Surface code note

1. 简介

Surface code 是一种量子纠错码，属于stabilizer code 的一种，它的实现利用了平面的拓扑结构，具有很好的局部性，并且对于Pauli error （由 X, Y 算子生成）具有优秀、简明的纠错策略。和大多数的stabilizer code 一样，surface code 可以方便地定义 X_L, Z_L 算子，也很容易初始化为这些算子的特征态$|0\rangle, |+\rangle$。此外，通过对编码多个logic qubit的surface code进行一定的变换，则可以让我们定义 $CNOT$ gate。

2. 背景

这里我们主要考虑Pauli error，每个qubit都有可能经历随机的 $X, Z, Y(=ZX)$的变换，可以表达为operator sum的形式，如果初始时系统状态用density operator ρ 表示，则错误的作用为：

$$\varepsilon(\rho) = p_1\rho + p_2X\rho X + p_3Y\rho Y + p_4Z\rho Z$$

而所谓纠错，就是要找到一个作用C（也可以表示为operator sum的形式），使得对于这种错误，有:

$$C(\varepsilon(\rho)) = \rho$$

更一般来说，所有的错误都可以建模为：

$$\varepsilon(\rho) = \sum_i p_iE_i\rho E_i^T, p_i \geq 0$$

只需要该作用可以保持矩阵的trace和正定性。我们把 $E := \{E_i\}$ 称为错误类型，若某种纠错作用C可以对 E 纠错，可以证明，对于E中元素的线性组合也可以纠错。

假设量子系统由多个qubit组成，其空间表示为 H，则元素 $\{\pm 1(i)I_j, \pm 1(i)X_j, \pm 1(i)Z_j\}$ 生成了H 上的observable的一个有限群（称为Pauli群）P。对于 C 是 P 的一个子集，我们称 C stabilize $|\phi\rangle$ 若对于 C 中任意元素 c ，都有 $c|\phi\rangle = |\phi\rangle$，可以证明：

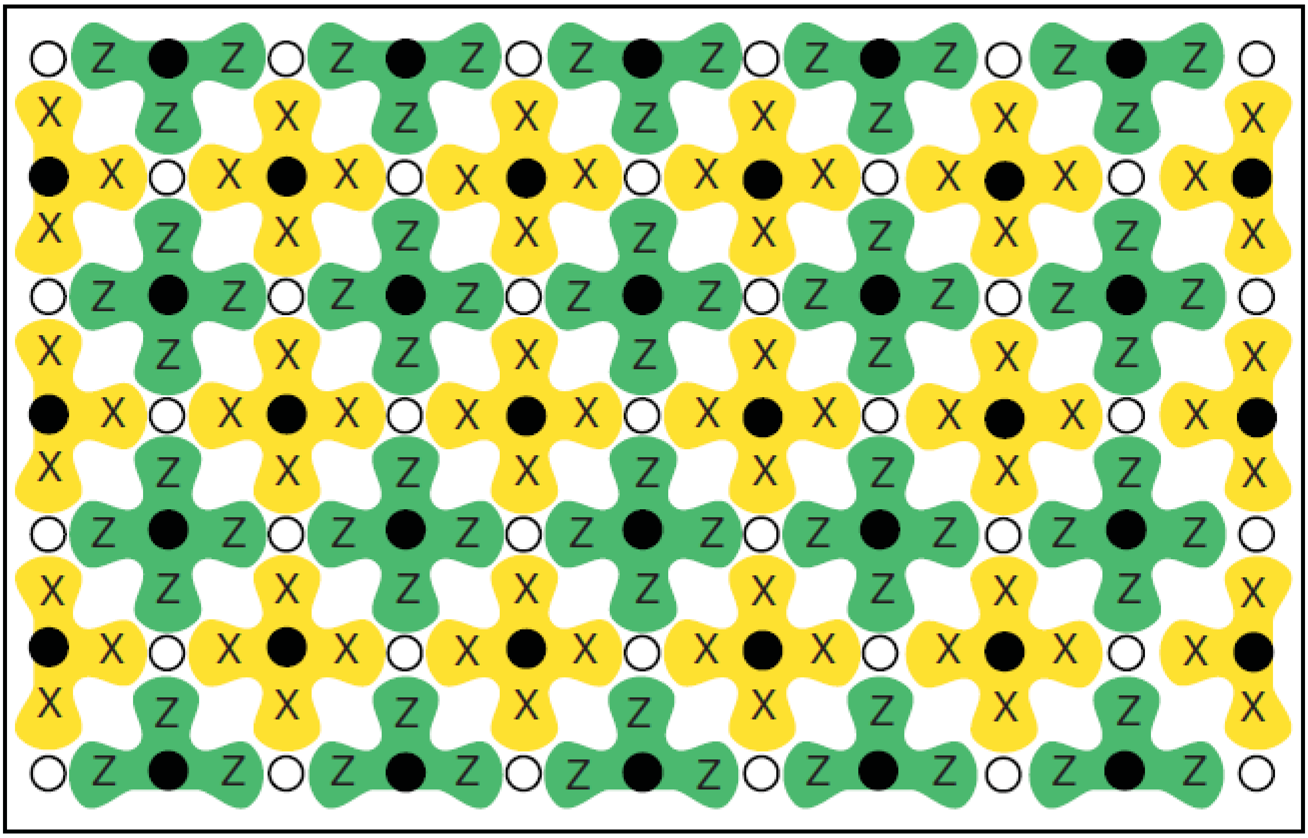
1. 若 C stabilize $|\phi\rangle$ ，则 C 生成的子群也是如此，因此接下来我们可以假设 C 是一个群；
2. 被 C stabilize 的元素组成 H 的一个子空间，记做 V_C；
3. 若 $dimH = h$，C 不含 $-I$，且它的最小生成元集合大小为 k，则 $dimV_C = h - k$。

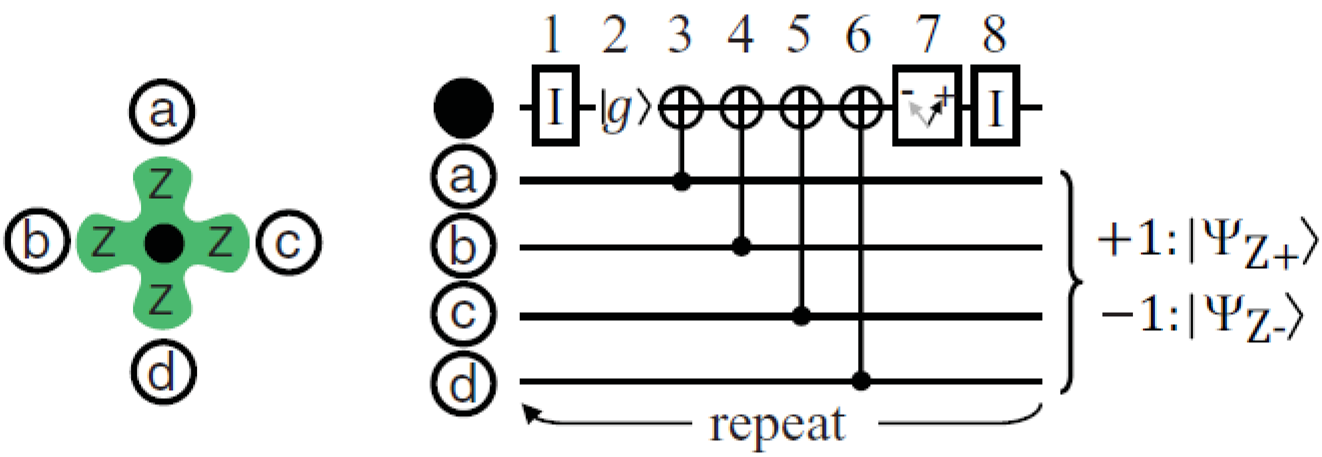
若 C 的最小生成元集合是 G_C，并且我们可以让它们都是互相可交换的，这样我们在这些observable上可以同时进行测量，并要求测量结果为 $+1$，这样我们就得到了一个子空间，可以用于编码 $h-k$ 个logic qubit。我们可以在上面定义 $X_{L,j}, Z_{L,j}$，只需要它们满足这样的性质即可：即不同位置上的算子可交换；$X_{L,j}^2 = I, Z_{L,j}^2 = I$；同一位置的不同算子反交换；这些operator可以被限制在 V_C 上。

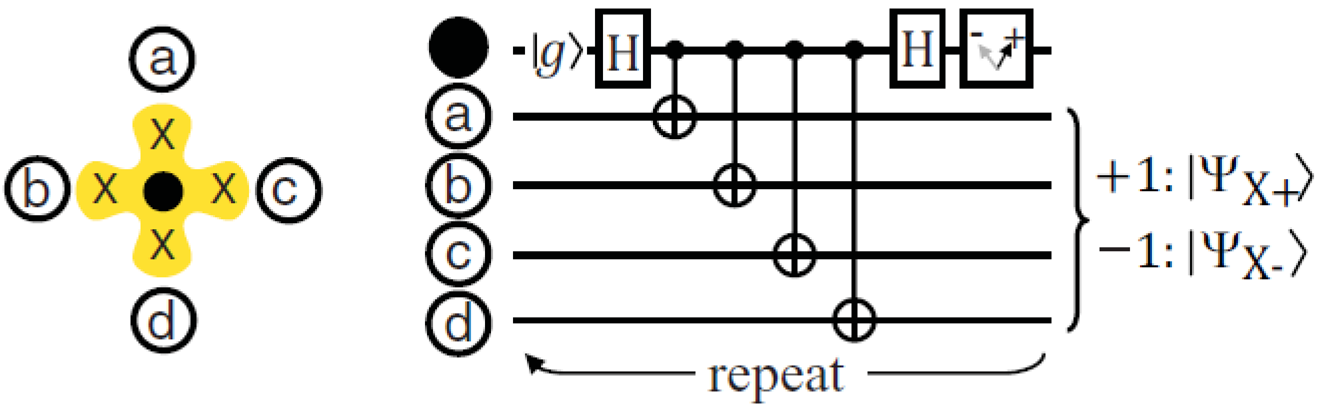
事实上，在这些observable上进行测量，我们可以得到一个诊断结果（每个位置上的测量结果），如果这个结果和上一个结果相比改变了，则我们可以推断有错误发生，若错误发生的概率很低，我们可以用算法计算出最有可能发生的错误，然后进行纠错。确实有错误可以保持测量结果不变，我们发现这些错误恰好是我们刚刚定义的逻辑算子，这样的错误是难以发现的，因为我们把这种错误所改变的qubit的最小数量称为是这个stabilizer code的距离 d。

3. Surface code

Surface code 的构造如下图所示，其中白色圆点表示data qubit，黑色圆点表示measure qubit，measure qubit又分为两种，用 X 和 Z 表示，它们与周围的 4 个 data qubit 连接（在边界处的是 3 个）







可以看到这两种连接分别表示在 $Z_aZ_bZ_cZ_d$ 和 $X_aX_bX_cX_d$ 这两个observable上进行测量，这些observable正是surface code所对应的 C 的生成元，我们可以验证按照上图的排布，这些observable都是可交换的。

我们把在上述测量操作中保持不变的状态称之为静止态（仅针对data qubit，measure qubit的存在只是辅助我们进行测量），对于 2^k 个不同的测量结果，静止态可能生活在其中任何一个测量结果所定义的子空间中。具体来说，假设某个measure qubit执行测量$Z_aZ_bZ_cZ_d$，得到结果Z_{abcd}，则我们可以令 $Z_{abcd}\cdot Z_aZ_bZ_cZ_d\in C$，对每个位置的测量结果都如此做后，可以得到对应的 V_C，所谓静止态就生活在 V_C 中。一般来说，我们不会事先规定这些observable的符号，而是在经过多轮测量得到稳定的结果后才规定符号，然后得到对应的stabilizer code。在进行量子计算时，我们也一般从一个静止态出发，此后在每一轮测量中，如果没有错误发生，测量结果始终和上一轮一样。

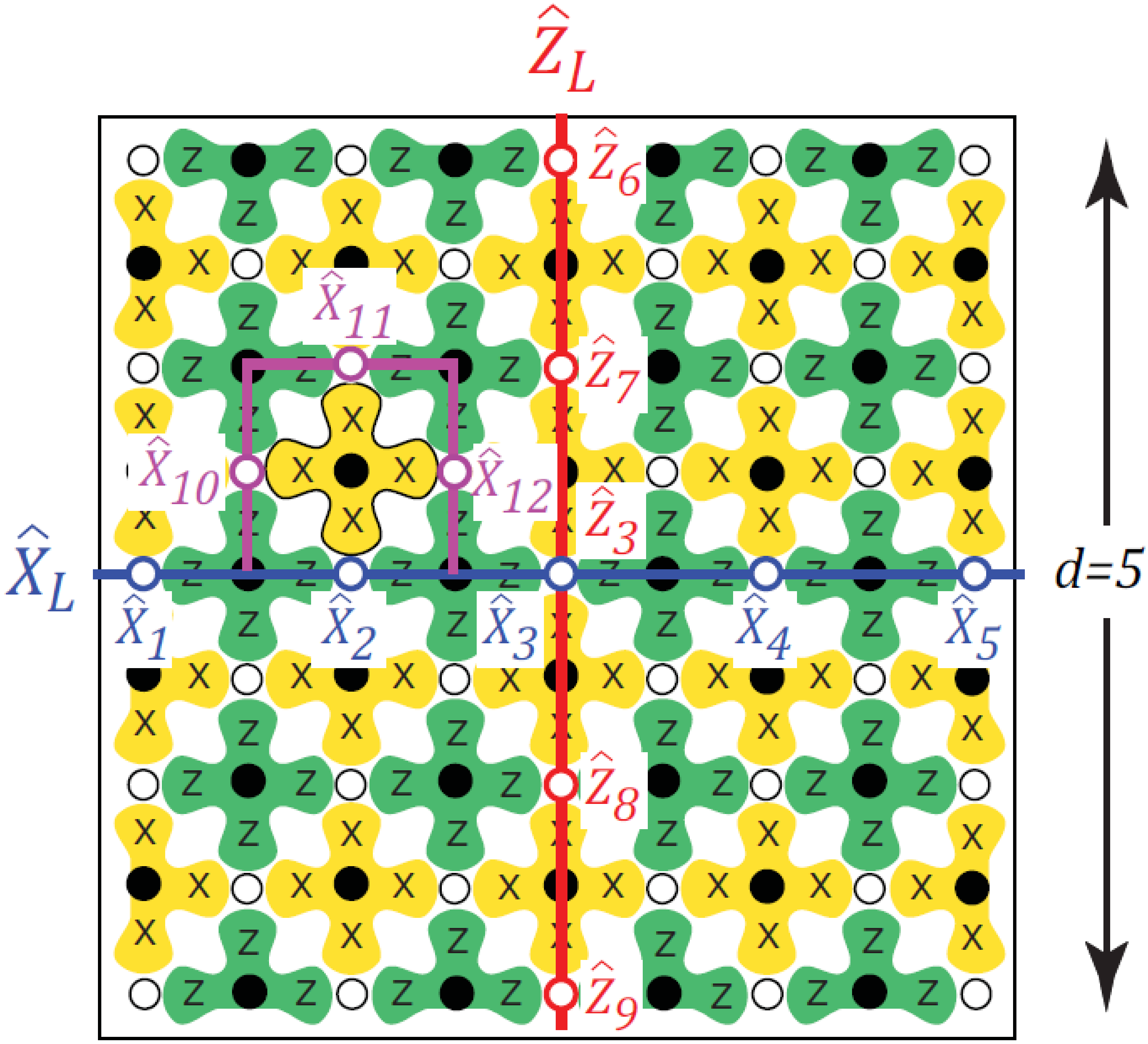
4. 单比特错误

首先，我们可以考虑在一个data qubit 上面的 X 错误，一旦发生，则和其相邻的 Z measure qubit 的测量结果都会发生改变，因此我们可以定位这样的错误，对于 Y 错误，也是类似的。

当然，在测量过程中发生的错误也需要考虑，但是这种错误往往只会改变一个测量结果，而且由于每一轮的测量都是独立的，若假设发生错误的概率很低，我们可以进行多轮测量，则很大概率测量结果就会回归正常，因此对于测量错误我们也可以检测。

5. X_L 与 Z_L operator

X_L 和 Z_L 算子可以按照下图定义，其中 X_L 是对蓝色线上的 data qubit 作用 X 算子，蓝色线连接平面的由 X measure qubit 组成的边界；而 Z_L 则是对红色线上的 data qubit 作用 Z 算子，连接 Z measure qubit 边界。可以验证，对于以下图方式编码的logic qubit，这样定义的逻辑算子确实满足X_L 和 Z_L 的要求，不会改变测量结果。

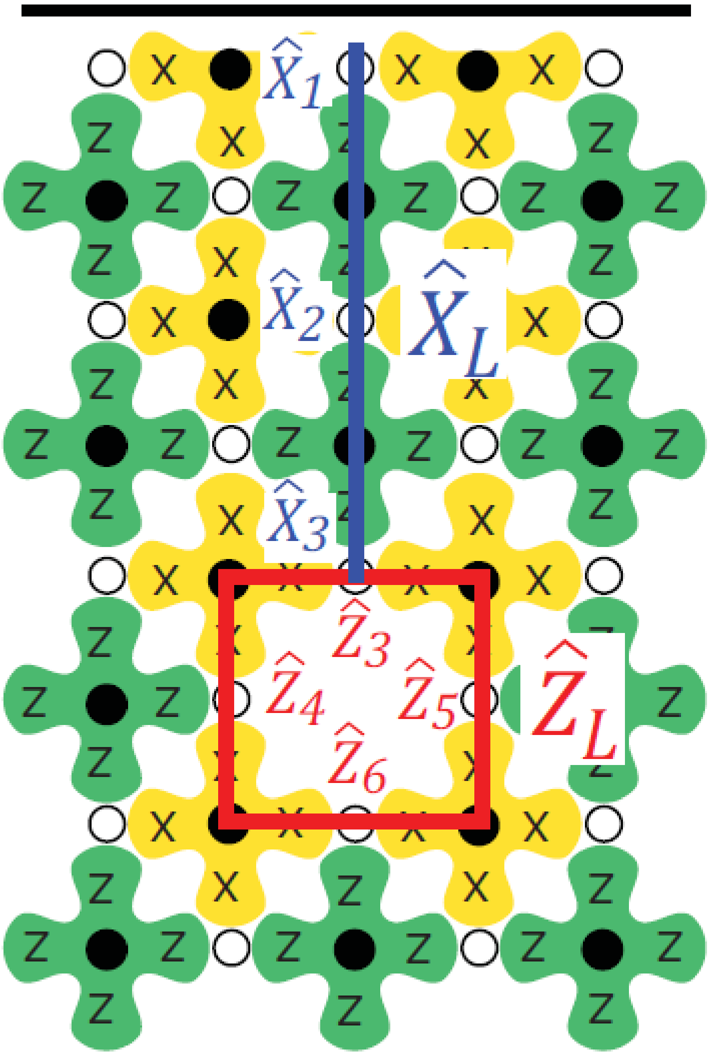


可以注意到，还有其他定义方式，比如说：可以将蓝色线换成紫色线，但是我们可以注意到这样得到的 $X_L' = (X_2 X_{10} X_{11} X_{12}) \cdot X_L$，只相差一个 C 中的元素（不考虑符号），这意味着在stabilizer code 的子空间中，这两个算子只相差一个global phase，即测量结果 $X_{2,10,11,12}$。因此，所有的合法的逻辑算子在这种意义下都是等价的。

当我们确定测量结果后，我们可以把 H 分解为 $Q \otimes L$，每一个静止态都可以表示为 $|Q\rangle|q_L\rangle$，C 中的元素只作用在空间 Q 上，且保持 $|Q\rangle$ 不变，而我们刚刚定义的逻辑算子则作用在 $|q_L\rangle$ 上。

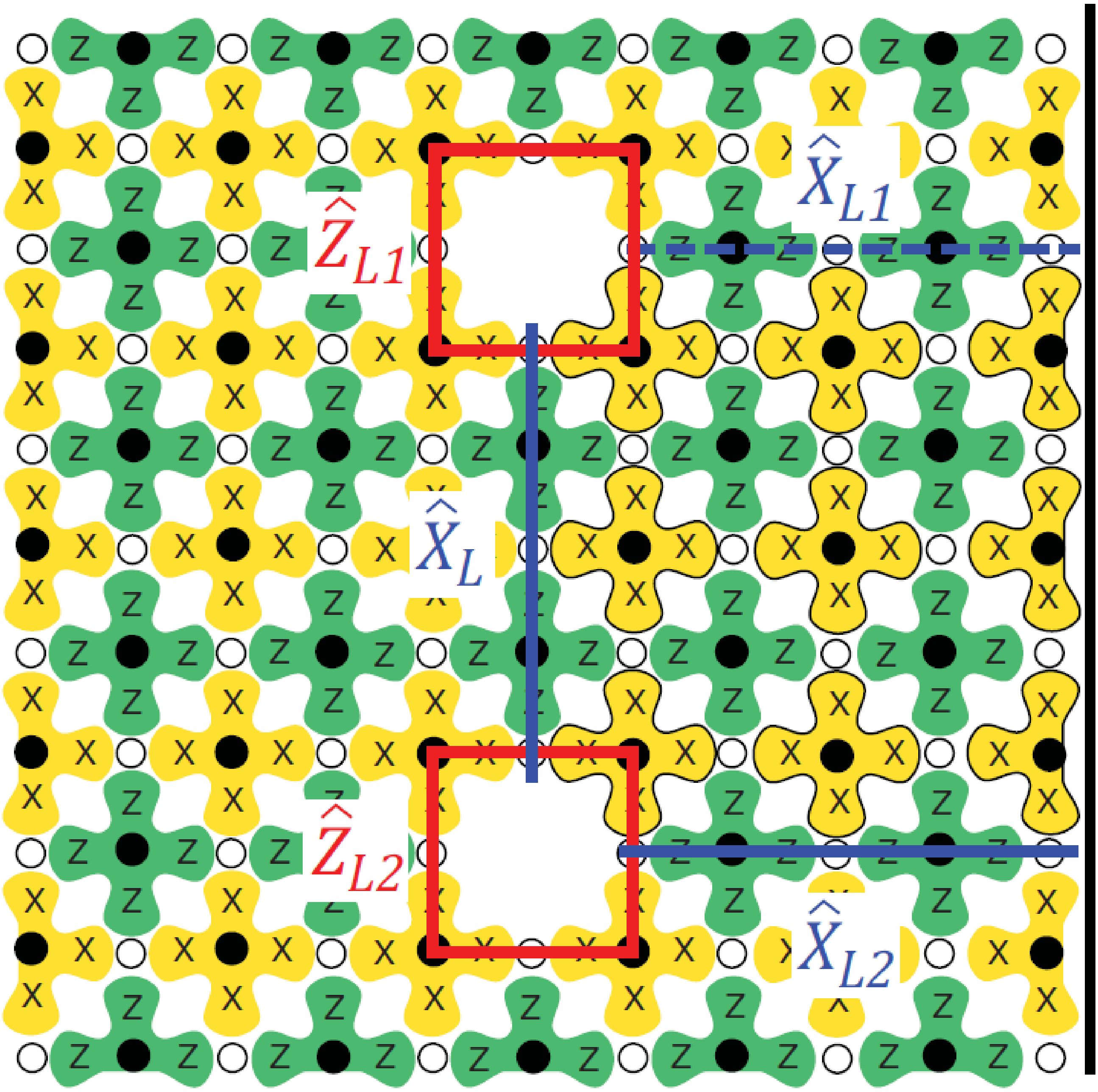
6. 其他 logical qubit 的表示方式

按照上面的表示方式，如果我们使用 $N \times N$ 的 data qubit 晶格表示一个 qubit，则每个逻辑算子均要同时作用在 N 个 physical qubits 上，因此我们提出了以下的表达方式，我们可以通过关闭某些 measure qubit 来在平面上形成“洞”，通过这些“洞”我们可以在局部实现逻辑算子，不过这样也就意味着对应的surface code 的距离 d 的减小。



这里我们关闭了一个 Z measure qubit，称为 Z-cut qubit。可以验证，分别验证图中蓝色线和红色线的 X-chain 和 Z-chain 是合法的逻辑算子。当然我们可以注意到，如果晶格的边界都是 Z-边界，则即使通过关闭中间的一个measure qubit也不能创造出新的自由度，这恰恰是因为此刻measure qubit的数量和data qubit的数量一样，没有多余的自由度。对于 X-cut qubit的情形是类似的。

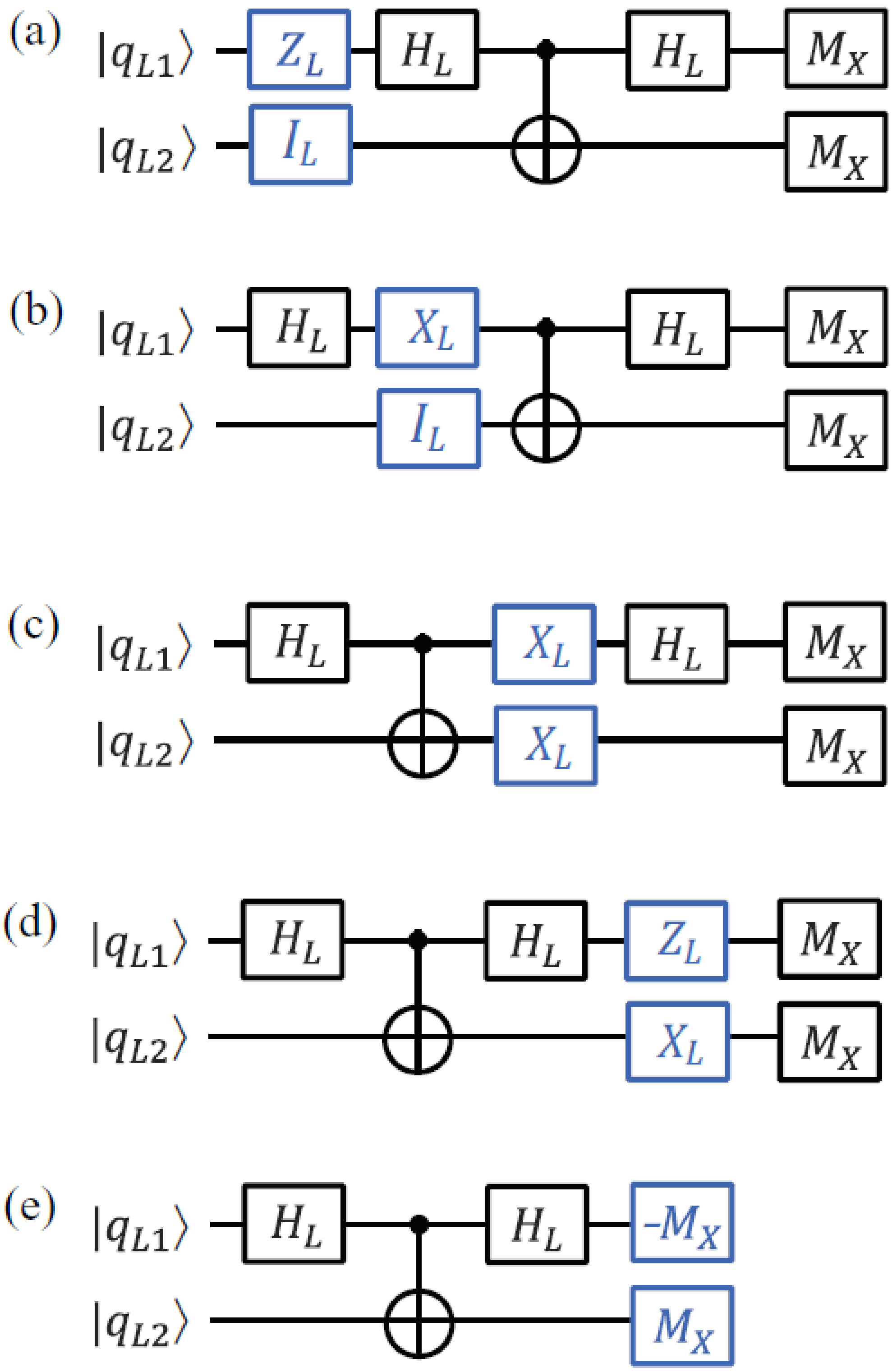
不过我们还有如下的方案，可以开两个“洞”，称之为 double Z-cut qubit，这当然带来了额外的 4 个自由度，不过我们只对在其中的两个自由度上的修改感兴趣（可以将其他两个自由度并入 $|Q\rangle$ 中），所以可以按照下图的线来定义逻辑算子。一般我们选择其中一个红圈来定义 Z_L，连接两个红圈的蓝线来定义 X_L。



这样的构造对于我们后续通过topological braid transformation实现 $CNOT$ 十分重要。

7. X_L 和 Z_L 的软件实现

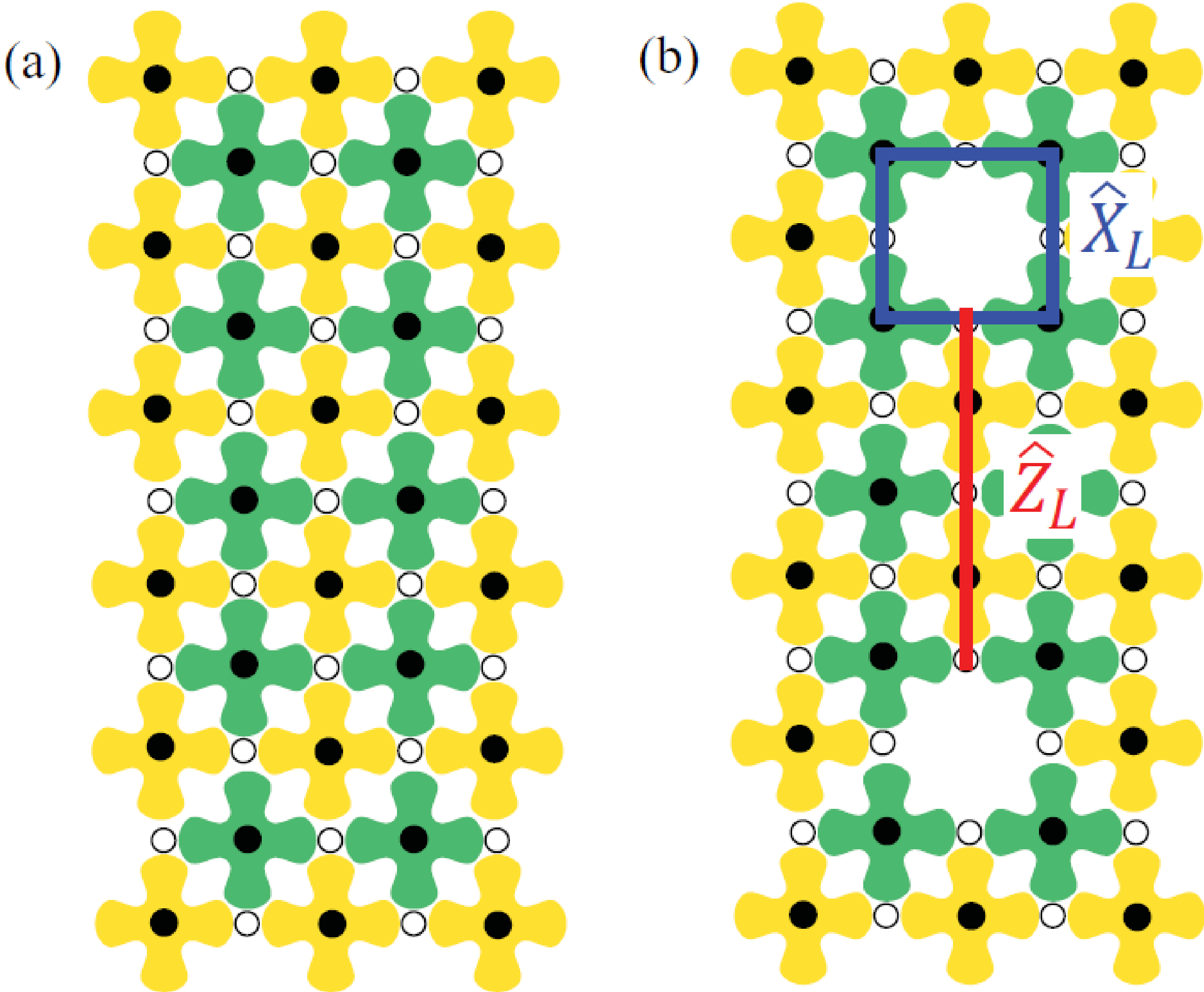
考虑到实际的量子电路中，一般只含有 $H, X, Z, CNOT$ 这些门，最后在每个 qubit 处进行测量。注意到 X, Z 算子和这些算子具有良好的交换性或者反交换性，所以我们可以不必真的对 qubit 实施X_L 或者 Z_L，而是可以通过软件记录这些操作，然后通过交换性将这些影响一直沿着电路传递到测量前，然后我们根据这些影响来矫正测量的结果。



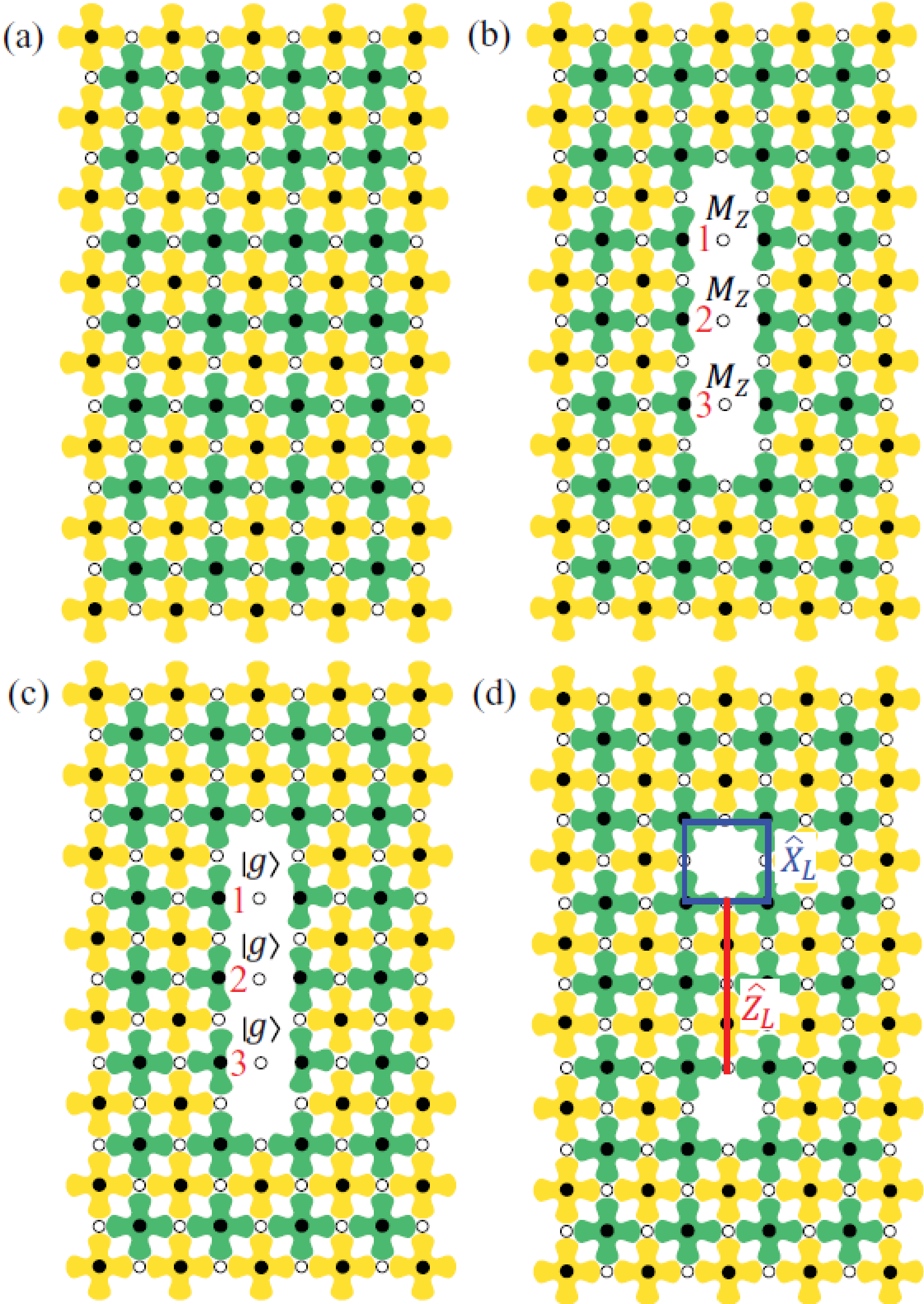
8. 初始化与测量

8.1 初始化

对于X-cut qubit，初始化又分为“难”和“易”，容易的是准备 $|+_L\rangle$ 和 $|-_L\rangle$ 状态困难的是准备 $|g_L\rangle$ 和 $|e\rangle$ 状态。前者只需要把每个data qubit置为$|+\rangle$ 或者 $|-\rangle$ 即可，或者采用下图的方式，先开启所有的 measure qubit，进行数轮测量，然后关闭两个 X measure qubit，则X_L 所围绕的 measure qubit 的上一次测量结果正是此时logical qubit的状态，$+1$ 对应 $|+_L\rangle$，-1 对应 $|-_L\rangle$。我们可以适当应用 Z_L 得到想要的状态。



后者的准备当然可以通过一个 Hadamard gate 实现，但是logical Hadamard gate 本身实现就比较复杂，下面我们介绍另一种方法。首先关闭部分的 measure qubit 如 (b) 所示，注意到 “洞”的边界上的 Z measure qubit 也只连接 3 个 data qubit，然后对这几个被孤立的 data qubit 进行 M_Z 测量，与其他活跃的 measure qubit 的测量一起进行，这主要是为了在初始化阶段保持纠错能力。然后将 3 个 data qubit 都设置为 $|g_L\rangle$，恢复部分的 measure qubit 如 (d) 所以，进行几轮测量后，则可以得到对应于 Z_L 特征值 $+1$ 的静止态。



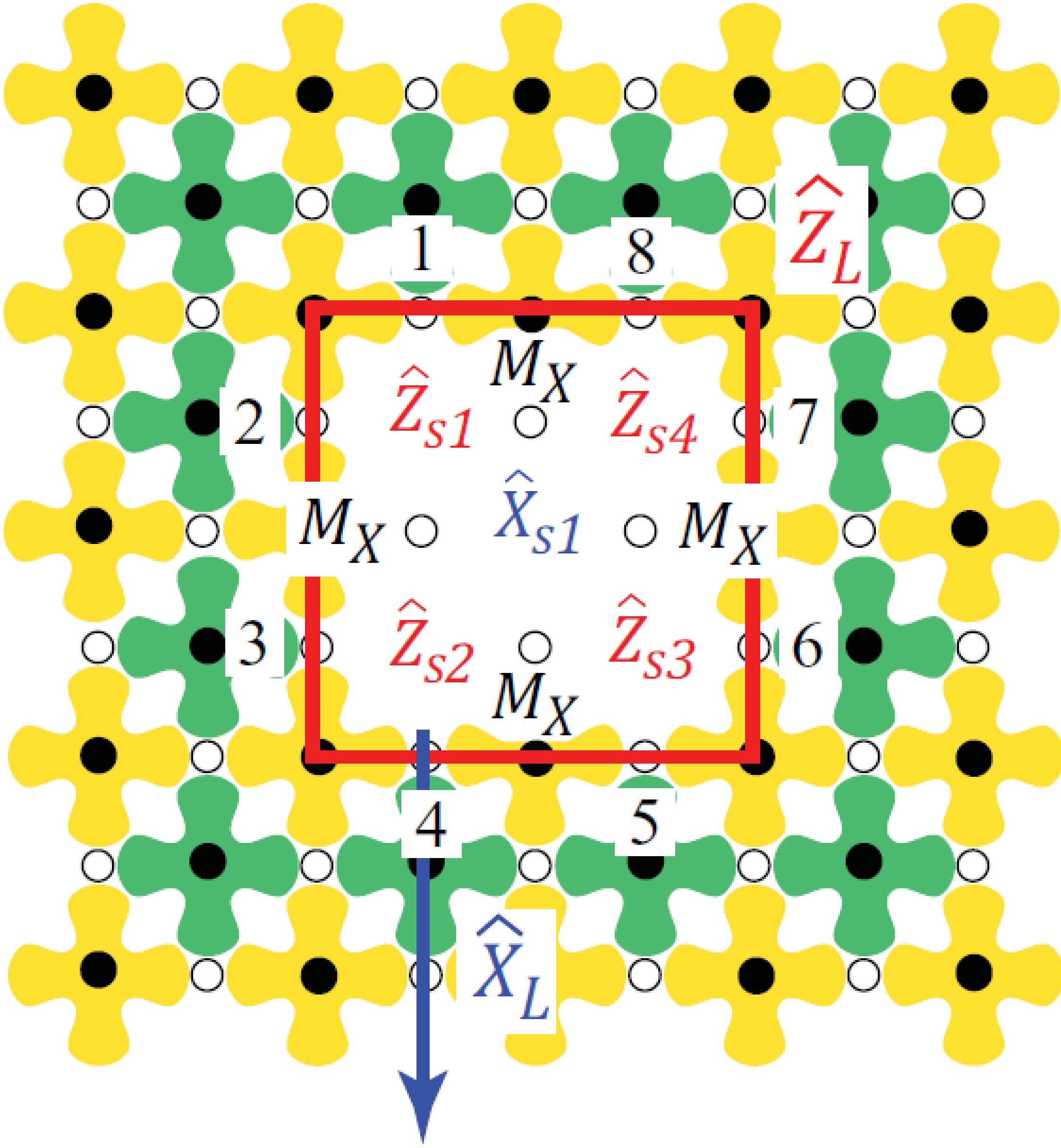
对于这个过程，可以有这样的解释：设置孤立的 data qubit 后，系统状态为 $|\phi\rangle = |ggg\rangle|\phi'\rangle$，是$Z_L$ 对应于特征值 $+1$ 的特征向量，但是却并非静止态。考虑到Z_L 和所有的 X measure operator 都是可交换的，所以它们的特征空间有一组共同的基，这样的话$|\phi\rangle$ 可以看作是一些静止态的叠加态，且这些静止态都对应于 Z_L 的$+1$ 特征值，进行一轮测量后会把系统状态投影到其中的一个静止态上。

8.2 测量

测量过程几乎就是初始化过程的“逆过程”，对应于测量的基也分为“难”和“易”。以 X-cut qubit 为例，容易的测量只需要开启X_L 所围绕的 X measure qubit，然后报告该处的测量结果即可，而困难的测量则需要再次如(b)一样孤立出 data qubit，然后分别进行 M_Z，最后恢复原状，进行测量，测量的结果就是 $Z_{1,2,3} = Z_1Z_2Z_3$。

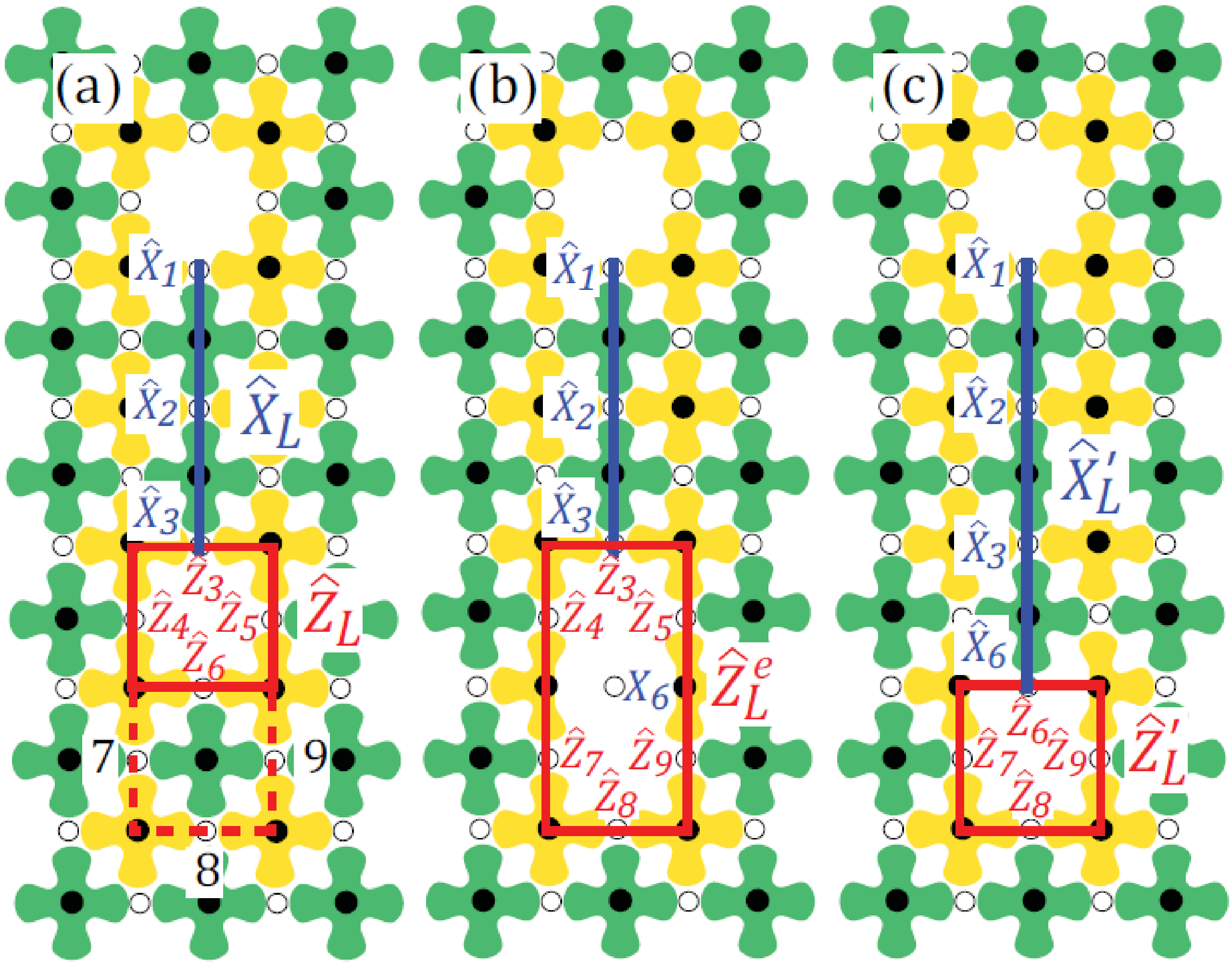
9. 更大的距离

通过制造更大的洞，我们可以增大距离d，从而提高surface code的抗干扰能力，而它的逻辑算子、初始化、测量，都可以仿照上面所说的进行。以下图为例，它将 d 从 4 增加到了 8。而在 Z_L 上的测量则可以通过对中间 4 个孤立qubit进行 M_Z 测量并且将结果相乘。



10. 移动 qubit

我们可以将移动 qubit 的过程分为两部分，一部分是物理上的操作，一部分是logical operator的变化。首先物理上，第一步首先进行一次测量，在下一次测量前关闭部分测量，如(b)所示，然后进行一轮测量，包括测量X_6，然后开启一部分测量，如(c)所示，再进行一轮测量即可。



注意到在这个过程中的logical operator的变化，从(a)到(b)的过程中，有如下变化：

$$
\begin{aligned}
Z_L &:= Z_3Z_4Z_5Z_6 \to Z_L^e := Z_3Z_5Z_9Z_8Z_7Z_4 \\
X_L &:= X_1X_2X_3 \to X_L' := X_1X_2X_3X_4
\end{aligned}
$$

在(b)到(c)的过程中，有如下变化：

$$
Z_L^e \to Z_L' := Z_6Z_7Z_8Z_9
$$

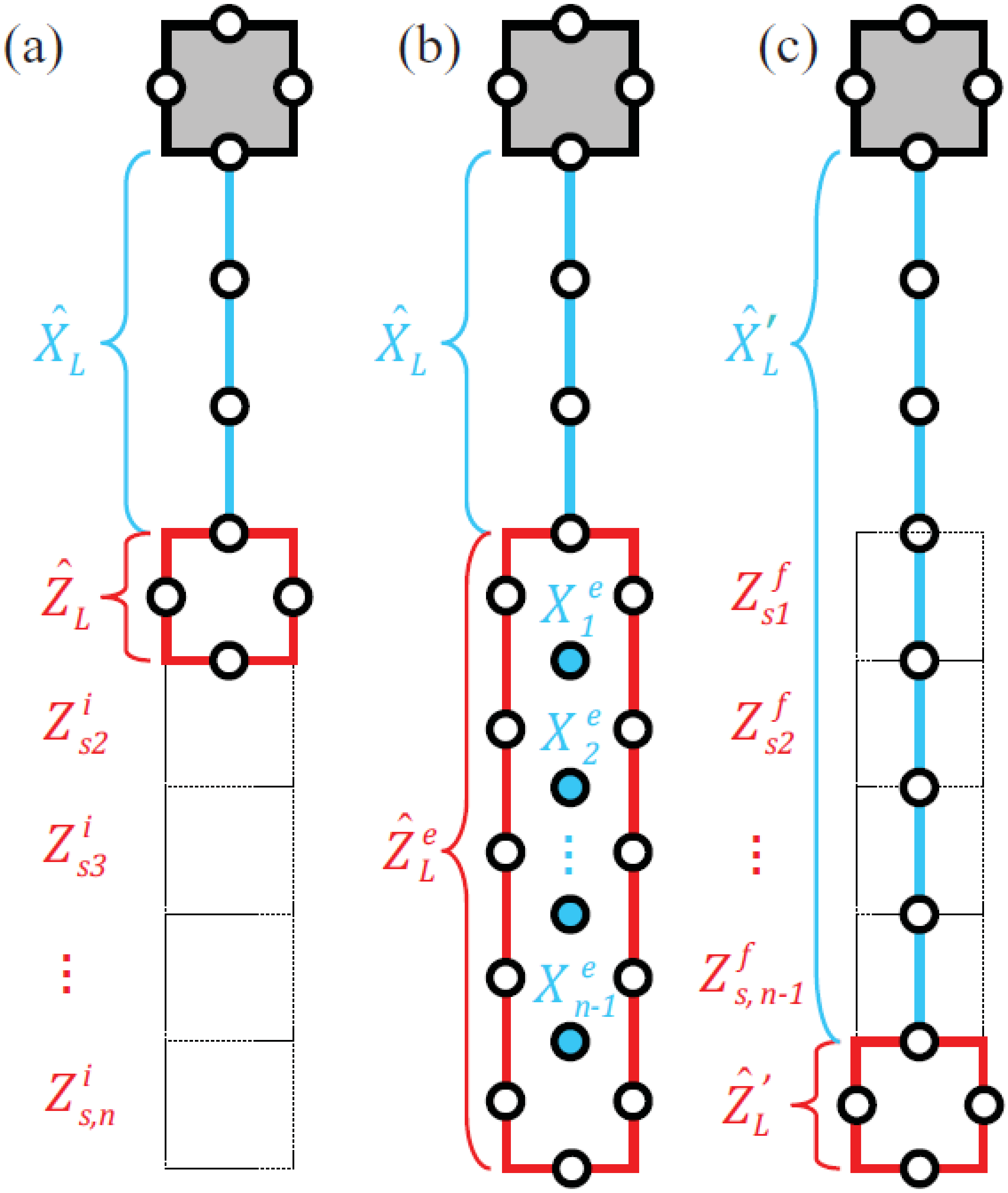
最后得到的 X_L' 和 Z_L' 就是新的系统上的逻辑算子。

但是注意到，如果 X_6 和变化中涉及到的两个 Z measure 结果的乘积不为 $+1$，则新得到的算子和原来的算子在符号上会有不同。这里我们可以采用 Heisenberg 的观点：对于变换 $|\phi\rangle \to U|\phi\rangle$，我们可以视为状态保持不变，但是所有作用在上面的operator A 有变换 $A \to U^\dagger AU$。但是反过来看，如果所有的operator都被一个 U 共轭作用了，那么可以看作operator不变，但是状态受到一个U 作用。这样的话，由于 $-X = Z^\dagger XZ, -Z = X^\dagger ZX$，我们可以看作是状态被如下作用：

$$
|\psi\rangle \to X_L'^{P_Z} Z_L'^{P_X} |\psi'\rangle
$$

其中 $|\psi'\rangle$ 是我们想要得到的状态，也即和原来的状态 $|\psi\rangle$ 相比在新的X_L定义下有相同的状态。

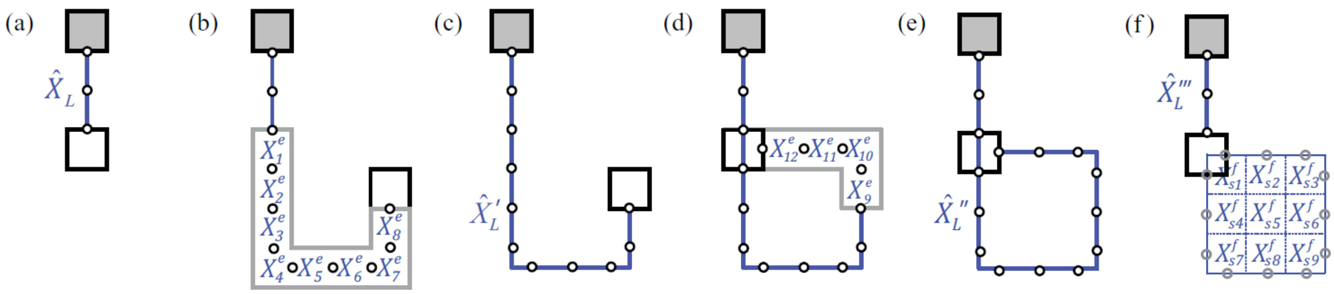
上面介绍了如何移动单个qubit，移动多个qubit和这个类似，逻辑算子也可以仿照上面进行改变，同时也要注意算子的符号变化和测量结果的关系：

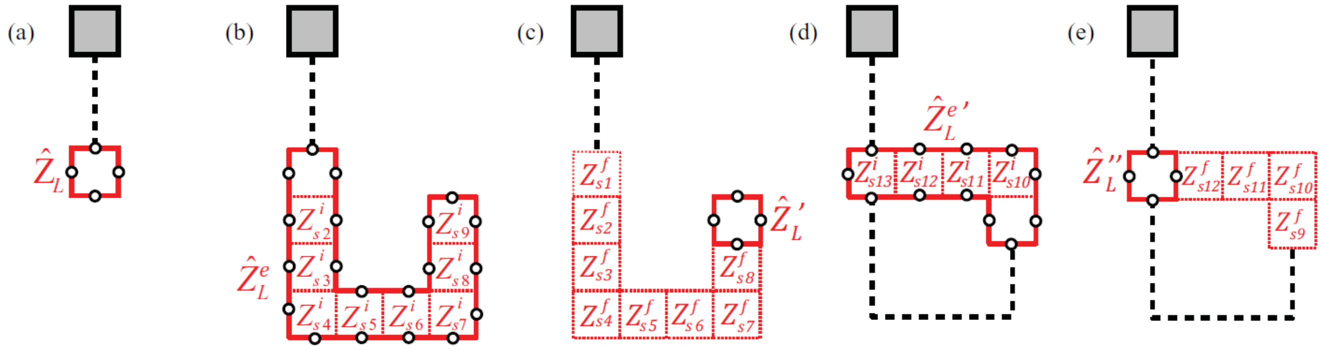


11. Topological braid transformation

11.1 单个qubit

有了上面定义的移动多个 qubit 的方法后，我们可以通过移动两次，来让一个 qubit 绕一个圈：

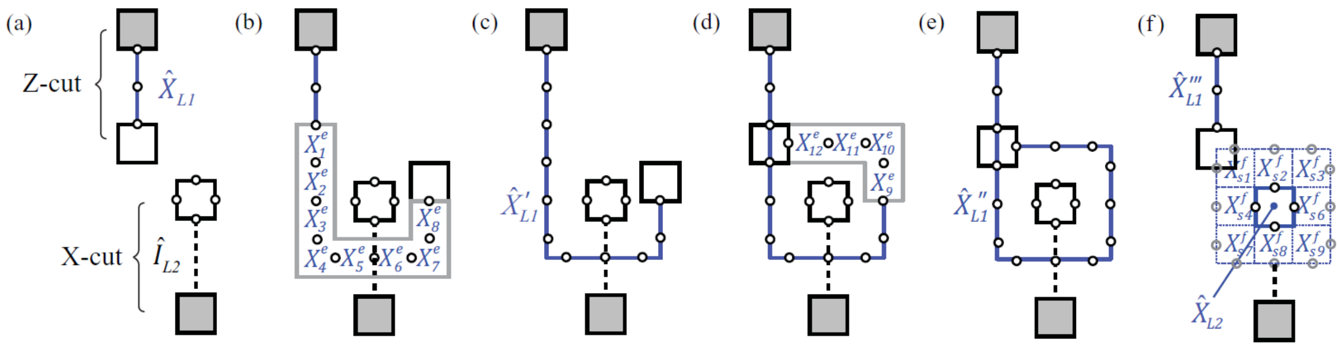


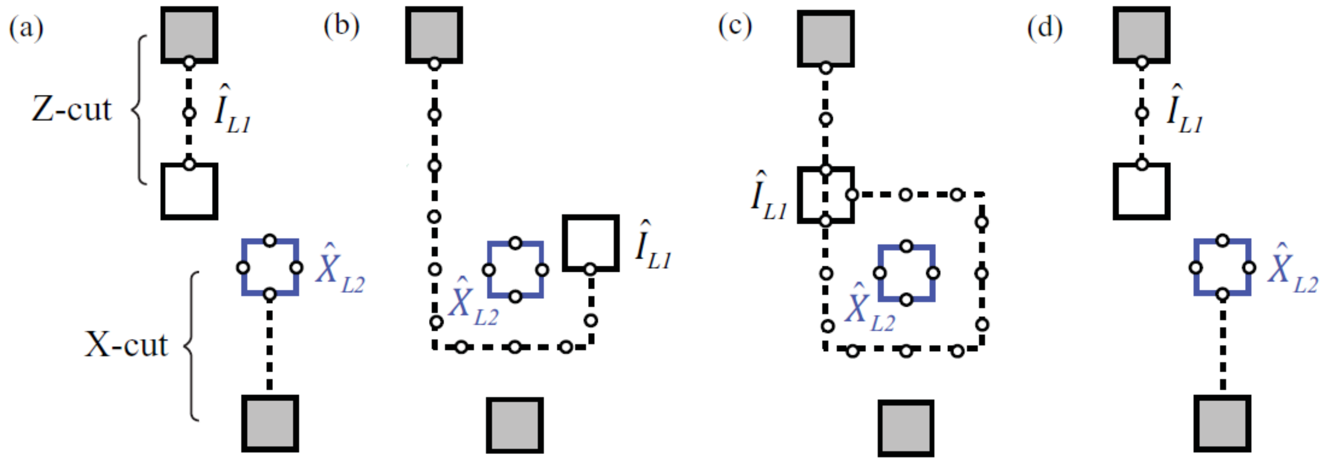


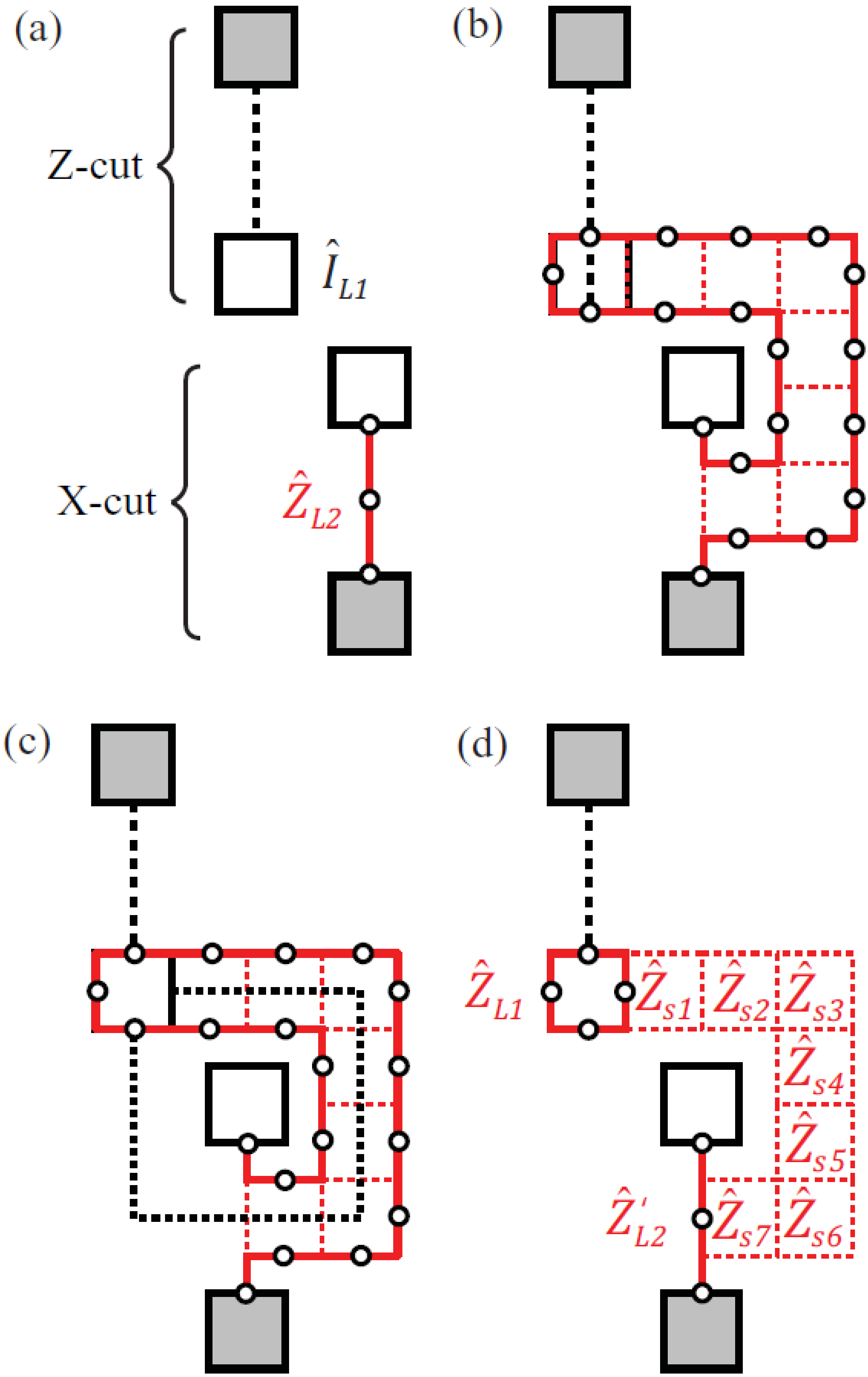
我们发现经过两步移动后，新的X_L 和 Z_L 和原来相比几乎没有改变，只是符号可能由于测量结果的不同而有所改变，我们需要把它表示成 logical qubit 的改变。

11.2 两个 qubit

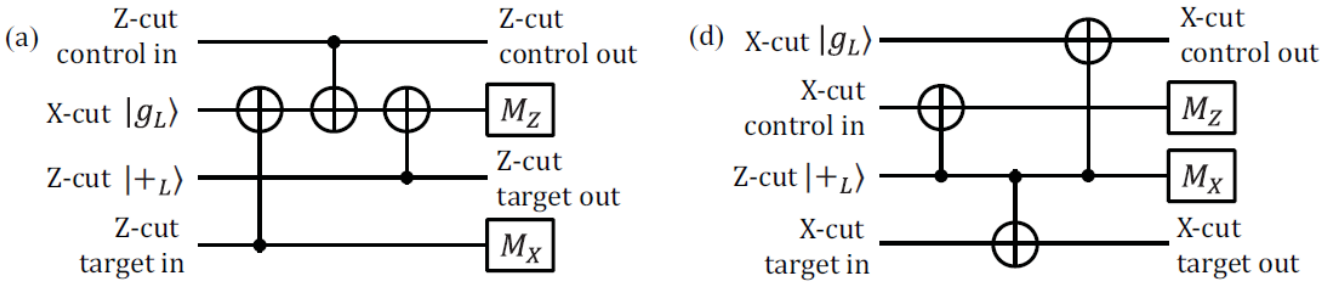
如果第一个logical qubit 是Z-cut，第二个是 X-cut，则我们令第一个logical qubit 按照下图的方式绕着第二个qubit一圈，我们将看到这样的操作，在不考虑逻辑算子的符号改变的情况下（我们可以通过对状态进行变换来修正这一点）与 $CNOT$ 是等价的。实际上，我们只需要检查该变换在 $I_{L,1}\otimes X_{L,2}, I_{L,1}\otimes Z_{L,2}, X_{L,1}\otimes I_{L,2}, Z_{L,1}\otimes I_{L,2}$ 上的作用即可，这并不困难，按图索骥即可：





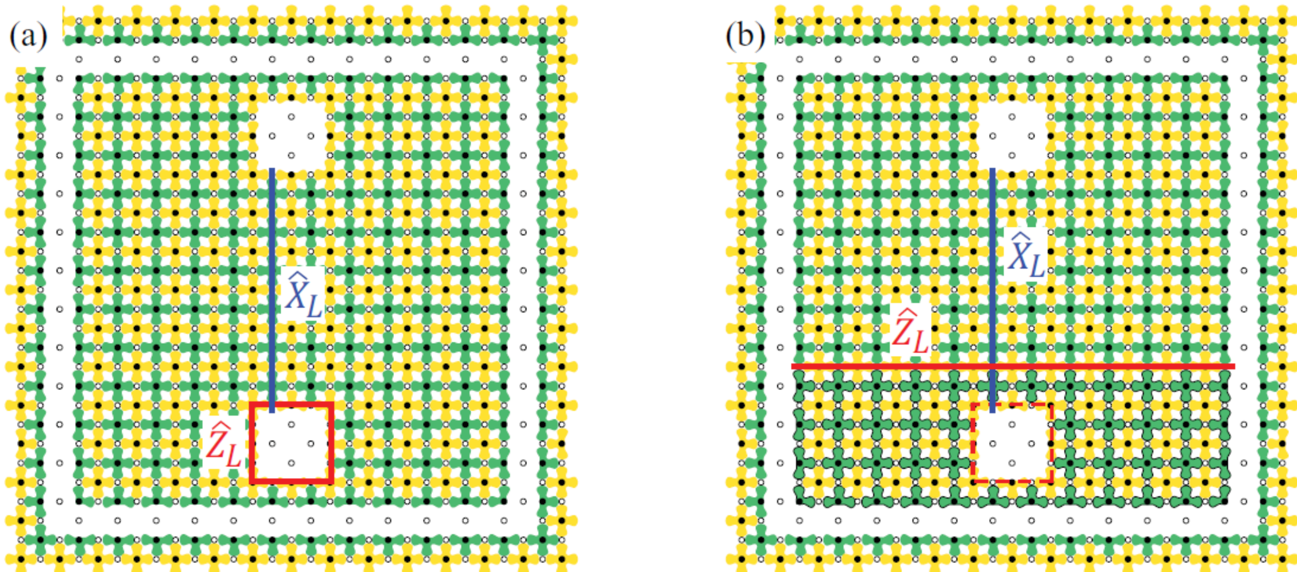


上面的操作虽然构成了$CNOT$，但是却要求作用在不同类型的logical qubit上，使用以下电路可以纠正这个问题：

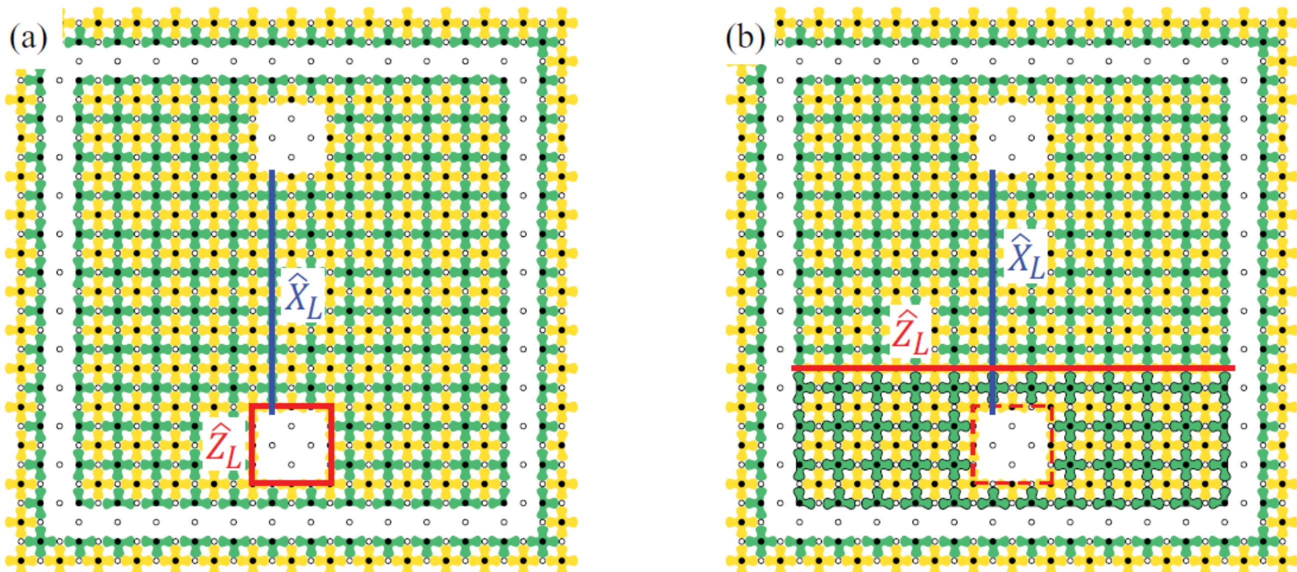


12. Hadamard gate

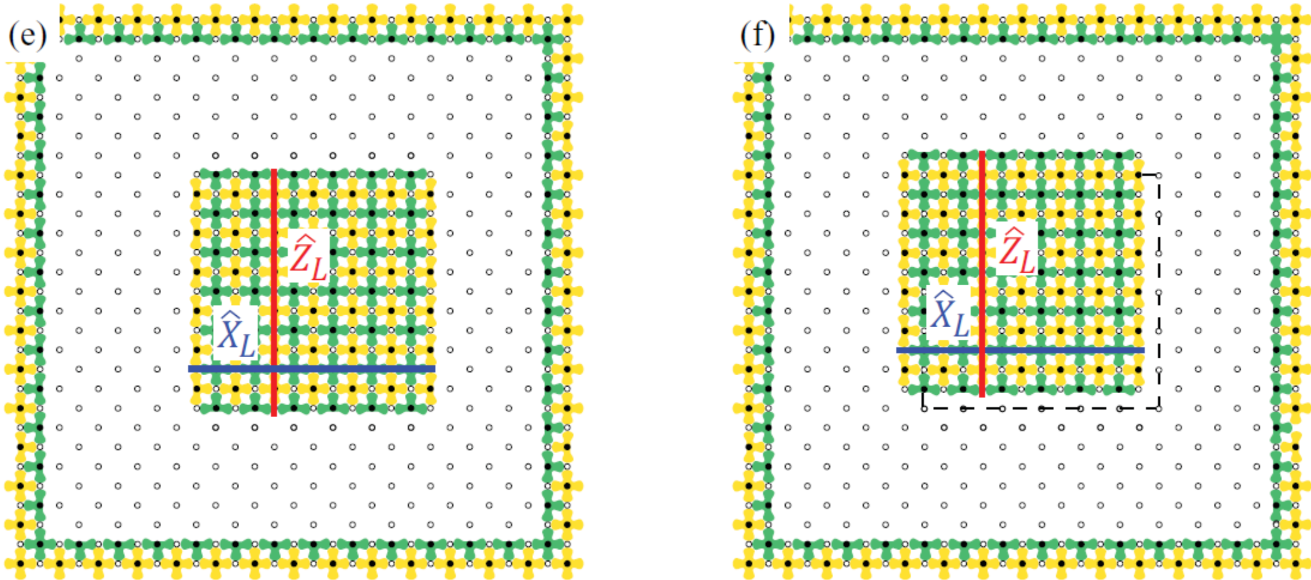
通过surface code实现Hadamard gate十分复杂，因此只是简述以下过程：首先通过关闭部分 measure qubit 隔离出要作用的 logical qubit，然后通过乘上C中的元素来改变Z_L 的表示（此处需要记录或者处理符号改变）；



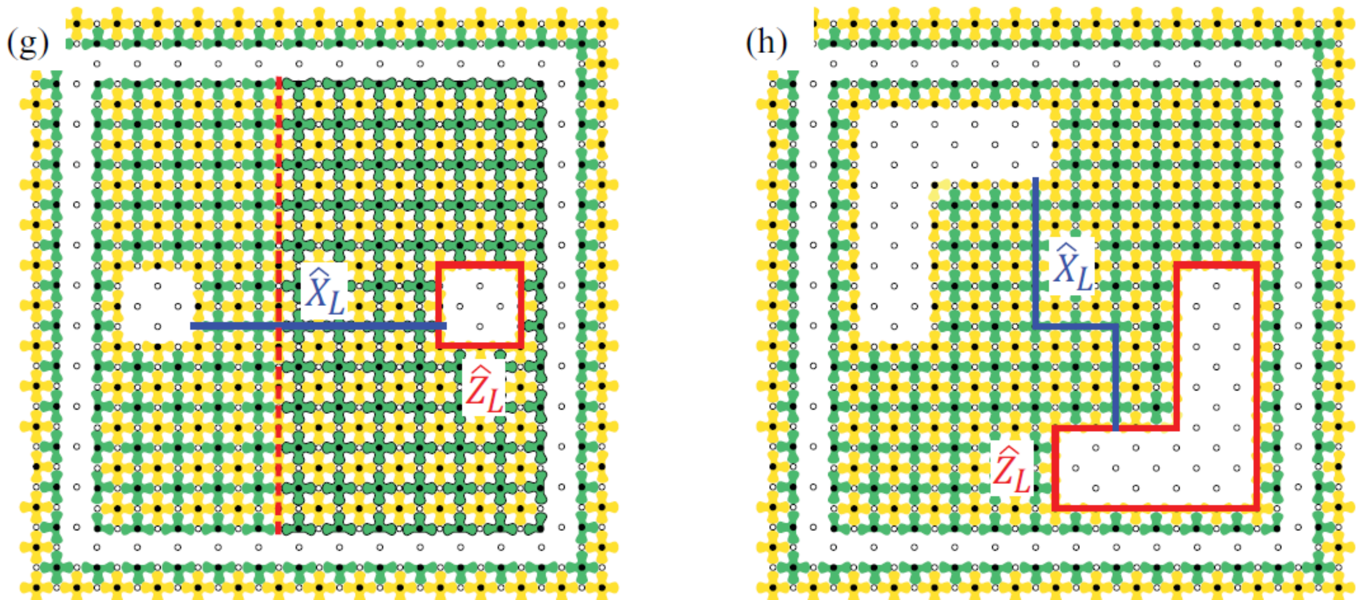
然后将黑色虚线框外的measure qubit也关闭，但是对于环中的data qubit还是需要进行M_Z来保证纠错功能；

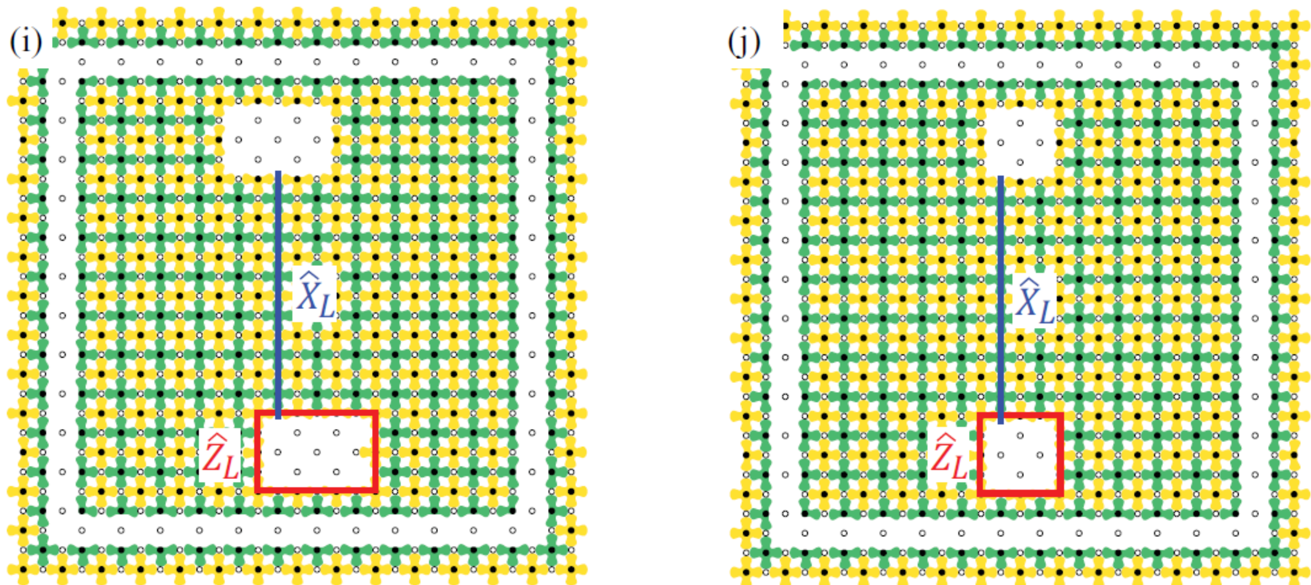


对框中的所有data qubit作用一个 Hadamard gate，这会导致 X_i 和 Z_i 交换，形成如(e)不能对齐的情况，但是可以通过和左上相邻的qubit进行swap操作来再次对齐；



然后就是逐步恢复刚刚被关闭的measure qubit，然后通过符号调整使得逻辑算子和作用前一样。





此外，如果可以制备一些特定的量子态，我们可以利用上面已经实现的算子来实现 S_L 和 T_L，而量子态的制备则可以使用称为 state distillation 的技术，详情可以参考：*Surface codes: Towards practical large-scale quantum computation*。

13. 一些变体

13.1 planar code

我们刚刚定义大部分操作都是基于带缺陷（X-cut 或者 Y-cut）的surface code，但实际上我们不需要缺陷也可以定义 $CNOT$ gate，只需要引入新的操作 merge 和 split。这里的记号与刚刚稍有不同，这里大个的格点表示data qubit，小个的格点表示measure qubit，其中位于节点处的是 Z measure qubit，位于面心的是 X measure qubit。

Lattice merging

对于两个分离的logical qubit，我们可以按照如下的方式将其粗糙边界（边界上是 X measure qubit）融合起来，形成一个新的logical qubit。为了方便起见，我们可以事先假设这两个logical qubit的X_L 定义在即将融合的边界上，Z_L 也在同一个水平线上。如果不是的话，可以事先做一些符号变换来得到这样的设置。

首先，将缝合处的新的 data qubit 设置为 $|g\rangle$，然后以下图的方式进行连接，得到一个新的logical qubit，然后经过一轮的measure。由于我们实现设定新的 data qubit 为 $|g\rangle$，所以所有的 Z measure 的结果并不会改变，与此同时我们记录缝合处的 X measure 的结果的乘积，可以看出，这样的操作等效于对 $X_{L,1} \otimes X_{L,2}$ 这个 observable 的测量。根据测量结果M，若融合的两个状态分别是$|\psi\rangle$ 和 $|\phi\rangle$，则测量后的结果是：

$$\frac{1}{\sqrt{2}}(|\psi\rangle|\phi\rangle + (-1)^M|\bar{\psi}\rangle|\bar{\phi}\rangle)$$

其中，$|\bar{A}\rangle = X_L|A\rangle$。

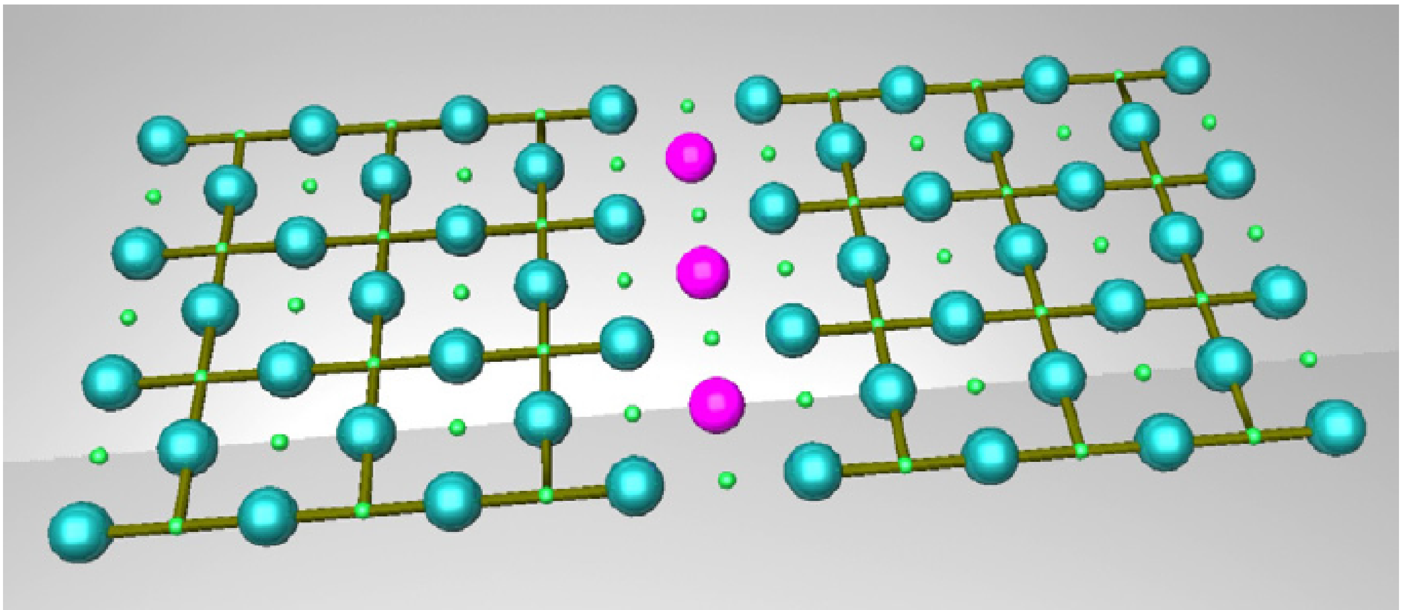
再考虑如何将融合后的状态映射回到融合前的，我们观察到新的 Z'_L 恰好是原来的 $Z_{L,1}, Z_{L,2}$ 的乘积，因而对应于融合后的 $|g_L\rangle$ ，必然对应融合前的 $|gg_L\rangle, |ee_L\rangle$ 的线性组合，同时考虑到新的 X'_L 定义为其中一个 logical qubit 原来的 X_L ，这这样的映射与测量结果 M 有关，具体来说是：

$$|g_L\rangle \to \frac{1}{\sqrt{2}}(|gg_L\rangle + (-1)^M|ee_L\rangle)$$
$$|e_L\rangle \to \frac{1}{\sqrt{2}}(|ge_L\rangle + (-1)^M|eg_L\rangle)$$

所以融合之后的效果等效于，令 $|\psi\rangle = \alpha|g_L\rangle + \beta|e_L\rangle$：

$$|\psi\rangle merge|\phi\rangle \to \alpha|\phi\rangle + (-1)^M\beta|\bar{\phi}\rangle$$

类似的，在光滑边界的融合效果为也可以写成同样形式，只不过在 $|+_L\rangle, |-_L\rangle$ 下分解。



Lattice spliting

这次我们考虑在光滑边界上做分割，这操作十分简单，我们先将两个平面分开，然后在连接处的data qubit执行 M_X，使得剩下的状态和这3个离开的data qubit 解除纠缠。若是考虑到分开前后 X_L 和 Z_L 的作用，我们有如下的映射：

$$\alpha|g_L\rangle + \beta|e_L\rangle \to \alpha|gg_L\rangle + \beta|ee_L\rangle$$

$CNOT$

假设 $|C\rangle$ 是控制 qubit，$|T\rangle$ 是目标 qubit，$|INT\rangle$ 是辅助 qubit，初始化为 $|+_L\rangle$。首先让$|T\rangle$ 和

$|INT\rangle$ 做一个光滑的融合，得到：

$$\begin{aligned}|C\rangle merge|INT\rangle &\to \bar{\alpha}|+_L\rangle + (-1)^M\bar{\beta}|+_L\rangle \\ &= \frac{1}{\sqrt{2}}(\bar{\alpha} + (-1)^M\bar{\beta})|g_L\rangle + \frac{1}{\sqrt{2}}(\bar{\alpha} - (-1)^M\bar{\beta})|e_L\rangle\end{aligned}$$

根据测量结果M适当地做bit flip，可以得到 $|C\rangle merge|INT\rangle \to \alpha|g_L\rangle + \beta|e_L\rangle$。

接下来将二者分开，则有：

$$|C'INT\rangle = \alpha|gg_L\rangle + \beta|ee_L\rangle$$

然后让 $|INT\rangle$ 和 $|T\rangle$ 做一个粗糙融合，得到：

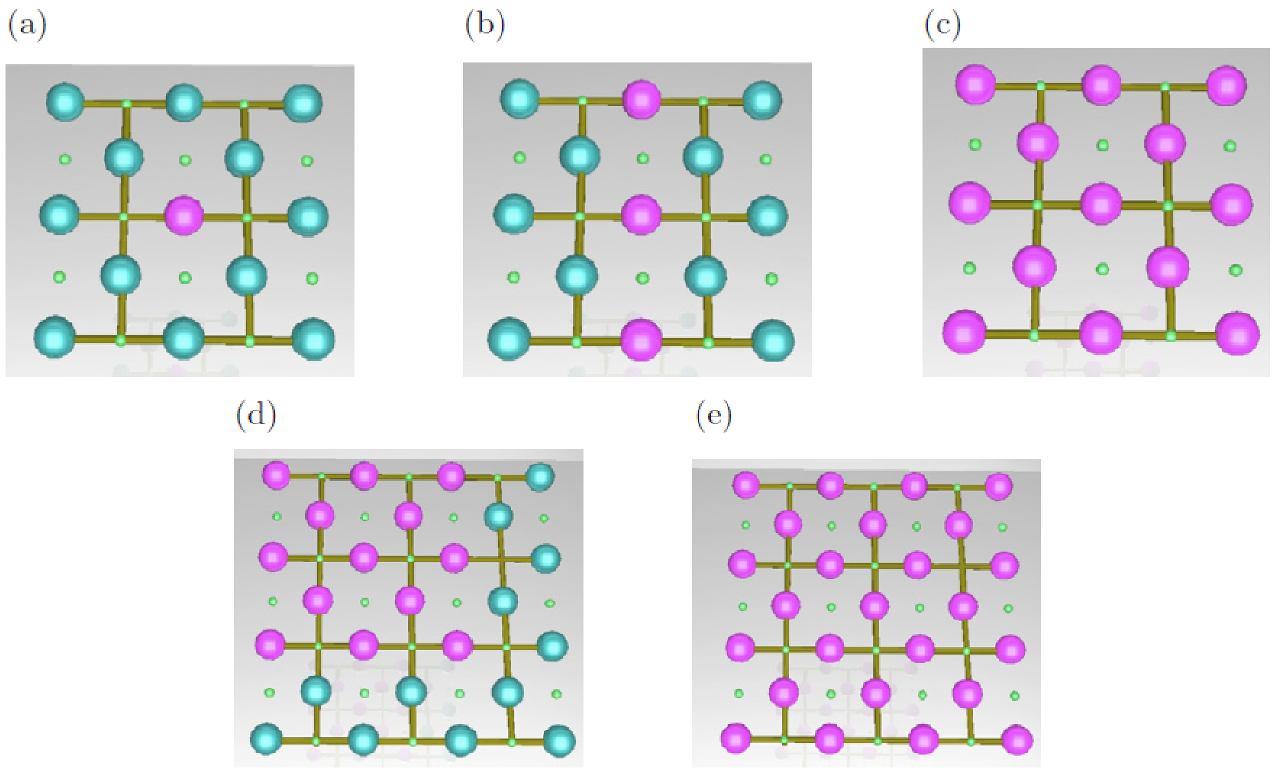
$$|C'INT'\rangle merge|T\rangle \to \alpha|g_L\rangle|T\rangle + (-1)^{M'}|e_L\rangle|XT\rangle$$

这里的 $(-1)^{M'}$ 系数可以使用一个 $Z_{L,1}$ 来进行纠正。

于是，我们通过一个辅助的qubit，使用融合与分裂操作，实现了 $CNOT$ gate。

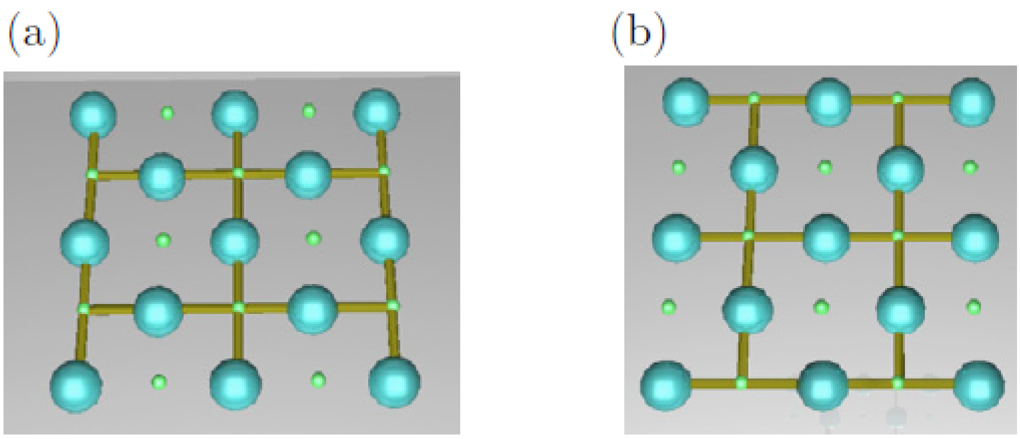
状态注入

只要我们能够将一个physical qubit初始化为任意状态，我们就可以将planar code编码的logical qubit编码为任何一种状态，这种技术被称为“状态注入”。如下图所示，(a)中先将中间的 data qubit 初始化为 $\alpha|g\rangle + \beta|e\rangle$，其余初始化为 $|g\rangle$；(b)中使用$CNOT$ gate 和上下相邻的 data qubit形成纠缠态 $\alpha|ggg\rangle + \beta|eee\rangle$；注意到此时的状态在$Z_L$特征空间下的分解，利用和之前的困难初始化类似的分析，我们知道在(c)中开启所有 measure qubit 进行一轮测量，最后得到某一个静止态，并且对应于该测量结果下的$\alpha|g_L\rangle + \beta|e_L\rangle$；如果需要扩大，只需要像(d)(e)那样把额外的measure qubit和data qubit连接上来，全部初始化为 $|g\rangle$，然后进行一轮测量。

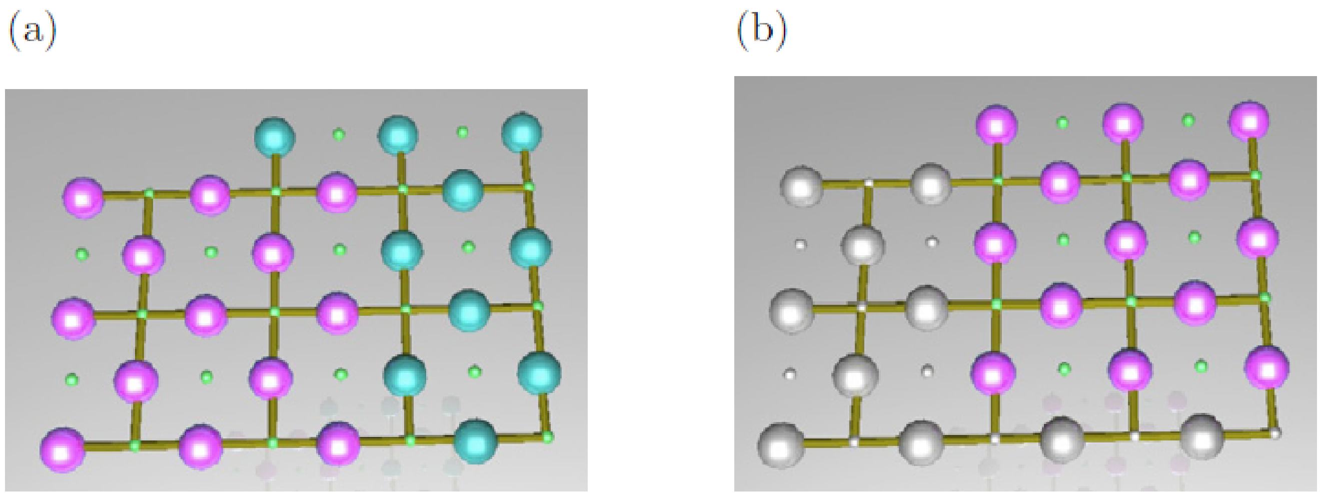


Hadamard gate

单纯对所有的data qubit采用一个Hadamard 变换将会让我们得到正确的结果，但是由于X 和 Z 的对换，我们发现planar code 旋转了 90° ，如果物理实现上这些晶格可以移动，那么只需要再旋转回来。



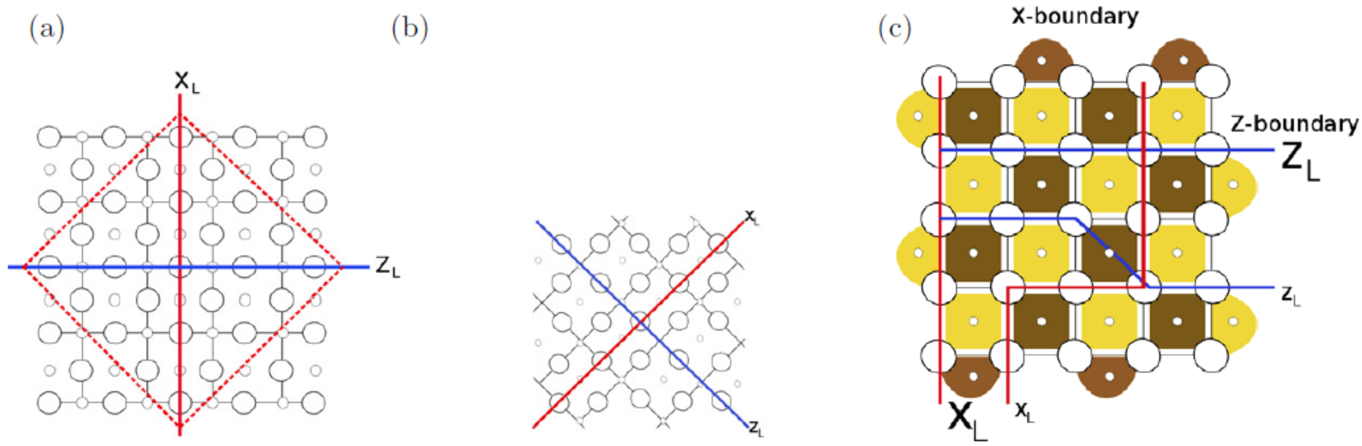
否则的话，采用下图的方法：先将一些data qubit和其拼接形成一个更大的planar code，把多余部分通过 M_Z 去掉，最后用 $SWAP$ 把新得到的方向正确的planar code平移回原来的位置。



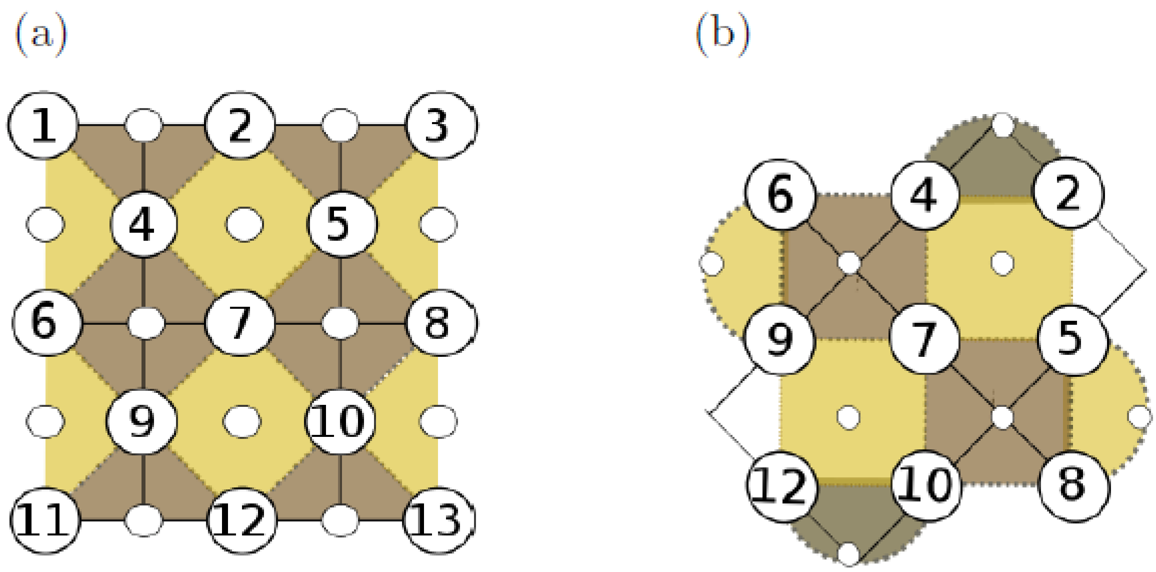
> remark：planar code和defeat-based surface code之间可以相互转化，这点我们前文中的defeat-based surface code的Hadamard gate实现中正是利用了这一点，先将后者转化为前者，实现Hadamard gate，然后转化回后者。

简化

下图展示了一个 $d=5$ 的planar code，可以对其简化来得到一个包含更少 data qubit 但 d 不变的planar code。裁剪掉红色边框外的所有 data qubit 和 measure qubit，除了那些和边框上的 data qubit 有连接的 measure qubit。可以明显看出，新的planar code的距离没有改变。



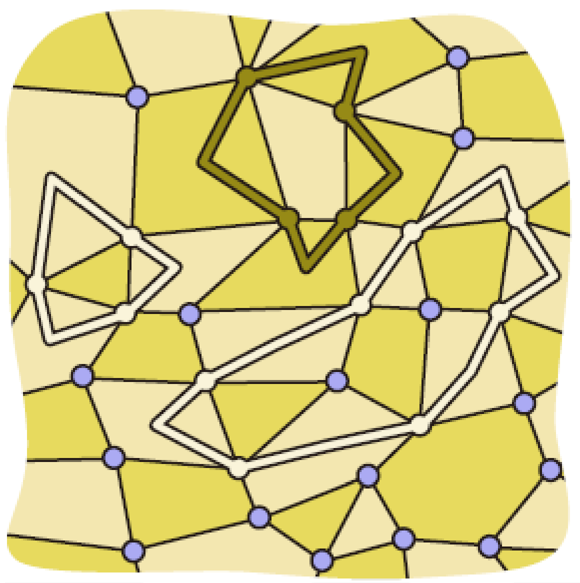
另一个例子：



Standard lattice stabilizers	Rotated lattice stabilizers
$X_2X_7X_{12}$ $(= X_L)$	$X_2X_7X_{12}$ $(= X_L)$
$X_1X_2X_4$	
$X_2X_3X_4$	X_2X_4
$X_4X_6X_7X_9$	$X_4X_6X_7X_9$
$X_5X_7X_8X_10$	$X_5X_7X_8X_{10}$
$X_9X_{11}X_{12}$	
$X_{10}X_{12}X_{13}$	$X_{10}X_{12}$
$Z_1Z_4Z_6$	
$Z_2Z_4Z_5Z_7$	$Z_2Z_4Z_5Z_7$
$Z_3Z_5Z_8$	Z_5Z_8
$Z_6Z_9Z_{11}$	Z_6Z_9
$Z_7Z_9Z_{10}Z_{12}$	$Z_7Z_9Z_{10}Z_{12}$
$Z_8Z_{10}Z_{13}$	

13.2 color code

接下来，我们不妨换一种方式来叙述surface code，这也能帮助我们更好地看出它与color code的联系。我们可以认为surface code可以由四价格点在某个表面上的可以二着色的排布，因此我们可以将着色之后的面分为亮面和暗面，分别记做P_L和P_D，如下图所示：



(a)

有如下记号：

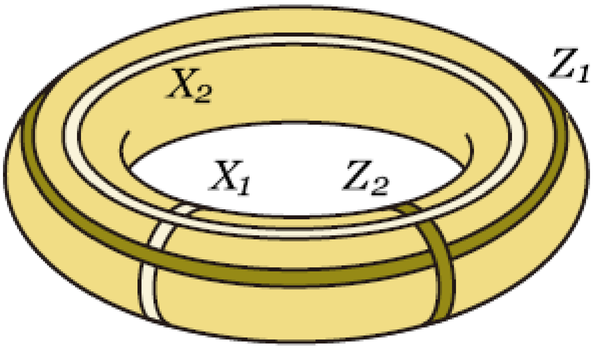
$$B_p^\sigma := \otimes_i \sigma_i^{s_p(i)}, \sigma = X, Z$$

其中 $s_p(i)$ 是面 p 的边界点的指示函数。则 surface code 的对应的 C 的生成元集合为：

$$\{B_p^X | p \in P_D\} \cup \{B_p^Z | p \in P_L\}$$

一条 Z-string 由一连串的格点表示，它在每个这些格点上都作用上 Z，同时要保证每个 $p \in P_D$ 含有偶数个格点；X-string 也可以类似定义。任何的operator都可以表示为一条 Z-string 和一条 X-string 的乘积，若不考虑相位的话。由相同类型的色块对应的生成元的乘积所产生的 σ-string 称为 **边界**，那些不是 **边界** 的 **闭合** 的 σ-string 构成了那些无法被发现的错误（我们认为起始点和终点在表面边界上的也是闭合的，这真是我们之前看到的surface code的情形），我们也可以从中选取logical qubit 的算子。

我们主要考虑不同编码的效率，考虑最早的 torus code，一个 g-torus 可以编码 $2g$ 个logical qubit，如下图所示，注意不同类型的 string 相交奇数次表示它们反交换，偶数次表示它们交换：

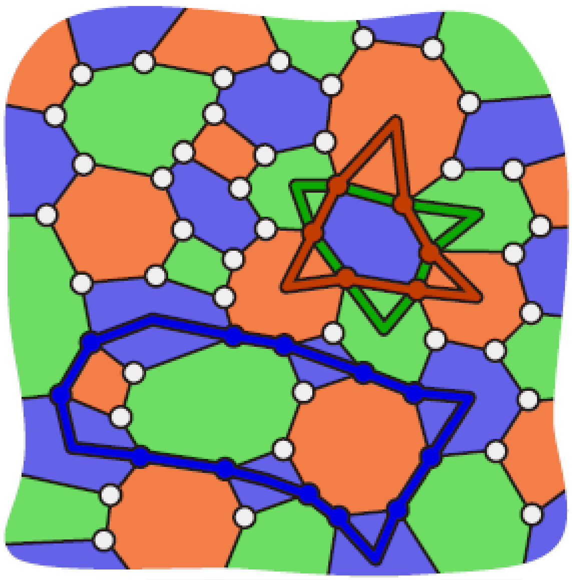


(a)

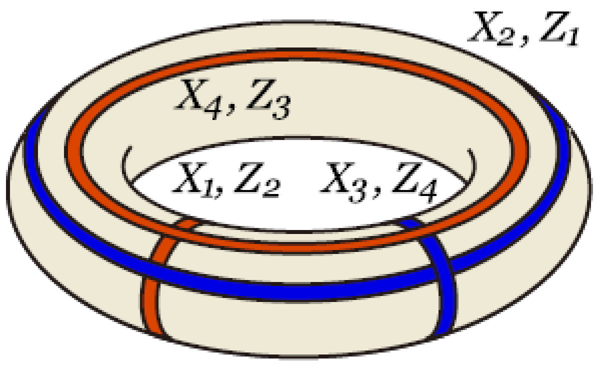
color code 则是考虑由三价格点在某个表面上的可以三着色的排布，色块可以分为 P_G, P_R, P_B，它的生成元集合为：

$$\{B_p^\sigma | \sigma = X, Z, p \in P\}$$

这里的 string 定义除了类型（X, Z），还需要考虑颜色，比如说蓝色的 string 只能连接蓝色的色块，意味着它可以在蓝色色块中间穿过，而只能沿着其余两种颜色的色块的边沿。此外，关于 **边界**、**闭合** 之类的讨论和之前类似。当然也可以直接考虑算符对于每个生成元的测量结果来直接推出logical qubit 的逻辑算子。

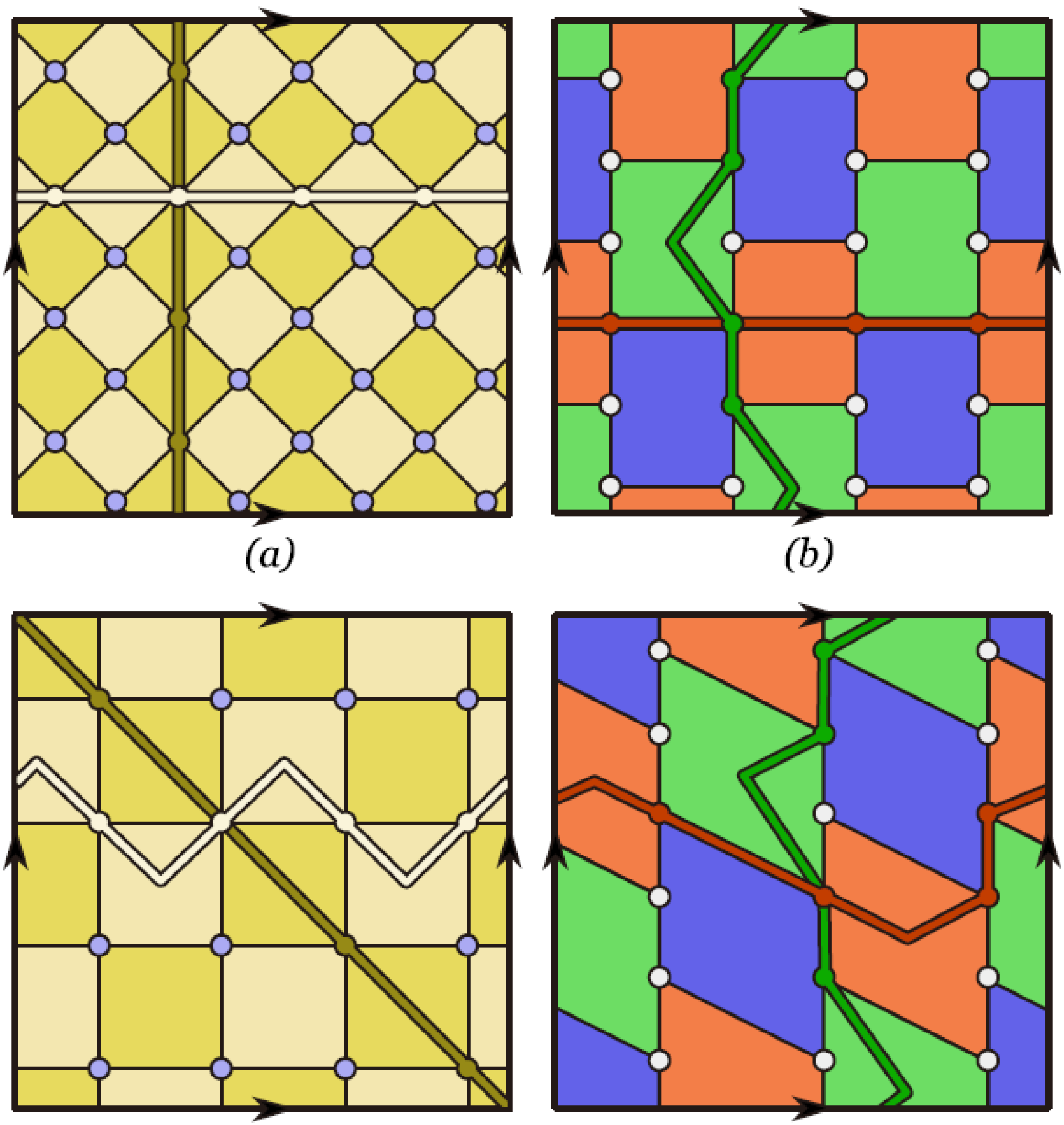

(b)

若是采用 color code，一个 g-torus 可以编码 $4g$ 个 logical qubit：



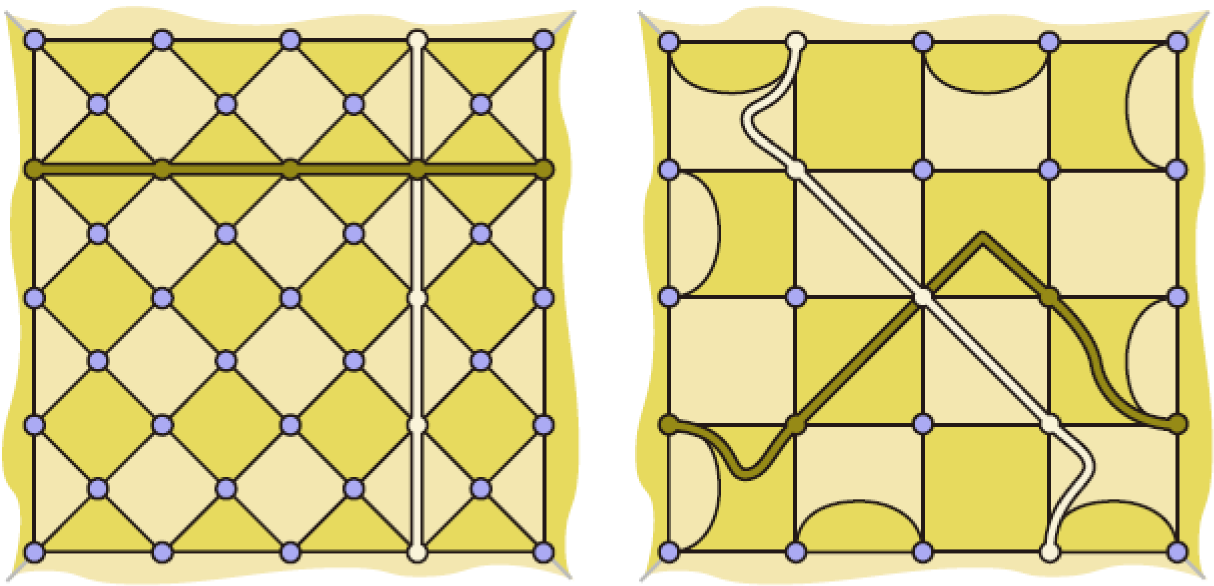
(b)

下面是一个实际的例子，展示了$d = 4$ 的情况下，普通的surface code 和 color code 在一个圆环上的排布，可以看到在前两个正规排布下 surface code 的 $C_s := \frac{n_s}{d_s^2}$ （纠错率）要大于 color code，但是编码的logical qubit 数量却不如；(c)(d) 分别展示了优化后的情形，此时 d 不变，二者具有差不多的纠错率。

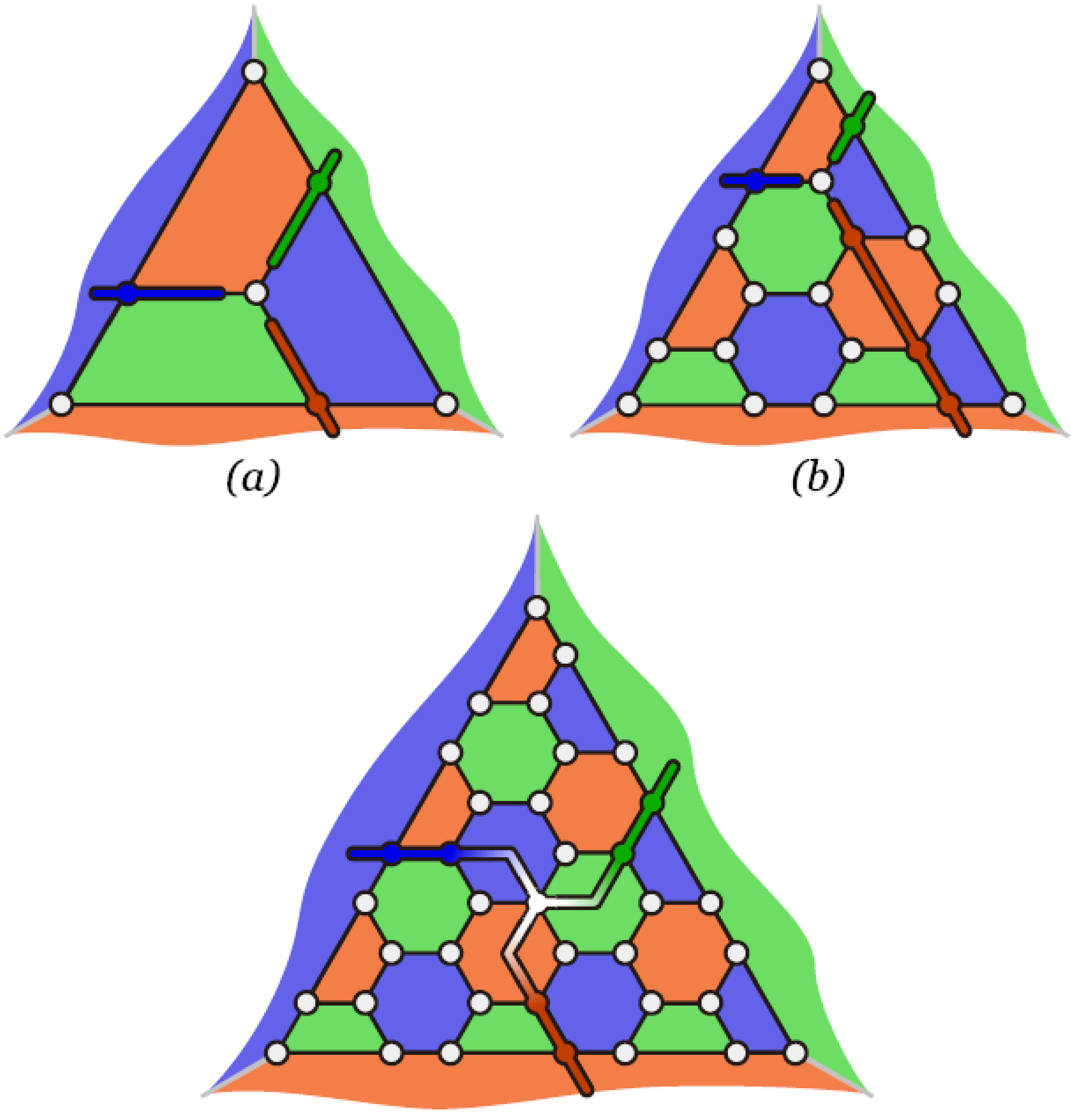


(a) (b) (c) (d)

我们刚刚讨论的都是闭合曲面，如果要得到 planar code，那么只需要不断地移除色块直到表面可以被展开到平面即可，这样的定义就和之前相同了，为了使string的定义相容，可以假设产生的边界也表示一个色块。



(a) (b)



(a) (b) (c)