Garvie, Liam and Duncan, Ross (2018) Verifying the smallest interesting colour code with quantomatic. Electronic Proceedings in Theoretical Computer Science, EPTCS, 266. pp. 147-163. ISSN 2075-2180, http://dx.doi.org/10.4204/EPTCS.266.10

This version is available at https://strathprints.strath.ac.uk/64967/

Strathprints is designed to allow users to access the research output of the University of Strathclyde. Unless otherwise explicitly stated on the manuscript, Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Please check the manuscript for details of any other licences that may have been applied. You may not engage in further distribution of the material for any profitmaking activities or any commercial gain. You may freely distribute both the url (https://strathprints.strath.ac.uk/) and the content of this paper for research or private study, educational, or not-for-profit purposes without prior permission or charge.

Any correspondence concerning this service should be sent to the Strathprints administrator: strathprints@strath.ac.uk
Verifying the Smallest Interesting Colour Code with Quantomatic

Liam Garvie and Ross Duncan
University of Strathclyde
26 Richmond Street, Glasgow G1 1XH, Scotland

In this paper we present a Quantomatic case study, verifying the basic properties of the Smallest Interesting Colour Code error detection code.

1 Introduction

Error correction will form a crucial layer in the software stack of any realistic quantum computer for the foreseeable future. Since the implementation of any error correction scheme will depend on the details of the actual hardware, it is generally expected that error correction will be added to a quantum program late in the compilation process (see, for example [18]). In other words, the fault-tolerant executable program will be automatically generated from a higher-level description which is unaware of the error-correction scheme to be employed. At a minimum, any such translation process from “logical” quantum circuits to their fault-tolerant versions should be proven sound—i.e. that translation does not change the meaning of the program. However we might demand more. For example, with knowledge of the hardware operations and the fault-tolerant program, it may be possible to optimise beneath the error correction scheme. Such optimisations again require correctness proofs, not just of soundness, but to guarantee that the optimisation preserves fault-tolerance.

To do any of this, a language combining circuits, error-correcting schemes, and translations between the two is required, and this language should support robust and powerful automated reasoning, capable of proving that relevant properties of error-correcting schemes hold. In this paper we present a case study along these lines. Precisely, we use the zx-calculus [12] as a language, in concert with the interactive theorem prover Quantomatic [19,24], to study the Smallest Interesting Colour Code [7]. We provide formal proofs of the basic properties of the code itself and its fault-tolerant operations.

A similar study was conducted in 2013 for the 7-qubit Steane code [15] (see also the recent [10]), however the Quantomatic system has undergone significant development in the intervening four years, and is vastly more powerful. Many of our proofs can be produced automatically by Quantomatic; others require the formalisation of human insight into reusable tactics; still others resist full automation with the current technology. In the final section we discuss the obstacles encountered, and desiderata for future development of the zx-calculus/Quantomatic system.

Acknowledgements The authors wish to thank Aleks Kissinger: without his timely intervention this project would have never have been completed.
The Quantomatic project files  All the proofs which appear in this paper and its appendix are publicly available as a downloadable Quantomatic project at [https://gitlab.cis.strath.ac.uk/kwb13215/Colour-Code-QPL](https://gitlab.cis.strath.ac.uk/kwb13215/Colour-Code-QPL).

Supplementary Material  Links to the supplementary material, containing the full proofs of the main results are found in Appendix [C].

2 The Code

The “smallest interesting colour code” takes its name from a blog post by Earl Campbell [7] which provided the inspiration for this project, and also appears in the papers [9, 8, 26]. It is an [[8, 3, 2]] code, meaning it encodes 3 logical qubits into 8 physical qubits, and has a distance of 2, meaning it can detect any single qubit error but not correct it. The code can be presented geometrically as a cube where each vertex corresponds to a qubit.

In this picture, the Z-stabilizers of the code correspond to the face and cell operators of the cube, and the single X-stabilizer corresponds to the cell; see [7] for nice pictures.

The logical Pauli operators are transversal, with the three logical X operators obtained by applying X to the three faces of the cube, while the logical Z is obtained as an edge operator.

From the description of the X operators, it’s straightforward to find the translation of the computational basis states to codewords, which is shown in Table [4].

So, why is this code interesting? It is the smallest member of the class of quasi-transversal codes [9, 8], which allow the efficient synthesis of multiqubit non-Clifford gates from noisy magic states [6]. For our purposes, its key property is that the logical 3-qubit doubly-controlled Z (CCZ) gate is transversal, and can be implemented using the single-qubit non-Clifford T gate.

\[ CCZ = \text{diag}(1, 1, \ldots, 1, -1) \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \]
Table 1: The computational basis as codewords

<table>
<thead>
<tr>
<th>logical state</th>
<th>codeword state</th>
<th>logical state</th>
<th>codeword state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1/12[(00000000) + (11111111)]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1/√2[(10101010) + (01010101)]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1/2[(11001100) + (00110011)]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1/√2[(01100110) + (10011001)]</td>
<td></td>
</tr>
</tbody>
</table>

More precisely, the CCZ is obtained by applying $T$ to black vertices and $T^\dagger$ to white vertices. Therefore this code enables the distillation of 8 noisy T-states into a less noisy CCZ logical state. Since CCZ is Clifford equivalent to a Toffoli gate, this code provides a basis for fault-tolerant quantum computation. Therefore it is a particularly interesting test case for the use of zx-calculus and automated reasoning.

3 The zx-calculus

The zx-calculus [12] is a universal formal graphical notation for representing quantum states and processes, and a sound equational theory for reasoning about them. In this section we briefly review its syntax and semantics.

Since its introduction in [11], the zx-calculus has undergone various refinements, and various changes to the axioms have been considered; for examples see [16, 17, 13, 27]. In this paper, we use the unconditional calculus, without scalars, without supplementarity, and we take the Hadamard gate as a primitive element. This version of the calculus is complete for the stabilizer fragment of quantum mechanics [1], and for the 1-qubit Clifford+$T$ group [2]; however it is not complete for the full, multi-qubit, Clifford+$T$ group [27], a fact which will be pertinent later.

Definition 3.1. A term of the zx-calculus is a finite open graph whose boundary is partitioned into inputs and outputs, and whose interior vertices are of the following types:

- $Z(\alpha)$ vertices, labelled by an angle $\alpha$, $0 \leq \alpha < 2\pi$. These are depicted as green or light grey circles; if $\alpha = 0$ then the label is omitted.
- $X(\beta)$ vertices, labelled by an angle $\beta$, $0 \leq \beta < 2\pi$. These are are depicted as red or dark grey circles; again, if $\beta = 0$ then the label is omitted.
- $H$ vertices; unlike the other types $H$ vertices are constrained to have degree exactly 2. They are depicted as yellow squares.

The allowed vertex types are shown in Figure 1. We adopt the convention that inputs are on the left, and outputs on the right.

Terms, also called diagrams, may be composed by joining some number (maybe zero) of the outputs of one term to the inputs of another. Given a diagram $D : n \to m$ we define its adjoint

1By sound we mean every equation derivable in the zx-calculus calculus is true in the standard Hilbert space interpretation; by complete we mean that every true statement in the Hilbert space model is derivable in the zx-calculus. Note that the calculus is not complete in this sense, but is complete for fragments of quantum theory.

2That is, for simplicity we will assume all the scalars are 1 and drop them whenever they occur; this is harmless, since in this work the zero scalar will not arise, and all the rest can be restored if needed. See Backens [3] for a rigorous treatment of scalar factors in the zx-calculus.
$D^\dagger: m \to n$ to be the diagram obtained by reflecting the diagram around the vertical axis and negating all the angles. The terms of the zx-calculus naturally form a $\dagger$-symmetric monoidal category [13], or more precisely a $\dagger$-PROP [25, 14], which we call ZX. The only use we will make of this fact is defining the semantics as a (strict) monoidal functor. Note that the diagrams with a single interior vertex generate all the rest by composition.

**Definition 3.2.** Given a zx-term $D: n \to m$, its interpretation is a linear map $[D]: (\mathbb{C}^2)^\otimes n \to (\mathbb{C}^2)^\otimes m$ defined as follows:

$$
\begin{align*}
\begin{bmatrix}
|0\rangle^\otimes m \\
|1\rangle^\otimes m
\end{bmatrix} &\mapsto |0\rangle^\otimes n \\
|1\rangle^\otimes m &\mapsto e^{i\alpha}|1\rangle^\otimes n
\end{align*}
\quad
\begin{align*}
\begin{bmatrix}
|+\rangle^\otimes m \\
|−\rangle^\otimes m
\end{bmatrix} &\mapsto |+\rangle^\otimes n \\
|−\rangle^\otimes m &\mapsto e^{i\beta}|−\rangle^\otimes n
\end{align*}
\quad
\begin{bmatrix}
\alpha
\end{bmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix}
1 & 1 \\
1 & −1
\end{pmatrix}.
\end{align*}
$$

Since the above diagrams are the generators of ZX, the above definition extends uniquely to a strict $\dagger$-symmetric monoidal functor $[\cdot]: ZX \to fdHilb$.

Any quantum circuit can be translated into the zx-calculus via the universal gate set shown.

Basic state preparations have a simple form:

$$
|0\rangle = \begin{bmatrix}
\bullet
\end{bmatrix} \quad |1\rangle = \begin{bmatrix}
\pi
\bullet
\end{bmatrix} \quad |+\rangle = \begin{bmatrix}
\bullet
\end{bmatrix} \quad |−\rangle = \begin{bmatrix}
\pi
\bullet
\end{bmatrix}
$$
However not all gates have simple translations. Using the standard decomposition [5], the CCZ is translated via CNOTs and single-qubit phase gates, leading to quite a complex diagram.

This diagram can be reduced to a simpler form, but not greatly so. Note the use of $T = Z(\frac{\pi}{4})$ gates makes explicit that this diagram is outside the Clifford fragment and in the realm where the zx-calculus is not complete.

The zx-calculus has a rich equational theory based on the theory of Frobenius-Hopf algebras [12, 14]. A concise\(^3\) presentation of the axioms is shown in Figure 2. As noted earlier, the equations are sound \(^4\) with respect to $\llbracket \cdot \rrbracket$.

4 Working with Quantomatic

We now give a brief overview of Quantomatic for the uninitiated. For a full description of the system see [24]; to obtain it, see [19].

Quantomatic is an interactive theorem prover which can prove equations between terms of the zx-calculus. The user draws the desired term in the graphical editor, and builds the proof by applying rewrite rules to the current graph. A rewrite rule is a directed equation; i.e. two
graphs which have the same boundary. The user selects the rule or rules they wish to apply and Quantomatic will display where the LHS of the rule matches a subgraph of the current term, alongside the term obtained by replacing the matched subgraph with the RHS of the rule. A proof consists of a sequence of terms linked by the application of a particular rule at a particular location in the term.

A Quantomatic proof development begins with a set of axioms, i.e. rules without proofs which are justified semantically. Whereas in pen-and-paper presentations (like Figure 2) we might aim for a minimal set of rules, for automated proof developments a larger, possibly redundant, set of rules is usually more convenient. However any proof can be promoted to a lemma, which can then be used as a rule. In the course of a proof development the user will typically build up a collection of increasingly powerful lemmas encapsulating pieces of local reasoning within the larger proof. In some cases the lemmas may simply be special cases of more general axioms intended to cut down the search space of possible rule applications.

Quantomatic also supports automated deduction via simplification procedures, aka simprocs. A simproc is a simple program built from combinators which allow the sequencing of rules, greedy application of every rule in a given ruleset, reduction according to some rules until some condition is met, and so on. Simprocs allow quite lengthy proofs to be carried out automatically. However some caution is required. The complete zx-calculus is neither terminating nor confluent, so the rules to be included in a simproc must be chosen with care. Termination can be ensured by using only rules where the LHS is in some sense strictly “bigger” than the RHS; obviously the graph cannot get smaller forever. However many essential rules (e.g. π-commute) do not fit this pattern. The other danger is that by using a simproc we give up control of where in the term rewrites will be applied; lack of confluence means it is possible for the simproc to make a bad choice, from which it cannot reach the desired final term. In practice this means that a proof often consists of an alternation of simproc application and human selected rewrites which expand the graph, although depending on the proof it is sometimes possible to formalise this alternation itself in a simproc.

Some of the rules shown in Figure 2 have ellipses, to indicate that the given diagram indicates an infinite family of terms. Such families are formalised in Quantomatic using !-boxes. Briefly, a !-box is a marked subgraph within a term, which indicates that the subgraph may occur any number of times when matching the graph. For example, the π-commute (below left) rule may be formalised using !-boxes as shown on the right,

\[
\pi \alpha \rightarrow \pi -\alpha
\]

where the blue boxes are the indicated subgraphs. Note that the matching of the LHS in the target graph determines how many times the box should be repeated, on both sides of the rule. Many of the multi-ary rules of the zx-calculus are formalised using !-boxes; in order to better control of the matching algorithm, it is often necessary to derive fixed arity special cases as lemmas.

\[\text{^5}\text{The axiom ruleset used for this project is given in Appendix B.}\]
\[\text{^6}\text{Less briefly, see [20, 21, 22, 28] for a full treatment of !-boxes and their associated logic.}\]
val red_pi_lemma = load_rule "created_theorms/pauli/red_pi_lemma";
val red_sp = load_rule "axioms/red_sp";

val reduction_before_pi = load_ruleset [
  "axioms/red_copy", "axioms/red_sp", "axioms/green_sp", "axioms/hopf",
  "axioms/red_scalar", "axioms/green_scalar", "axioms/green_id",
  "axioms/red_id", "axioms/red_loop", "axioms/green_loop"
];

val simproc = (LOOP (REDUCE_ALL reduction_before_pi ++ REWRITE red_pi_lemma ++ REDUCE red_sp

Figure 3: An example simproc: push_pauli_x

5 The Encoder and Decoder

No circuit for encoding the logical qubits into physical qubits is given in any of the references [7, 9, 8, 26] so our first task is to define the encoding operation in the zx-calculus. Looking at Table 1, we can see that the basis states are prepared by conditionally applying Pauli X operations an 8-qubit GHZ state. In the zx-calculus GHZ states have a very simple form:

\[
|00000000\rangle + |11111111\rangle
\]

The bit flips are applied by “copying” a Pauli X from a (logical) input qubit to the appropriate four physical qubits, as shown below.

The final encoder is obtained by composing these four diagrams along the 8 physical qubits; logical qubits come in on the left, codewords come out on the right. The decoder is simply the
encoder in reverse.

We note that the graph for Enc contains no $H$ vertices and no angles. For such terms we have the following result:

**Proposition 5.1.** [28] Any $z x$-calculus term without angles or $H$ vertices is equal to a term in subset-spanning form; further, the subset-spanning form is quasi-unique and can be computed by a terminating algorithm.

Space does not permit a description of the quasi-normal forms; the important point here is that the normalisation procedure is implemented in a Quantomatic simproc called `rotate_simp`. We will make extensive use of this in the sequel.

**Proposition 5.2.** Encoding followed by decoding is the identity, i.e. $\text{Enc}^\dagger \circ \text{Enc} = \text{id}_3$, or in pictures:

This result can be proven by `rotate_simp` without human intervention; it requires 66 rewrite steps and takes approximately 5 seconds of real time. See Supplementary Material C.1.1. Note that Proposition 5.2 doesn’t establish that the 8 codewords of Table 1 are prepared as required; this is a corollary of the tranversality of the Pauli group, which we show in the next section.

The astute reader will have noticed that the encoder is not actually a quantum circuit: the top “rails” end in a projection onto the state $|+\rangle$. However, no post-selection is required: we can implement Enc with a circuit using five ancilla qubits.

**Proposition 5.3.** $\text{Enc}$ is a unitary embedding.

**Proof.** We do the proof in two stages, shown below.
(For the full proof see Supplementary Material C.1.2.) From the final diagram, one can easily read the translation back into conventional circuit notation.

\[\equiv\]

The first equation above is the normalisation of \(\text{Enc}\) using \texttt{rotate\_simp}; it takes 28 rewrite steps and 2 seconds. The second equation required substantial human intervention. The proof requires \textit{expanding} the graph rather than reducing it, orienting the rewrite rules in the opposite direction to normal. If naively applied this would lead to an uncontrolled blow-up of the graph. Further, even under human control, the normal ruleset produces too many matches to be helpful. For these reasons it was necessary to prove some (near trivial) intermediate lemmas:

\textbf{Lemma 1.}
\begin{equation*}
\begin{array}{c}
\text{(bialgebra x1)}
\end{array}
\end{equation*}

\textbf{Lemma 2.}
\begin{equation*}
\begin{array}{c}
\text{(bialgebra x1, Z identity x2)}
\end{array}
\end{equation*}

\textbf{Lemma 3.}
\begin{equation*}
\begin{array}{c}
\text{(bialgebra x1, X identity x2)}
\end{array}
\end{equation*}

The sharp-eyed will notice that each of these lemmas introduces an explicit CNOT into the circuit. However, they are not mathematically interesting statements: they are chosen to have few matches on the in-progress proof term, and hence take control of the matching algorithm.

6 Fault-tolerant operations

Given our error correcting scheme, we are interested in the \textit{fault-tolerant operations}. That is, unitary operators on the codeword space which commute with the encoder. More precisely, given a unitary \(L\) acting on the logical Hilbert space, and a unitary \(P\) acting on the physical Hilbert space we say that \(L\) is the fault tolerant version of \(P\) if the equation

\[\text{Enc} \circ L = P \circ \text{Enc}\]  \hspace{1cm} (1)

holds. Since we don’t really care what \(P\) does to non-codewords, and by virtue of Proposition 5.2 the weaker equation

\[L = \text{Enc}^\dagger \circ P \circ \text{Enc}\]  \hspace{1cm} (2)

will suffice; given the nature of our tools, reducing a big complicated graph to a small one will usually be easier than showing that two rough-equally-complex graphs are equal. Statements of the form (1) are most easily proven by rewriting both sides of the equation to a common reduct.
6.1 The Paulis

As claimed in Section 2, the Pauli gates are transversal for this code.

**Proposition 6.1.** We have the following equations:

\[
\begin{align*}
\text{Enc} \circ (1 \otimes 1 \otimes X) &= (X \otimes X \otimes X \otimes X \otimes 1 \otimes 1 \otimes 1 \otimes 1) \circ \text{Enc} \quad (x1) \\
\text{Enc} \circ (1 \otimes X \otimes 1) &= (X \otimes X \otimes 1 \otimes 1 \otimes X \otimes X \otimes 1 \otimes 1) \circ \text{Enc} \quad (x2) \\
\text{Enc} \circ (X \otimes 1 \otimes 1) &= (X \otimes 1 \otimes X \otimes 1 \otimes X \otimes X \otimes 1 \otimes 1) \circ \text{Enc} \quad (x3) \\
\text{Enc} \circ (1 \otimes 1 \otimes Z) &= (Z \otimes 1 \otimes 1 \otimes 1 \otimes Z \otimes 1 \otimes 1 \otimes 1) \circ \text{Enc} \quad (z1) \\
\text{Enc} \circ (1 \otimes Z \otimes 1) &= (Z \otimes 1 \otimes Z \otimes 1 \otimes 1 \otimes 1 \otimes 1 \otimes 1) \circ \text{Enc} \quad (z2) \\
\text{Enc} \circ (Z \otimes 1 \otimes 1) &= (Z \otimes Z \otimes 1 \otimes 1 \otimes 1 \otimes 1 \otimes 1 \otimes 1) \circ \text{Enc} \quad (z3)
\end{align*}
\]

**Proof.** We demonstrate (x1) by rewriting as shown below.

![Rewrite Diagram]

The right-hand rewrite sequence is purely reduction, while the left-hand sequence makes use of one expansion step. Both parts of the proof can be accomplished by the simproc `push_pauli_x`, shown in Figure 3. The proof has a total of 32 individual steps. The other equations are proved in the same way, although the proofs of equations (z1), (z2), and (z3) use a `push_pauli_z` simproc, which is the colour dual of `push_pauli_x`.

**Corollary 6.2.** The computational basis states are prepared as shown in Table 1.

6.2 The CNOT

Since the CNOT gate preserves the computational basis states, by inspecting Table 1 we can work out what the encoded CNOT should be. For example, let:

\[
\text{CNOT}^L_{2,3} = \quad \text{CNOT}^P_{2,3} =
\]

**Proposition 6.3.** With the above definitions, \( \text{CNOT}^L_{2,3} = \text{Enc}^\dagger \circ \text{CNOT}^P_{2,3} \circ \text{Enc} \).
Proof. We reduce as shown:

Noticing that neither diagram has any angles, we can again employ \texttt{rotate_simp}; this does most of the work, but terminates in a different quasi-normal form to what we want. However, it requires only one expansion step to get over this hurdle, and simple reductions can automatically finish the proof. The complete proof has 75 steps, and can be found in the Supplementary Material C.2.3.  

The other possible CNOTs can be discovered by applying permutations to the above, and the same proof technique will establish their correctness.

6.3 The CCZ

As noted in Section 2, the main point of interest of this code is the fact that the CCZ gate can be implemented fault tolerantly using only \( T \) and \( T^\dagger \) operations on the physical qubits. Precisely:

\[
CCZ_{1,2,3}^P = T \otimes T^\dagger \otimes T \otimes T^\dagger \otimes T \otimes T^\dagger \otimes T \otimes T^\dagger.
\]

To establish the correctness of this claim, ideally we would prove:

\[
CCZ_{1,2,3}^L = \text{Enc}^\dagger \circ CCZ_{1,2,3}^P \circ \text{Enc},
\]

or, in pictures:
Unlike our previous results, the non-Clifford $T$ gate plays a crucial role here, and the \( zx \)-calculus is known to be incomplete for the Clifford+$T$ fragment of quantum mechanics, even after the addition of equations which we do not consider here [27]. It remains possible that \([3]\) may be established from the usual \( zx \)-calculus axioms, however this proved beyond our reach. We resort to brute force:

**Proposition 6.4.** For \( x, y \in \{0, 1\} \) such that \( xy = 0 \) we have:

\[
\begin{align*}
(\text{Enc}^\dagger \circ \text{CCZ}^P_{1,2,3} \circ \text{Enc})|11+\rangle &= |11-\rangle \\
(\text{Enc}^\dagger \circ \text{CCZ}^P_{1,2,3} \circ \text{Enc})|11-\rangle &= |11+\rangle \\
(\text{Enc}^\dagger \circ \text{CCZ}^P_{1,2,3} \circ \text{Enc})|xy\pm\rangle &= |xy\pm\rangle
\end{align*}
\]

**Proof.** We treat each of the 8 cases separately by performing, for example, the reduction:

A typical such proof has 80–90 steps; the majority can be performed by either the basic simplifier or \texttt{rotate_simp}, however at several points specially prepared lemmas are required to simplify specific graphs that occurred during the proof. The proof strategy did not appear amenable to automation. The complete proof is in Supplementary Material C.2.4.

**Corollary 6.5.** \( \text{CCZ}^P_{1,2,3} \) acts as a logical CCZ on code words.

7 Conclusions and Future Work

In this paper we have formalised the Smallest Interesting Colour Code in the \( zx \)-calculus, and provided formal proofs certifying some of its basic properties. Along the way we discovered a novel quantum circuit for encoding three input qubits into the 8-qubit codeword space. We did not consider a deterministic decoder, nor the error detecting circuit for this code.

The size of the graphs involved in these terms, and the number of steps involved in the proofs, meant that Quantomatic was indispensible to this work. The power of the standard simplifiers (\texttt{basic_simp} and especially \texttt{rotate_simp}) is striking; given such a hammer, everything looks like a nail. Therefore, while lemmas could in principle be used to shorten and add conceptual structure to proofs in this work they have mostly been used to climb out of a position where

\[\text{Evaluating the map on a basis is not sufficient to establish the equality unless global phase is taken into account. However evaluating the map on the equal sum of the basis elements will suffice to complete the proof, even without phases. This is done in the electronic material.}\]

\[\text{Indeed, while typesetting the enormous proof of Proposition 6.4 in Supplementary Material 6.3 we discovered a much shorter and clearer proof; sadly too late to be included in this version of the paper.}\]
the simplifiers got stuck. From this point of view there are two main uses of lemmas: (1) to chain together some rewrites in an order the simplifier would not choose; and (2) to embed a rewrite in a larger graph and thus restrict where the matcher may apply it. For example we count many variations on the theme of “unspidering” among the frequently employed lemmas. Lacking any other mechanism to control where rewrites will be applied, lemmas of the second type are essential to writing useful simprocs. Care must also be taken with the formulation of the axioms to control matching. For example, \texttt{gen\_bialg\_qrule} (See Appendix B) is formulated too generally to be useful. While it does encapsulate the general form of the bialgebra equation, its left-hand-side also matches on the empty graph and every edge of a 2-coloured graph. The resulting deluge of trivial matches is more hindrance than help.

During our attack on Proposition 6.4 we codified Selinger and Bian’s generators and relations for 2-qubit Clifford+T circuits [29] hoping the additional equations would help. However, their novel relations involve large circuits, which did not occur in the graphs we were working with, and so could not be used. A useful rule must have a left-hand-side which captures some non-trivial feature of a term already in reduced form (i.e. two-colored and simple).

The most glaring absence from this work is the Clifford group. No treatment was attempted, due to limited available time, but is the most obvious next step. Beyond that, this code is an interesting test case for further study of the Clifford+T fragment in the zx-calculus. Recall from Section 6.3 that the encoded CCZ is very simple in comparison to the usual 3-qubit circuit. It is also rather simpler than the encoded CNOT circuit we presented in Section 6.2 which, without performing any optimisation, contains five physical CNOT operations. Since CCZ is Clifford equivalent to Toffoli, the CNOT can be generated by CCZs and Cliffords. This implies there exists a family of equations between CNOT circuits and Clifford+T circuits which we conjecture are not provable in the zx-calculus. This offers a new approach to completing the calculus, while helping with applications along the way. More generally, applying automated reasoning and the zx-calculus to verify practical gate synthesis protocols such as [9, 8] is an important area for further investigation.

References


The Smallest Interesting Colour Code in Quantomatic


A  Electronic Resources

All the work described in this paper is available as a downloadable quantomatic project from the following URL.

https://gitlab.cis.strath.ac.uk/kwb13215/Colour-Code-QPL

Please download it and play around!

B  The Ruleset

green_sp.qrule  green_id.qrule  green_loop.qrule

\[
\begin{align*}
\text{x} & = \text{x} + \text{y} \\
\text{y} & = \text{y} \\
\text{z} & = \text{z}
\end{align*}
\]

green_scalar.qrule  green_pi.qrule  green_copy.qrule

\[
\begin{align*}
\text{π} & = -\text{π} \\
\text{zx} & = -\text{x}
\end{align*}
\]

green_elim.qrule  red_sp.qrule  red_id.qrule

\[
\begin{align*}
\text{x} & = \text{x} + \text{y} \\
\text{y} & = \text{y} \\
\text{z} & = \text{z}
\end{align*}
\]

red_loop.qrule  red_scalar.qrule  red_pi.qrule

\[
\begin{align*}
\text{πx} & = -\text{x} \\
\text{x} & = \text{x}
\end{align*}
\]
C Proofs omitted from main article

For those readers who prefer to see the proofs written down rather than as a Quantomatic proof development, they are available in supplementary document. This available in two places:

- The second author’s website, at the time of writing:
  http://personal.strath.ac.uk/ross.duncan/

- The first version of this paper stored on the arXiv (arxiv:1706.02717v1):
  https://arxiv.org/abs/1706.02717v1