psi_00 + \psi_01 f_1^\dagger + \psi_10 f_2^\dagger + \psi_11 (f_1^\dagger f_1^\dagger f_2 f_2^\dagger + f_2 f_2^\dagger f_1^\dagger f_2^\dagger) I.$$
where \( g \) and \( \overline{g} \) stand for \( g = 0 \) and \( g = 1 \), respectively. Finally, by means of (14), the final 2-qubit can be rewritten in Dirac notation as

\[
(\psi_{00}|00\rangle + \psi_{01}|10\rangle + \psi_{10}|01\rangle + \psi_{11}|11\rangle),
\]

which is the expected result.

The other gates may be interpreted in the very same way. Thus, we have explained how quantum computation is realized in QRA using the Witt basis axioms (20)–(22), together with the additional axioms (10) and (11). The use of the additional axioms may seem redundant and complicated, but they only simplify the written form of calculations making the functionality easier for demonstration. Indeed, whereas these rules help to avoid unnecessary prolongation of expressions, they may be completely ignored due to the two following reasons:

- If we interpret the element \( I \) as \( f_1 f_1^\dagger \cdots f_n f_n^\dagger \), the rules (10) and (11) do not apply since they are direct consequences of (20)–(22).
- We use the axioms (20)–(22) for calculations in the Witt basis which naturally corresponds to Dirac formalism. But the Witt basis is just a different set of generators for QRA elements. Thus, the axioms (20)–(22) are derived from Geometric Algebra axioms (6).

Now, from (5), we present one more illustrative example, considering the CNOT gate

\[
|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|.
\]

(23)

Using the identifications (14) and (16), we rewrite the expression (23) in QRA as:

\[
\begin{align*}
\text{CNOT} &= f_1 f_1^\dagger f_2 f_2^\dagger + f_2^\dagger I f_2 - f_1^\dagger I f_1 f_2 + f_1^\dagger f_2 f_1 f_2^\dagger I f_1 \\
&= f_1 f_1^\dagger f_2 f_2^\dagger + f_2^\dagger f_1 f_1^\dagger f_2 - f_1^\dagger f_1 f_2 + f_1^\dagger f_2^\dagger f_2 f_2^\dagger f_1 \\
&= f_1 f_1^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2 f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger.
\end{align*}
\]

(24)

Note that the element (24) acts on a 2-qubit state \(|\psi\rangle = (\psi_{00} + \psi_{01} f_2^\dagger + \psi_{10} f_1^\dagger + \psi_{11} f_1^\dagger f_2^\dagger) I\) as follows:

\[
\begin{align*}
\text{CNOT}|\psi\rangle &= (f_1 f_1^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2 f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger) I \\
&= (\psi_{00} + \psi_{01} f_2^\dagger + \psi_{10} f_1^\dagger + \psi_{11} f_1^\dagger f_2^\dagger) I \\
&= (f_1 f_1^\dagger f_2 f_2^\dagger)(\psi_{00}) + (f_1 f_1^\dagger f_2 f_2^\dagger)(\psi_{01} f_2^\dagger) \\
&- (f_1 f_1^\dagger f_2 f_2^\dagger)(\psi_{10} f_1^\dagger) - (f_1^\dagger f_1 f_2)(\psi_{11} f_1^\dagger) \\
&= \psi_{00} + f_2^\dagger \psi_{01} + f_1^\dagger \psi_{11} + f_1^\dagger f_2^\dagger \psi_{10}.
\end{align*}
\]

Our approach is based on the fact that the QRA and the Witt basis provide convenient language for written QC schemes. Since the expressions are very similar to Dirac
formalism, they are easy to understand for people who are not familiar with Geometric Algebra. However, unlike abstract Dirac formalism, our objects are elements of the QRA, i.e., we have a Geometric Algebra based on simple axioms. Furthermore, it is intuitive in implementation as will be shown in the following section.

4 GAALOP implementation

The GAALOP\(^1\) is a free software tool developed to optimize Geometric Algebra algorithms and cut the high complexity of Geometric Algebra before going to the real computing device. This is done by precomputing / precompiling Geometric Algebra algorithms as described in the book "Foundations of Geometric Algebra Computing" [11]. GAALOP generates optimized source code for programming languages such as C/C++, C++ AMP, OpenCL and CUDA as well as Python, Julia, MATLAB and Mathematica. For our purposes, we use the code generator for Latex code in order to directly integrate result into this article. If your GAALOP implementation does not include the QRA, please refer to appendix A for implementation details.

Currently, there is a new version of GAALOPWeb\(^2\) available. It allows online computations with an $n$-qubits without the installation of a specific software.

In order to show examples for both Clifford and non-Clifford gates, using GAALOP, we implemented the SWAP gate for a register with two qubits and the Toffoli gate for 3 and 5 qubits (please refer to Fig. 1 for a screenshot of GAALOP indicating the selection of QRA and the number of 2 qubits). Following (7) and (8), vectors $f_i$ and $f_i^\dagger$ are given by

$$f_1 = \frac{1}{2}(e_1 + i e_3), \quad f_1^\dagger = \frac{1}{2}(e_1 - i e_3),$$

$$f_2 = \frac{1}{2}(e_2 + i e_4), \quad f_2^\dagger = \frac{1}{2}(e_2 + i e_4),$$

\(^1\) download via http://www.gaalop.de.
\(^2\) http://www.gaalop.de/gaalopweb/.
where $i = r_1 r_2$. We use the definitions in (14) to implement the basis elements, identification (15) for $|\psi\rangle$, and (18) to define the operator of SWAP gate.

Now, we proceed to the application of SWAP operator on an element $|\psi\rangle$. But first, let us see the basis coordinates of ket vectors when they are multiplied by $I$. Listing 1 shows the corresponding GAALOPScript code for "code to optimize" field of GAALOPWeb.\(^3\)

Listing 1 SWAP gate in QRA for two qubits.

```plaintext
// Imaginary unit
i = er1*er2;

// Witt basis
f1 = 0.5*(e1+i*e3);
f1T = 0.5*(e1-i*e3);
f2 = 0.5*(e2+i*e4);
f2T = 0.5*(e2-i*e4);

// Element "I"
Id = f1*f1T*f2*f2T;

// ket basis vectors multiplied by "Id"
ket00 = 1*Id;
ket01 = f2T*Id;
ket10 = f1T*Id;
ket11 = f1T*f2T*Id;
```

It is important to show which coordinates correspond to each vector in order to see an interchanging of amplitudes when SWAP is applied to some linear combination of these vectors. The output for this code is shown below:

Listing 2 Basic elements multiplied by $I$.

```plaintext
ket00[0] = 0.25; // 1.0
ket00[42] = 0.25; // e1 ^ (e2 ^ (e3 ^ e4))
ket00[50] = -0.25; // e1 ^ (e3 ^ (er1 ^ er2))
ket00[55] = -0.25; // e2 ^ (e4 ^ (er1 ^ er2))
ket01[2] = 0.25; // e1
ket01[26] = -0.25; // e1 ^ (e3 ^ e4)
ket01[41] = -0.25; // e4 ^ (er1 ^ er2)
ket01[59] = 0.25; // e1 ^ (e2 ^ (e3 ^ (er1 ^ er2)))
ket10[1] = 0.25; // e1
ket10[32] = 0.25; // e2 ^ (e3 ^ e4)
ket10[40] = -0.25; // e3 ^ (er1 ^ er2)
ket10[60] = -0.25; // e1 ^ (e2 ^ (e4 ^ (er1 ^ er2)))
ket11[7] = 0.25; // e1 ^ e2
ket11[16] = -0.25; // e3 ^ e4
ket11[51] = -0.25; // e1 ^ (e4 ^ (er1 ^ er2))
ket11[54] = 0.25; // e2 ^ (e3 ^ (er1 ^ er2))
```

\(^3\) See Fig. 1.
Note that all identities given in (11)–(20) can be observed in GAALOP. Let us choose a vector $|\psi\rangle$ given by

$$|\psi\rangle = (|00\rangle + 2|01\rangle + 3|10\rangle + 4|11\rangle)I = (1 + 2f_2^\dagger + 3f_1^\dagger + 4f_1^\dagger f_2^\dagger)I,$$

and apply the SWAP gate on it. The expected result is given by:

$$|\bar{\psi}\rangle = (1 + 3f_2^\dagger + 2f_1^\dagger + 4f_1^\dagger f_2^\dagger)I = (|00\rangle + 3|01\rangle + 2|10\rangle + 4|11\rangle)I.$$

Now, we add 3 lines to the code in Listing 1.

**Listing 3** Definition and application of the SWAP gate on GAALOP.

1 //SWAP
2 SWAP=(f1*f1T*f2*f2T)+(f1T*f2)-(f1*f2T)+(f1T*f1*f2T*f2);
3 ?psi = ket00 + 2*ket01 + 3*ket10 + 4*ket11;
4 ?SwapPsi = SWAP*psi;

The output is shown below.

**Listing 4** Output of the application of the SWAP gate on GAALOP.

1 psi[0] = 0.25; // 1.0
2 psi[1] = 0.75; // e1
3 psi[2] = 0.5; // e2
4 psi[7] = 1.0; // e1 ^ e2
5 psi[16] = -1.0; // e3 ^ e4
6 psi[26] = -0.5; // e1 ^ (e3 ^ e4)
7 psi[32] = 0.75; // e2 ^ (e3 ^ e4)
8 psi[40] = -0.75; // e3 ^ (e1 ^ e2)
9 psi[41] = -0.5; // e4 ^ (e1 ^ e2)
10 psi[42] = 0.25; // e1 ^ (e2 ^ (e3 ^ e4))
11 psi[50] = -0.25; // e1 ^ (e3 ^ (e1 ^ e2))
12 psi[51] = -1.0; // e1 ^ (e4 ^ (e1 ^ e2))
13 psi[54] = 1.0; // e2 ^ (e3 ^ (e1 ^ e2))
14 psi[55] = -0.25; // e2 ^ (e4 ^ (e1 ^ e2))
15 psi[59] = 0.5; // e1 ^ (e2 ^ (e3 ^ (e1 ^ e2)))
16 psi[60] = -0.75; // e1 ^ (e2 ^ (e4 ^ (e1 ^ e2)))
17 SwapPsi[0] = 0.25; // 1.0
18 SwapPsi[1] = 0.5; // e1
19 SwapPsi[2] = 0.75; // e2
20 SwapPsi[7] = 1.0; // e1 ^ e2
21 SwapPsi[16] = -1.0; // e3 ^ e4
22 SwapPsi[26] = -0.75; // e1 ^ (e3 ^ e4)
23 SwapPsi[32] = 0.5; // e2 ^ (e3 ^ e4)
24 SwapPsi[40] = -0.5; // e3 ^ (e1 ^ e2)
25 SwapPsi[41] = -0.75; // e4 ^ (e1 ^ e2)
26 SwapPsi[42] = 0.25; // e1 ^ (e2 ^ (e3 ^ e4))
27 SwapPsi[50] = -0.25; // e1 ^ (e3 ^ (e1 ^ e2))
28 SwapPsi[51] = -1.0; // e1 ^ (e4 ^ (e1 ^ e2))
29 SwapPsi[54] = 1.0; // e2 ^ (e3 ^ (e1 ^ e2))
30 SwapPsi[55] = -0.25; // e2 ^ (e4 ^ (e1 ^ e2))

4 The vector $|\psi\rangle$ is not normalized, so the final result should be divided by $\sqrt{29}$.  

≧ Springer
31 \text{SwapPsi}[59] = 0.75; // e1 \ ^ \ (e2 \ ^ \ (e3 \ ^ \ (er1 \ ^ \ er2)))
32 \text{SwapPsi}[60] = -0.5; // e1 \ ^ \ (e2 \ ^ \ (e4 \ ^ \ (er1 \ ^ \ er2)))

Note that the groups of coordinates \{0, 42, 50, 55\} and \{7, 16, 51, 54\}, which refer to the coefficients of \ket{00} and \ket{11}, respectively, remained unchanged from \psi to SwapPsi, as expected. However, note that the coefficients in the coordinates \{2, 26, 41, 59\} of \psi that correspond to \ket{01} were exactly 2 times the coefficients of ket01 given by Listing 2 and, in SwapPsi, they became 3 times those same coefficients. On the other hand, the coordinates \{1, 32, 40, 60\} changed from 3 (in psi) to 2 (in SwapPsi) times the coefficients of ket10 in Listing 2, that corresponds to \ket{10}. So, the output vector SwapPsi is exactly vector \ket{\bar{\psi}}.

4.1 The Toffoli gate

In this section, we will show how an arbitrary gate can be easily converted to the QRA. From the resulting representation, it will be clear that, especially in the case of sparse matrices, our description is effective. We start with the classical case of the Toffoli gate, which is not a Clifford one. It is a 3-qubit gate, represented as a block matrix

$$\text{Toffoli} = \begin{pmatrix} E & 0 \\ 0^T & A \end{pmatrix},$$

where 0 is a zero $6 \times 2$ matrix, $E_{6\times6}$ is the $6 \times 6$ unit matrix, and $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. In Dirac notation (as in the QRA notation), we write only the non-zero elements of the matrix, implying that we get the sum of the diagonal part and the matrix $A$. In Dirac notation, we have

$$\text{Toffoli} = \ket{000}\bra{000} + \ket{001}\bra{001} + \ket{010}\bra{010} + \ket{011}\bra{011} + \ket{100}\bra{100} + \ket{101}\bra{101} + \ket{110}\bra{110} + \ket{111}\bra{111}. \quad (25)$$

To find the QRA representation, we need the representation of the ket and bra vectors on the 3-qubit space, given as follows,

$$\langle a_1a_2a_3 | \leftrightarrow I(f_3)^{a_3}(f_2)^{a_2}(f_1)^{a_1}, \quad \text{where} \ a_i \in \{0, 1\}$$

$$|a_1a_2a_3 \rangle \leftrightarrow (f_1)^{a_1}(f_2)^{a_2}(f_3)^{a_3}I,$$

which implies that the Toffoli gate is represented in the QRA by

$$\text{Toffoli} = I + f_3^\dagger f_3 + f_2^\dagger f_2 + f_3^\dagger f_3^\dagger f_3^2f_2 + f_1^\dagger I f_1 + f_1^\dagger f_3^\dagger f_3 f_1$$
$$+ f_1^\dagger f_2^\dagger f_3^2 f_2 f_1 + f_1^\dagger f_2^\dagger f_3 f_2^\dagger f_1 + f_1^\dagger f_3 f_3^\dagger f_3 f_1$$
$$= I + f_3^\dagger (f_1^\dagger f_1^\dagger f_2^\dagger f_3 f_1) f_3 + f_3^\dagger (f_1^\dagger f_1^\dagger f_3 f_3^\dagger f_3 f_2) + f_2^\dagger f_3^\dagger (f_1^\dagger f_1^\dagger f_3 f_3) f_3 f_2$$
$$+ f_3^\dagger (f_2^\dagger f_2^\dagger f_3 f_3^\dagger f_3 f_1) f_3 f_1 + f_1^\dagger f_3^\dagger (f_2^\dagger f_2^\dagger f_3 f_3^\dagger f_3 f_2) f_1 + f_1^\dagger f_2^\dagger f_3^\dagger f_3^\dagger f_3 f_2.$$  

(26)
Rewriting the first part, we get a fairly compact notation (see Listing 5 for GAALOP code):

$$\text{Toffoli} = \sum_{S \subseteq \{1...3\}} \prod_{i \notin S} f_i f_i^\dagger \prod_{i \in S} f_i^\dagger f_i + \sum_{i=1}^3 f_i f_i^\dagger f_1 f_2 f_3 f_1 + f_1^\dagger f_2^\dagger f_3^\dagger f_2 f_2^\dagger,$$  \hspace{1cm} (27)

The advantage of the QRA notation is that it can be easily generalized to a space with more qubits. For example, considering the generalized Toffoli gate on the space of general n-qubits, we obtain

$$\text{Toffoli}_n = \sum_{S \subseteq \{1...n\}} \prod_{i \notin S} f_i f_i^\dagger \prod_{i \in S} f_i^\dagger f_i + \prod_{j=1}^{n-1} f_j^\dagger \prod_{j=n}^n f_j + \prod_{j=1}^{n-1} f_j \prod_{j=n}^n f_j.$$ \hspace{1cm} (28)

We can see that the QRA apparatus not only yields a compatible expression of gates corresponding to the sparse matrix. Because it is an algebra, it allows the use of algebraic properties. Finally, we have the GAALOP code for the 5-qubits register in Listing 5.

**Listing 5** Definitions and tests for Toffoli operations on 3 qubits registers.

```
1 // Witt basis definitions
2 i = er1 * er2 ;
3 f1 = 0.5*( e1 + i * e4 );
4 f1T = 0.5*( e1 - i * e4 );
5 f2 = 0.5*( e2 + i * e5 );
6 f2T = 0.5*( e2 - i * e5 );
7 f3 = 0.5*( e3 + i * e6 );
8 f3T = 0.5*( e3 - i * e6 );
9 Id = f1 * f1T * f2 * f2T * f3 * f3T;
10
11 // QRA expression for the Toffoli operator
12 Toff=Id+f3T*(f1*f1T*f2*f2T)*f3 + f2T*(f1*f1T*f3*f3T)*f2 + f2T*f3T*(f1*f1T)*f3*f2 + f1T*(f2*f2T*f3*f3T)*f1 + f1T*f3T*(f2*f2T)*f3*f1 + f1T*f2T*f3T*f2*f1 + f1T*f2T*f3T*f2*f1;
13
14 // basis vectors
15 psi0=Id; // |000>
16 psi1=f3T*Id; // |001>
17 psi2=f2T*Id; // |010>
18 psi3=f1T*Id; // |011>
19 psi4=f1T*f2T*Id; // |110>
20 psi5=f1T*f3T*Id; // |101>
21 psi6=f2T*f3T*Id; // |011>
22 psi7=f1T*f2T*f3T*Id; // |111>
23
24 // Application on each basis vectors;
25 X0=Toff*psi0; // expected identity
26 X1=Toff*psi1; // expected identity
27 X2=Toff*psi2; // expected identity
28 X3=Toff*psi3; // expected identity
29 X4=Toff*psi4; // expected psi7
```

\[\text{Springer}\]
Gaalop returns an empty file, since all the check results are 0.

Listing 6 Definitions and particular test for Toffoli operations on 5 qubits registers.

1 //Witt basis definitions
2 f1 = 0.5* ( e1 + i * e6 );
3 f1T = 0.5*( e1 - i * e6 );
4 f2 = 0.5* ( e2 + i * e7 );
5 f2T = 0.5*( e2 - i * e7 );
6 f3 = 0.5* ( e3 + i * e8 );
7 f3T = 0.5*( e3 - i * e8 );
8 f4 = 0.5* ( e4 + i * e9 );
9 f4T = 0.5*( e4 - i * e9 );
10 f5 = 0.5* ( e5 + i * e10 );
11 f5T = 0.5*( e5 - i * e10 );
12 Id = f1 * f1T * f2 * f2T * f3 * f3T * f4 * f4T * f5 * f5T ;
13
14 //QRA expression for the Toffoli operator
15 Toff5= Id + f1T*Id*f1 + f2T*Id*f2 + f1T*f2T*Id*f2*f1
16 + f3T*Id*f3 + f1T*f3T*Id*f3*f1 + f2T*f3T*Id*f3*f2
17 + f1T*f2T*f3T*Id*f3*f2*f1 + f4T*Id*f4 + f1T*f4T*Id*f4*f1
18 + f2T*f4T*Id*f4*f2 + f1T*f2T*f4T*Id*f4*f2*f1
19 + f3T*f4T*Id*f4*f3 + f2T*f3T*f4T*Id*f4*f3*f2
20 + f1T*f3T*f4T*Id*f4*f3*f1 + f1T*f2T*f3T*f4T*Id*f5*f4*f3*f2*f1
21 + f5T*Id*f5 + f1T*f5T*Id*f5*f1 + f2T*f5T*Id*f5*f2
22 + f1T*f2T*f5T*Id*f5*f2*f1 + f3T*f5T*Id*f5*f3
23 + f1T*f3T*f5T*Id*f5*f3*f1 + f2T*f3T*f5T*Id*f5*f3*f2
24 + f1T*f2T*f3T*f5T*Id*f5*f3*f2*f1 + f4T*f5T*Id*f5*f4
25 + f1T*f4T*f5T*Id*f5*f4*f1 + f2T*f4T*f5T*Id*f5*f4*f2
26 + f1T*f2T*f4T*f5T*Id*f5*f4*f2*f1 + f3T*f4T*f5T*Id*f5*f4*f3
27 + f2T*f3T*f4T*f5T*Id*f5*f4*f3*f2 + f1T*f3T*f4T*f5T*Id*f5*f4*f3*f1
28 + f1T*f2T*f3T*f4T*f5T*Id*f5*f4*f3*f2*f1;
29
30 //checking results with the expected outputs
31 psi0=f1T*f2T*f3T*f4T*Id; //|11110>
32 psi1=f1T*f2T*f3T*f4T*f5T*Id; //|11111>
33 X0=Toff5*psi0;
34 X1=Toff5*psi1;
35 ?checkX0=psi0 - X0;
36 ?checkX1=psi1 - X1;
37

Gaalop returns an empty file, since all the check results are 0. We showed the tests only for the registers that change the last qubit, but for any other basis vectors, the operator return the same vector, as for the 3-qubits example above.
5 Parallel and serial gates

5.1 Serial gates

Let \( A \) be an \( n \)-qubit gate represented by QRA element \( f_A \) and \( B \) be an \( n \)-qubit gate represented by QRA element \( f_B \).

So, the serial circuit depicted in Fig. 2 can be seen as an \( n \)-qubit gate with its QRA representation being a straightforward multiplication of the corresponding QRA elements \( f_B f_A \). We illustrate the construction of serial gates, using the example \( A = B = NOT \), with the following calculations:

\[
\text{NOT} \circ \text{NOT} = (f_2^\dagger f_1 + f_1^\dagger f_2 + f_1^\dagger f_2^\dagger + f_2 f_1)(f_2^\dagger f_1 + f_1^\dagger f_2 + f_1^\dagger f_2^\dagger + f_2 f_1)
\]
\[
= (f_2^\dagger f_1 f_2^\dagger f_1 + f_2^\dagger f_1 f_2 f_1 + f_2^\dagger f_1 f_2^\dagger f_2 + f_2^\dagger f_1 f_2 f_1)
\]
\[
+ f_2 f_1 f_2^\dagger f_1 + f_2 f_1 f_2 f_1 + f_2 f_1 f_2^\dagger f_2 + f_2 f_1 f_2 f_1
\]
\[
+ f_1 f_2^\dagger f_2^\dagger f_1 + f_1 f_2^\dagger f_2 f_1 + f_1 f_2^\dagger f_2^\dagger f_2 + f_1 f_2^\dagger f_2 f_1
\]
\[
+ f_2 f_1 f_2^\dagger f_1 + f_2 f_1 f_2 f_1 + f_2 f_1 f_2^\dagger f_2 + f_2 f_1 f_2 f_1)
\]
\[
= (f_2^\dagger f_1 f_2^\dagger f_1 + f_1^\dagger f_2 f_2 f_1 + f_1^\dagger f_2 f_2 f_1 + f_2 f_1 f_2^\dagger f_2)
\]
\[
= (f_1 f_2^\dagger f_2^\dagger f_2 + f_1^\dagger f_1 f_2 f_2^\dagger + f_1^\dagger f_1 f_2 f_2 + f_1 f_2^\dagger f_2^\dagger f_2)
\]
\[
= (f_1 f_2^\dagger (f_2^\dagger f_2 + f_2^\dagger f_2) + f_1^\dagger f_1 (f_2 f_2^\dagger + f_2^\dagger f_2))
\]
\[
= (f_1 f_2^\dagger + f_1^\dagger f_1)(f_2 f_2^\dagger + f_2^\dagger f_2) = I.
\]

5.2 Parallel gates

Let \( A \) be an \( n \)-qubit gate represented by QRA element \( f_A \) and \( B \) be an \( m \)-qubit gate represented by QRA element \( f_B \).

Then, the parallel circuit depicted in Fig. 3 forms an \((n + m)\)-qubit gate. Now, the establishment of the appropriate QRA representation of such gates will be illustrated using an example of two parallel 2-qubit NOT gates:

- 2-qubit NOT gate, acting on 2-qubit state, lies in the algebra QRA(2), with the QRA representation (based on elements \( f_1, f_2 \)) in the form

\[
\text{NOT} = f_2^\dagger f_1 + f_1^\dagger f_2 + f_1^\dagger f_2^\dagger + f_2 f_1.
\] (29)
– The realization of a quantum circuit based on the two parallel 2-qubit \textit{NOT} gates from Fig. 3 belongs to \textit{QRA(4)}. The circuit acts on the first two qubits with \textit{NOT}_1 and, at the same time, acts on the last two qubits with \textit{NOT}_2:

\begin{align}
\text{NOT}_1 &= f_2^\dagger f_1 + f_1^\dagger f_2 + f_1 f_2^\dagger + f_2 f_1, \\
\text{NOT}_2 &= f_4^\dagger f_3 + f_3^\dagger f_4 + f_3 f_4^\dagger + f_4 f_3. 
\end{align}

\[ (30) \]

\[ (31) \]

– The key aspect of computations in the \textit{QRA} is the following. The Geometric Algebra multiplication of two gates acting on different qubits forms a parallel circuit based on these gates (in the way of Fig. 3) up to the sign of the individual monomials. In fact, the \textit{QRA} representation of two parallel gates is achieved in two steps. Firstly, we multiply their \textit{QRA} representations. Secondly, we discuss and correct the signs of the particular monomials.

– In the following expression, we denote a parallel circuit based on gates \textit{NOT}_1 and \textit{NOT}_2 as \textit{NOT}_1 \otimes \textit{NOT}_2, and the terms \( f_i f_j \otimes f_k f_l \) denote the states before the sign discussion, so they are equal to \( f_i f_j f_k f_l \) or \(- f_i f_j f_k f_l \). We can see \( \otimes \) as an operation which is distributive and associative, i.e.,

\begin{align}
\text{NOT}_1 \otimes \text{NOT}_2 \\
= (f_2^\dagger f_1 + f_1^\dagger f_2 + f_1 f_2^\dagger + f_2 f_1) \otimes (f_4^\dagger f_3 + f_3^\dagger f_4 + f_3 f_4^\dagger + f_4 f_3) \\
= f_2^\dagger f_1 \otimes f_4^\dagger f_3 + f_2^\dagger f_1 \otimes f_3^\dagger f_4 + f_2^\dagger f_1 \otimes f_3 f_4^\dagger + f_2^\dagger f_1 \otimes f_4 f_3 + f_1^\dagger f_2 \otimes f_4^\dagger f_3 + f_1^\dagger f_2 \otimes f_3^\dagger f_4 + f_1^\dagger f_2 \otimes f_3 f_4^\dagger + f_1^\dagger f_2 \otimes f_4 f_3 + f_1^\dagger f_2 \otimes f_4^\dagger f_3 + f_1^\dagger f_2 \otimes f_3^\dagger f_4 + f_1^\dagger f_2 \otimes f_3 f_4^\dagger + f_1^\dagger f_2 \otimes f_4 f_3 + f_1 f_2 \otimes f_4^\dagger f_3 + f_1 f_2 \otimes f_3^\dagger f_4 + f_1 f_2 \otimes f_3 f_4^\dagger + f_1 f_2 \otimes f_4 f_3 + f_2 f_1 \otimes f_4^\dagger f_3 + f_2 f_1 \otimes f_3^\dagger f_4 + f_2 f_1 \otimes f_3 f_4^\dagger + f_2 f_1 \otimes f_4 f_3 + f_2^\dagger f_1 \otimes f_4^\dagger f_3 + f_2^\dagger f_1 \otimes f_3^\dagger f_4 + f_2^\dagger f_1 \otimes f_3 f_4^\dagger + f_2^\dagger f_1 \otimes f_4 f_3 + f_2 f_1 \otimes f_4^\dagger f_3 + f_2 f_1 \otimes f_3^\dagger f_4 + f_2 f_1 \otimes f_3 f_4^\dagger + f_2 f_1 \otimes f_4 f_3. 
\end{align}

\[ (32) \]

– To obtain the correct Geometric Algebra elements from the abstract notation above, we have to check the sign of every element according to the following rules:
– Find the number of elements \((f_i \text{ or } f_i^\dagger)\) on the right hand side of the tensor product. For example, in the expression \(f_2^\dagger f_1 \otimes f_4^\dagger f_3\), two elements appear on the right. Set \(s_1 = 2\).

– Find the number of elements \(f_\ell \text{ or } f_\ell^\dagger\) on the left hand side of the tensor product. For example, in the expression \(f_2^\dagger f_1 \otimes f_4^\dagger f_3\), there is one \(f_1\). Set \(s_2 = 1\).

– Then, set \(s = s_1s_2 = 2\) and obtain the correct sign as \((-1)^s\).

– Finally, we have

\[
f_2^\dagger f_1 \otimes f_4^\dagger f_3 = (-1)^2 f_2^\dagger f_1 f_4^\dagger f_3 = f_2^\dagger f_1 f_4^\dagger f_3.
\]

(33)

– It is easy to see that in all monomials of our \(NOT_1 \otimes NOT_2\) gate, there are exactly two elements on the right hand side, so the number \(s\) will always be even, implying that the sign of each monomial has to be positive:

\[
NOT_1 \otimes NOT_2 = f_2^\dagger f_1 f_4^\dagger f_3 + f_2^\dagger f_1 f_3^\dagger f_4 + f_2^\dagger f_3 f_4^\dagger f_3 + f_2^\dagger f_1 f_4^\dagger f_3 \\
+ f_1^\dagger f_2 f_3^\dagger f_3 + f_1^\dagger f_2 f_3^\dagger f_4 + f_1^\dagger f_2 f_3^\dagger f_4 + f_1^\dagger f_2 f_4 f_3 \\
+ f_1^\dagger f_2 f_4^\dagger f_3 + f_1^\dagger f_2 f_3^\dagger f_4 + f_1^\dagger f_2 f_3^\dagger f_4 + f_1^\dagger f_2 f_4 f_3 \\
+ f_2^\dagger f_1 f_4^\dagger f_3 + f_2^\dagger f_1 f_4^\dagger f_3 + f_2^\dagger f_1 f_3^\dagger f_4 + f_2^\dagger f_1 f_4^\dagger f_3.
\]

(34)

To explain our approach better, we show one more nontrivial example. Let us consider CNOT and NOT gates:

\[
CNOT = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|,
\]

\[
NOT = |0\rangle\langle 1| + |1\rangle\langle 0|.
\]

In QRA(3), these gates are represented by Geometric Algebra elements

\[
CNOT = f_1 f_2^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger,
\]

\[
NOT = f_3 + f_3^\dagger.
\]

Before the actual computation, the main reason for the sign discussion will be discussed. At first, let us show how parallel gates work in general: let us apply \(CNOT\) and \(NOT\) gates on qubits \(I = |00\rangle\) and \(I = |0\rangle\) respectively, i.e.,

\[
(f_1 f_1^\dagger f_2^\dagger f_2 + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger) I \otimes (f_3 + f_3^\dagger) I = I \otimes f_3^\dagger I.
\]

If we apply \(CNOT\) and \(NOT\) gates on qubits \(f_1^\dagger I = |10\rangle\) and \(I = |0\rangle\) in the same way, we obtain

\[
(f_1 f_1^\dagger f_2^\dagger f_2 + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger) f_1^\dagger I \otimes (f_3 + f_3^\dagger) I
\]
\[
\begin{align*}
&= (-f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger) f_1^\dagger \otimes (f_3 + f_3^\dagger) I = (f_1^\dagger f_2^\dagger) I \otimes f_3^\dagger I.
\end{align*}
\]

Therefore, \((CNOT \otimes NOT)\) gate acts on these elements in the following way:

\[
\begin{align*}
CNOT|00\rangle \otimes NOT|0\rangle &= I \otimes f_3^\dagger I = |00\rangle \otimes |1\rangle = |001\rangle, \\
CNOT|10\rangle \otimes NOT|0\rangle &= (f_1^\dagger f_2^\dagger) I \otimes f_3^\dagger I = |11\rangle \otimes |1\rangle = |111\rangle.
\end{align*}
\]

On the other hand, if we multiply \(CNOT\) and \(NOT\), this geometric product can be applied directly on 3-qubit states \(I = |000\rangle\) and \(f_1^\dagger I = |100\rangle\) to receive the results

\[
\begin{align*}
(f_1 f_1^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger)(f_3 + f_3^\dagger) I &= (f_3^\dagger) I
\end{align*}
\]

and

\[
\begin{align*}
(f_1 f_1^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger)(f_3 + f_3^\dagger) f_1^\dagger I &= -(f_1^\dagger f_2^\dagger) f_3^\dagger I.
\end{align*}
\]

Thus, multiplication works correctly only on some of the qubits, while for others a change of sign appears (this problem has to be corrected by the sign discussion). For example, the monomial \(-f_1^\dagger f_1^\dagger f_2 \otimes f_3\) has \(s_1 = 1\) and \(s_2 = 1\). Therefore, \(s = 1\) and

\[
-f_1^\dagger f_1^\dagger f_2 \otimes f_3 = (-1)^1(-f_1^\dagger f_1^\dagger f_2 \otimes f_3) = f_1^\dagger f_1^\dagger f_2 \otimes f_3,
\]

which fixes the wrong sign.

Computing the parallel \(CNOT \otimes NOT\) gate as

\[
\begin{align*}
(f_1 f_1^\dagger f_2 f_2^\dagger + f_1 f_1^\dagger f_2^\dagger f_2 - f_1^\dagger f_1 f_2 - f_1^\dagger f_1 f_2^\dagger) \otimes (f_3 + f_3^\dagger) &= f_1 f_1^\dagger f_2 f_2^\dagger \otimes f_3 + f_1 f_1^\dagger f_2^\dagger f_2 \otimes f_3 - f_1^\dagger f_1 f_2 \otimes f_3 - f_1^\dagger f_1 f_2^\dagger \otimes f_3 \\
&+ f_1 f_1^\dagger f_2 f_2^\dagger \otimes f_3 + f_1 f_1^\dagger f_2^\dagger f_2 \otimes f_3 - f_1^\dagger f_1 f_2 \otimes f_3 - f_1^\dagger f_1 f_2^\dagger \otimes f_3,
\end{align*}
\]

we finally obtain

\[
\begin{align*}
CNOT \otimes NOT &= f_1 f_1^\dagger f_2 f_2^\dagger f_3 - f_1 f_1^\dagger f_2^\dagger f_2 f_3 - f_1^\dagger f_1 f_2 f_3 + f_1^\dagger f_1 f_2^\dagger f_3 \\
&+ f_1 f_1^\dagger f_2 f_2^\dagger f_3 - f_1 f_1^\dagger f_2^\dagger f_2 f_3 - f_1^\dagger f_1 f_2 f_3 + f_1^\dagger f_1 f_2^\dagger f_3.
\end{align*}
\]

Note that we can see QRA as a real instance of a complex Geometric Algebra. With the help of this identification, the properties of serial and parallel gates can be derived from the complex case as well [9].
6 GAALOP examples

To reproduce the following examples, use GAALOPWeb with Latex output and QRA(4). In Listing 7, we introduce $\iota$ and compute two gates in serial circuit according to Sect. 5.1.

Listing 7 GAALOPScript for the computation of two serial NOT gates.

```cpp
i = er1*er2;
f1 = 0.5*(e1+i*e5);
f1T = 0.5*(e1-i*e5);
f2 = 0.5*(e2+i*e6);
f2T = 0.5*(e2-i*e6);
NOT = f2T*f1 + f1T*f2 + f1T*f2T + f2*f1;
S = NOT*NOT;
```

At first, we define an imaginary unit $\iota$ ($i$ in the listing) and the part of the Witt basis needed for this example according to (7) and (8). Then, we define the NOT gate and compute the product of two such gates. The resulting multivector $S$ has only one component,

$$S_0 = 1/1.0,$$

which is a scalar (indicated by GAALOPWeb as the comment $//1.0$) and its value is simply 1. Multiplication of this scalar by $I$ proves the result of Sect. 5.1.

Listing 8 shows the computation for parallel NOT gates.

Listing 8 GAALOPScript for the computation of two parallel NOT gates.

```cpp
i = er1*er2;
f1 = 0.5*(e1+i*e5);
f1T = 0.5*(e1-i*e5);
f2 = 0.5*(e2+i*e6);
f2T = 0.5*(e2-i*e6);
f3 = 0.5*(e3+i*e7);
f3T = 0.5*(e3-i*e7);
f4 = 0.5*(e4+i*e8);
f4T = 0.5*(e4-i*e8);
NOT1 = f2T*f1 + f1T*f2 + f1T*f2T + f2*f1;
NOT2 = f4T*f3 + f3T*f4 + f3T*f4T + f4*f3;
P = NOT1*NOT2;
```

Now, we have to extend the Witt basis to the case of four qubits, define two NOT gates and multiply them. In this case, the resulting multivector has also only one

---

5 QRA with 4 qubits.
component,

\[ P_{282} = \frac{1}{e^2} \wedge (e^4 \wedge (e^5 \wedge e^7)), \]  

(37)

which is the element with index 282 representing the blade \( e^2 \wedge e^4 \wedge e^5 \wedge e^7 \) and its value is 1.

GAALOPWeb can also be used for the simplification of expression (34) according to Listing 9.

Listing 9 GAALOPScript for the simplification of expression (34).

```plaintext
1 i = er1*er2;
2 f1 = 0.5*(e1+i*e5);
3 f1T = 0.5*(e1-i*e5);
4 f2 = 0.5*(e2+i*e6);
5 f2T = 0.5*(e2-i*e6);
6 f3 = 0.5*(e3+i*e7);
7 f3T = 0.5*(e3-i*e7);
8 f4 = 0.5*(e4+i*e8);
9 f4T = 0.5*(e4-i*e8);
10 ?E = f2T*f1*f4T*f3 + f2T*f1*f3T*f4 + f2T*f1*f3T*f4T + f2T*f1*f4*f3 + f1T*f2*f4T*f3 + f1T*f2*f3T*f4 + f1T*f2*f3T*f4T + f1T*f2*f4*f3 + f2*f1*f4T*f3 + f2*f1*f3T*f4 + f2*f1*f3T*f4T + f2*f1*f4*f3;
```

Again, the result of the multivector \( E \) is

\[ E_{282} = \frac{1}{e^2} \wedge (e^4 \wedge (e^5 \wedge e^7)), \]

which corresponds to the element \( e^2 \wedge e^4 \wedge e^5 \wedge e^7 \) as expected.

When we use GAALOPWeb to simplify the result of computation of the parallel \( \text{CNOT} \) and \( \text{NOT} \) gates according to Listing 10,

Listing 10 GAALOPScript for the simplification of the parallel \( \text{CNOT} \) and \( \text{NOT} \) gate.

```plaintext
1 i = er1*er2;
2 f1 = 0.5*(e1+i*e5);
3 f1T = 0.5*(e1-i*e5);
4 f2 = 0.5*(e2+i*e6);
5 f2T = 0.5*(e2-i*e6);
6 f3 = 0.5*(e3+i*e7);
7 f3T = 0.5*(e3-i*e7);
8 f4 = 0.5*(e4+i*e8);
9 f4T = 0.5*(e4-i*e8);
10 ?C = f1*f1T*f2*f2T*f3 - f1*f1T*f2T*f2*f3 - f1T*f1*f2*f3 + f1T*f1T*f2*f2T*f3T - f1T*f1T*f2T*f2*f3T + f1T*f1T*f2*f3T - f1T*f1T*f2*f3T;
```
the result is a multivector $C$ with the following components:

\[
C_{210} = 0.5 / e_1 \wedge (e_3 \wedge (e_5 \wedge e_6)), \\
C_{346} = -0.5 / e_3 \wedge (e_6 \wedge (e_1 \wedge e_2)), \\
C_{392} = 0.5 / e_1 \wedge (e_2 \wedge (e_3 \wedge (e_5 \wedge e_6))), \\
C_{542} = -0.5 / e_2 \wedge (e_3 \wedge (e_6 \wedge (e_1 \wedge e_2))).
\]

### 6.1 GAALOP to Mathematica

To perform the tensor product between CNOT and NOT using GAALOP, we assign a variable for each term of both operators representing the possible results for $s_1$ and $s_2$ as follows,

\[
\text{CNOT} = a_2 f_1 f_1^\dagger f_2 f_2^\dagger + a_1 f_1 f_1^\dagger f_2^\dagger f_2 - a_2 f_1^\dagger f_1 f_2 f_2^\dagger - a_1 f_1^\dagger f_1^\dagger f_2 f_2,
\]

\[
\text{NOT} = b_1 f_3 + b_1 f_3^\dagger,
\]

where $a_1$ and $a_2$ represent $s_1 = 1$ and $s_1 = 2$, respectively; $b_1$ and $b_2$ represent $s_2 = 1$ and $s_2 = 2$, respectively. Then, we perform geometric product between both operators and obtain resulting coordinates which depend on the products

\[
\{a_1 b_1, a_1 b_2, a_2 b_1, a_2 b_2\}.
\]

where only $a_1 b_1$, which represents $s = s_1 s_2$ with $s_1 = 1$ and $s_2 = 1$, leads to the sign $-1$, whereas the remaining products return $1$. Thus, after compiling the code from Listing 1 in GAALOPWeb and exporting it to Wolfram Mathematica, it is sufficient to assign these values for the corresponding products.

**Listing 11** Code for the tensor product $CNOT \otimes NOT$.

```plaintext
1  // Imaginary unit
2  i = er1*er2;
3  // Witt basis
4  f1 = 0.5*( e1+i*e5 );
5  f1T = 0.5*( e1 -i*e5 );
6  f2 = 0.5*( e2+i*e6 );
7  f2T = 0.5*( e2 -i*e6 );
8  f3 = 0.5*( e3+i*e7 );
9  f3T = 0.5*( e3 -i*e7 );
10 f4 = 0.5*( e4+i*e8 );
11 f4T = 0.5*( e4 -i*e8 );
12
13  // a1: s1=1; a2: s1=2; b1: s2=1; b2: s2=2;
14
15  CNOT=a2*f1*f1T*f2*f2T + a1*f1*f1T*f2T*f2
```
Quantum Register Algebra: the mathematical language…

GAALOPWeb generates a module function with the file name (see [1]) to compute the product. Outside the module, we use the following command to assign the values for the products (38).

Listing 12 Module function.

```plaintext
Expand[TensorCnotNot[a1, a2, b1, b2]] /. 
{a1*b1 -> -1, a1*b2 -> 1, 
a2*b1 -> 1, b1*b2 -> 1},
```

where “TensorCnotNot” is the name of the module function and the sequence /.{a1 * b1 → −1, a1 * b2 → 1, a2 * b1 → 1, b1 * b2 → 1} assigns the correct sign for each term. The output shows that only four coordinates are nonzero,

\[
\begin{align*}
0.5 \quad & e_1 \wedge (e_2 \wedge (e_5 \wedge e_7)), \\
0.5 \quad & e_2 \wedge (e_7 \wedge (r_1 \wedge r_2)), \\
0.5 \quad & e_3 \wedge (e_6 \wedge (r_1 \wedge r_2)), \\
0.5 \quad & e_1 \wedge (e_3 \wedge (e_5 \wedge (r_1 \wedge r_2))),
\end{align*}
\]

which are the same as the ones that we have already obtained from the computation in Listing 10.

7 Generalized SWAP gates

In this section, we explain the benefits of the proposed approach with respect to generalized SWAP gates. In a general \(n\)-qubit circuit, it is natural to consider the SWAP action between two arbitrary selected qubits. Let us consider a 3-qubit circuit with a SWAP action on a second and third qubit. In matrix notation, we obtain

\[
E \otimes SWAP = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} SWAP & 0_{4\times4} \\ 0_{4\times4} & SWAP \end{pmatrix}.
\]

In Dirac notation, we have the expression

\[
(\vert 0 \rangle \langle 0 \vert + \vert 1 \rangle \langle 1 \vert) \otimes (\vert 00 \rangle \langle 00 \vert + \vert 01 \rangle \langle 10 \vert + \vert 10 \rangle \langle 01 \vert + \vert 11 \rangle \langle 11 \vert)
\]

\[
= (\vert 000 \rangle \langle 000 \vert + \vert 001 \rangle \langle 010 \vert + \vert 010 \rangle \langle 001 \vert + \vert 011 \rangle \langle 011 \vert)
\]

\[
+ (\vert 100 \rangle \langle 100 \vert + \vert 101 \rangle \langle 110 \vert + \vert 110 \rangle \langle 101 \vert + \vert 111 \rangle \langle 111 \vert).
\]
The resulting expression of Dirac representation has eight terms that correspond to eight non-zero elements of the matrix representation. Using the QRA, the same gate is represented by the following half size element:

\[
    \text{SWAP}(2, 3) = f_2 f_2^\dagger f_3 f_3^\dagger + f_2^\dagger f_3 - f_2 f_3^\dagger + f_2^\dagger f_2 f_3 f_3^\dagger.
\]

We can say that the QRA representation of SWAP, according to (18), considers only indices of “swapped” qubits. The correctness of the representation can be verified as follows. In the QRA, 3-qubits are represented by the expression

\[
    |\psi\rangle = (a_{000} + a_{010} f_2^\dagger + a_{001} f_3^\dagger + a_{011} f_2 f_3^\dagger) + f_1^\dagger (a_{100} + a_{110} f_2^\dagger + a_{101} f_3^\dagger + a_{111} f_2 f_3^\dagger)
\]

and the SWAP(2,3) acts on \( |\psi\rangle \) as

\[
    \text{SWAP}(2, 3) |\psi\rangle = \text{SWAP}(2, 3)(\cdots) + \text{SWAP}(2, 3) f_1^\dagger (\cdots) = \text{SWAP}(2, 3)(\cdots) + f_1^\dagger \text{SWAP}(2, 3)(\cdots).
\]

The element \( f_1^\dagger \) commutes with SWAP because the indices are different and SWAP has all terms of even grade. Since our calculations are done in a well-defined algebra, we are able to verify the representation correctness automatically with the help of GAALOP. The following code

Listing 13 Code for the automatic proving of SWAP(2,3) correctness.

```plaintext
1 i = er1 * er2 ;
2 f1 = 0.5*( e1 + i * e4 ) ;
3 f1T = 0.5*( e1 - i * e4 ) ;
4 f2 = 0.5*( e2 + i * e5 ) ;
5 f2T = 0.5*( e2 - i * e5 ) ;
6 f3 = 0.5*( e3 + i * e6 ) ;
7 f3T = 0.5*( e3 - i * e6 ) ;
8 Id = f1 * f1T * f2 * f2T * f3 * f3T ;
9 psi = a000*Id+a100*f1T+a010*f2T+a110*f1T*f2T
10 +a001*f3T+a101*f1T*f3T
11 +a011*f2T*f3T+a111*f1T*f2T*f3T ;
12 SWAP23psi= a000*Id+a100*f1T+a010*f2T
13 +a101*f1T*f2T+a001*f2T
14 +a110*f1T*f3T+a011*f2T*f3T+a111*f1T*f2T*f3T ;
15 SWAP23 = f2 * f2T * f3 * f3T + f2T * f3 + f3T * f2
16 + f2T * f3T * f3 * f2 ;
17 ?X=SWAP23*psi*Id-SWAP23psi*Id;
```

has the output
Listing 14 Output for the automatic proving of SWAP(2,3) correctness.

```matlab
function [X] = script()
end
```

which proves the statement (absence of input and assignment means that the result of the difference of the two expressions is zero).

Finally, we note that SWAP(2,3) acts as a SWAP on elements that do not have a $\sigma_1$ in them. Generalizing this property to $n$-qubits can be done in straightforward way.

8 Conclusion

This paper develops the concept of Quantum Register Algebra (QRA) applied to quantum computing presented at the conference ICACGA 2022 [10]. We have explained how QRA can be used as an efficient tool for quantum computing and described how to construct quantum circuits within the Geometric Algebra framework. QRA allowed us to make a direct transcription from Dirac formalism to Geometric Algebra formalism and to discuss the related problems involving serial and parallel connections of Clifford and non-Clifford quantum gates. Moreover, using the newly developed QRA and its support by the extended GAALOP tool, we have also explained that the handling of quantum computing can be substantially simplified. Finally, we would like to thank the reviewers and editors for their friendly yet critical approach that has helped and improved our work.

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The Authors declare that there is no conflict of interest.

A Appendix: QRA configuration in GAALOP

The integration of QRA into GAALOP is done based on the file Definition.csv according to Chapter 9 of [13]. Listing 15 shows the file for the definition of one qubit.

Listing 15 Definition.csv for QRA for one qubit.

```
1, e1, e2, er1, er2
```

In general, this file consists of 5 lines defining an algebra. In the case of QRA, lines 2 and 5 are left blank since the used basis in line 1 and the standard basis in line 3 are the same and no transformations between the two bases are needed. The basis
is defined by basis vectors $e_1, e_2, e_{r1}, e_{r2}$, according to $e_1, e_2, r_1, r_2$ defined in the previous sections. All their squares are defined to be equal to 1, according to line 4.

For each additional qubit, we need two additional basis vectors, while $e_{r1}$ and $e_{r2}$ remain the same. The definition of a register with two qubits is shown in Listing 16.

Listing 16 Definition.csv for QRA based on two qubits.

|   | 1, e1, e2, e3, e4, e_{r1}, e_{r2} |
|---|----------------------------------|
| 1 | 1, e1, e2, e3, e4, e_{r1}, e_{r2} |
| 2 | e1=1, e2=1, e3=1, e4=1, e_{r1}=1, e_{r2}=1 |

For the second qubit, we need two additional basis vectors $e_3$ and $e_4$.

References

1. Alves, R., Hildenbrand, D., Steinmetz, C., Uftring, P.: Efficient development of competitive Mathematica solutions based on Geometric Algebra with GAALOPWeb. Adv. Appl. Clifford Algebras 30, 59 (2020)
2. Alves, R., Hildenbrand, D., Hrdina, J., et al.: An online calculator for quantum computing operations based on Geometric Algebra. Adv. Appl. Clifford Algebras 32, 4 (2022)
3. Bravyi, S.B., Kitaev, A.Y.: Fermionic quantum computation. Ann. Phys. 298(1), 210–226 (2002)
4. Cafaro, C., Mancini, S.: A Geometric Algebra perspective on quantum computational gates and universality in quantum computing. Adv. Appl. Clifford Algebras 21, 493–519 (2011)
5. Doran, C., Lasenby, A.: Geometric Algebra for Physicists. Cambridge University Press (2003)
6. Doran, C., Hestenes, A., et al.: Lie groups as spin groups. J. Math. Phys. 34(8), 66 (1993)
7. de Lima Marquezino, F., Portugal, R., Lavor, C.: A Primer on Quantum Computing. Springer, Berlin (2019)
8. Eryganov, I., Hrdina, J.: Complex Clifford algebra in repeated quantum prisoner’s dilemma. Meth. Appl. Sci. (2022). https://doi.org/10.1002/mma.8694
9. Hrdina, J., Návrat, A., Vašík, P.: Quantum computing based on complex Clifford algebras. Quantum Inf. Process. 21, 310 (2022). https://doi.org/10.1007/s11128-022-03648-w
10. Hrdina, J., Hildenbrand, D., Návrat A. et al.: Quantum Register Algebra: The Basic Concepts, LNCS paper, in print (2022)
11. Hildenbrand, D.: Foundations of Geometric Algebra Computing. Springer, Berlin (2013)
12. Hildenbrand, D.: Introduction to Geometric Algebra Computing. CRC Press/Taylor & Francis, Boca Raton (2019)
13. Hildenbrand, D.: The Power of Geometric Algebra Computing. CRC Press/Taylor & Francis Group, Boca Raton (2022)
14. Hitzer, E., Lavor, C., Hildenbrand, D.: Current survey of Clifford geometric algebra applications. Math. Meth. Appl. Sci. (2022). https://doi.org/10.1002/mma.8316
15. Lounesto, P.: Clifford Algebra and Spinors, 2nd edn. CUP, Cambridge (2006)
16. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information, vol. 10, Anniversary Cambridge University Press, Cambridge (2010)
17. Pavsic, M.: Quantized Fields á la Clifford and Unification. In: Licata, I. (ed.) Beyond Peaceful Coexistence: The Emergence of Space, Time and Quantum. Imperial College Press, London (2016)
18. Perwass, Ch.: Geometric Algebra with Applications in Engineering, 1st edn. Springer, Berlin (2009)
19. Tilly, J., Chen, H., et al.: The variational quantum eigensolver: a review of methods and best practices. Phys. Rep. 986, 1–128 (2022)
Quantum Register Algebra: the mathematical language…

Authors and Affiliations

J. Hrdina\textsuperscript{1} · D. Hildenbrand\textsuperscript{2} · A. Návrat\textsuperscript{1} · C. Steinmetz\textsuperscript{4} · R. Alves\textsuperscript{3} · C. Lavor\textsuperscript{5} · P. Vašík\textsuperscript{1} · I. Eryganov\textsuperscript{1}

D. Hildenbrand
dietmar.hildenbrand@gmail.com

A. Návrat
navrat.a@fme.vutbr.cz

C. Steinmetz
christian.steinmetz@e-mail.de

R. Alves
alves.rafael@ufabc.edu.br

C. Lavor
clavor@unicamp.br

P. Vašík
vasik@fme.vutbr.cz

I. Eryganov
xperyga00@vutbr.cz

\textsuperscript{1} Brno University of Technology, Brno, Czech Republic
\textsuperscript{2} Technische Universität Darmstadt, 64297 Darmstadt, Germany
\textsuperscript{3} Federal University of ABC, Sao Bernardo, Brazil
\textsuperscript{4} :em engineering methods AG, Darmstadt, Germany
\textsuperscript{5} University of Campinas, IMECC-UNICAMP, Campinas, Brazil