# Shrink

```
# attempt to shrink the stabilizer state by eliminating a part
# of the basis that has inner product α with vector ξ
def shrink(self, xi, alpha, lazy=False):
    # S <- { a ∈ [k] : (ξ, g) = 1 }
    # Note that a is zero-indexed.
    S = [a for a in range(self.k) if np.inner(self.G[a], xi) % 2 == alpha]

    beta = (alpha + np.inner(xi, self.h)) % 2
    if len(S) == 0 and beta == 1: return "EMPTY"
    if len(S) == 0 and beta == 0: return "SAME"

    i = S[0]  # pick any i ∈ S
    S.remove(i)

    for a in S:
        # g^a ← g^a ⊕ g^i
        # compute shift matrix for G
        shift = np.concatenate((np.zeros((a, self.n)), [self.G[i]],
                                np.zeros((self.n - a - 1, self.n))))
        self.G = (self.G + shift) % 2

        # update D, J using equations 48, 49 on page 10
        # compute k*k basis change matrix R (equation 47)
        if not lazy:
            R = np.identity(self.k)
            R[a, i] = 1
            self.updateDJ(R)

        # ḡ^i ← ḡ^i + Σ_a ḡ^a
        self.Gbar[i] += self.Gbar[a]
    self.Gbar = self.Gbar % 2

    # swap g^i and g^k, ḡ^i and ḡ^k
    # remember elements are zero-indexed, so we use k-1
    self.G[[i, self.k-1]] = self.G[[self.k-1, i]]
    self.Gbar[[i, self.k-1]] = self.Gbar[[self.k-1, i]]

    # update D, J using equations 48, 49 on page 10
    if not lazy:
        R = np.identity(self.k)
        R[[i, self.k-1]] = R[[self.k-1, i]]
        self.updateDJ(R)

    # h ← h ⊕ β · g^k
    self.h = (self.h + beta*self.G[self.k-1]) % 2

    if not lazy:
        # update Q, D using equations 51, 52 on page 10
        y = np.zeros(self.k)
        y[self.k-1] = beta
        self.updateQD(y)

        self.J = self.J[1:, 1:]  # remove last row and column from J
        self.D = self.D[1:]      # remove last element from D

    self.k -= 1

    return "SUCCESS"
```