- **Función *matgencaniter*(r,m)**. Dados dos enteros r y m tales que 0 ≤ r ≤ m, devuelve la matriz generadora de $\mathcal{RM}$(r,m) calculada a través de la **Proposición 9** para el orden canónico de $\boldsymbol{F}_2^m$.

In[1]:=

```
matgencaniter[r_, m_] :=
 Module[{lista = {}, i, j}, For[i = 0, i ≤ m, i++, For[j = 0, j ≤ i, j++,
 módulo                     para cada              para cada
    If[j == 0, lista = AppendTo[lista, Tuples[{1}, 2^i]], If[i == j, lista =
                       añade al final      tuplas                 si
       AppendTo[lista, Join[lista[[-1]], Join[Tuples[{0}, 2^i - 1], {{1}}, 2]]];
                       junta               junta  tuplas
      lista = Delete[lista, 1], lista = AppendTo[lista,
              elimina                    añade al final
        Join[Join[lista[[2]], ConstantArray[0, {Length[lista[[1]]],
             junta            arreglo constante      longitud
            Length[lista[[2]][[1]]]}]], Join[lista[[2]], lista[[1]]], 2]];
                                                 junta
      lista = Delete[lista, 1]]];
              elimina
    If[i == m && j == r, Return[lista[[-1]]]]]]]
                         retorna
```

- **Función *fostandar*(r,m)**. Dados dos enteros r y m tales que 0 ≤ r ≤ m, devuelve la forma estándar de la matriz obtenida comenzando con la que genera ***matgencaniter*(r,m)**, siguiendo los pasos dados en la **Proposición 13**, así como el orden estándar de $\boldsymbol{F}_2^m$ correspondiente.

In[2]:=

```
formordstandar[r_, m_] := Module[
                          módulo
  {G = matgencaniter[r, m], n, s, i = 1, j, k, H, aux, A, lista}, n = Length[G[[1]]];
                                                                    longitud
  s = Length[G];
      longitud
  A = G;
  lista = Tuples[{0, 1}, Log[2, n]];
          tuplas         logaritmo
  While[i ≤ s, If[A[[i, i]] == 1, For[j = 1, j ≤ s, j++,
  mientras        si               para cada
     If[A[[j, i]] == 1 && i ≠ j, A[[j]] = Mod[A[[j]] - A[[i]], 2]]];
     si                                    operación módulo
    i++, H = Transpose[A];
               transposición
    For[k = i, k ≤ n , k++, If[A[[i, k]] == 1, aux = H[[k]];
    para cada                  si
      H[[k]] = H[[i]];
      H[[i]] = aux;
      aux = lista[[k]];
      lista[[k]] = lista[[i]];
      lista[[i]] = aux;
      A = Transpose[H];
          transposición
      k = n + 1]]]];
  Return[{A, lista}]]
  retorna
```

- **Función *codificar*(x,r,m)**. Dados dos enteros r y m tales que 0 ≤ r ≤ m, así como la palabra binaria **x** a codificar mediante el código $\mathcal{RM}$(r,m) de longitud dim($\mathcal{RM}$(r,m)) = $\sum_{k=0}^{r}\binom{m}{k}$, devuelve su codificación usando la matriz generadora dada en forma estándar de $\mathcal{RM}$(r,m) generada por ***fostandar*(r,m)**.

In[3]:=
```
codificar[x_, G_] := Module[{}, Return[Mod[Dot[x, G], 2]]]
                     módulo      retorna  op···  producto escalar
```

El objetivo es diseñar el **algoritmo de Reed** en su versión más sencilla. **Fijados m un entero positivo arbitrario**, así como un orden para los elementos de la geometría finita *EG*(m,2) (en la práctica, se toma el orden estándar devuelto por ***fostandar*(r,m)**, se trata de realizar un programa que devuelva la **única decodificación** de una palabra binaria a través del código de Reed-Muller binario $\mathcal{RM}$(1,m). Para que el proceso funcione, esta debe tener a lo sumo 2^{m-2}-1 errores. Son necesarias las subrutinas siguientes:

- **Función *palabrasociada*(orden,*S*)**. Dados un orden para los elementos de la geometría finita *EG*(m,2) y *S* una variedad afín de esta, devuelve la palabra asociada a *S* para el orden dado.

In[4]:=
```
palabrasociada[orden_, S_] :=
 Module[{n = Length[S], l = Length[orden], i, j, palabra = {}, verificar = 0},
 módulo          longitud          longitud
  For[j = 1, j ≤ l, j++,
  para cada
   For[i = 1, i ≤ n, i++, If[orden[[j]] == S[[i]], palabra = AppendTo[palabra, 1];
                             si                                añade al final
     i = n + 1;
     verificar = 1]];
   If[verificar == 0, palabra = AppendTo[palabra, 0], verificar = 0]];
   si                               añade al final
  Return[palabra]]
  retorna
```

- **Función *subvariedad*(*S*,*T*)**. Dadas *S* y *T* variedades afines, determina si *S* es subvariedad para *T*.

In[5]:=
```
subvariedad[S_, T_] := Module[{n = Length[S], s = Length[T], i, j, verificar = 0},
                       módulo          longitud          longitud
  For[i = 1, i ≤ n, i++, For[j = 1, j ≤ s, j++, If[S[[i]] == T[[j]], j = s + 1;
                          para cada                si
     verificar = 1]];
   If[verificar == 0, Return[False], verificar = 0]];
   si                 retorna  falso
  Return[True]]
         verdadero
```

- **Función *pesopalabra*(x)**. Dada la palabra **x** binaria, devuelve su peso.

In[6]:=
```
pesopalabra[palabra_] :=
 Module[{cont = 0, i, longitud = Length[palabra]}, For[i = 0, i < longitud, i++;
 módulo                                 longitud              para cada
   If[palabra[[i]] ≠ 0, cont++]];
   si
  Return[cont]]
  retorna
```

- **Función *paridadbase*(x,orden,*S*)**. Dadas la palabra **x** binaria a decodificar, un orden para las palabras de la geometría finita *EG*(m,2) y *S* un **plano afín** de esta, devuelve su paridad respecto de **x** para el orden dado, calculada usando la **Proposición 15**.

In[7]:=
```
paridadbase[x_, S_, orden_] :=
 Module[{palabra = palabrasociada[orden, S], peso}, peso = pesopalabra[x * palabra];
 módulo
  If[Mod[peso, 2] == 0, Return["par"], Return["impar"]]]
     operación módulo   retorna          retorna
```

- **Función *casobase*(x,orden,variedadesmaximas)**. Dadas la palabra **x** binaria a decodificar, un orden para los elementos de la geometría finita *EG*(m,2) y un **conjunto formado por planos afines** de esta, devuelve un listado con las paridades de todos estos, obtenidas mediante la subrutina ***paridadbase*(x,orden,*S*)** anterior para cada plano afín *S*.

In[8]:=
```
casobase[x_, orden_, variedadesmaximas_] :=
 Module[{n, i, P = {}}, n = Length[variedadesmaximas];
 módulo                  longitud
  For[i = 1, i ≤ n, i++, P = AppendTo[P,
                             añade al final
     {variedadesmaximas[[i]], paridadbase[x, variedadesmaximas[[i]], orden]}]];
  Return[P]]
  retorna
```

- **Función *casogeneral*(x,orden,variedades,variedadesparidad)**. Dadas la palabra **x** binaria a decodificar, un orden para los elementos de la geometría finita *EG*(m,2) y dos conjuntos, uno con todas las variedades afines de *EG*(m,2) de una cierta dimensión, y otro con todas las variedades afines de *EG*(m,2) que tengan dimensión una unidad más, junto sus respectivas paridades respecto de **x** para el orden dado, devuelve las paridades del primero de estos conjuntos respecto de **x** para el orden dado, calculadas mediante el **CLM**.

In[9]:=
```
casogeneral[x_, orden_, variedades_, variedadesparidad_] :=
 Module[{n = Length[variedades], s = Length[variedadesparidad], i, j, T, P = {}},
 módulo          longitud                    longitud
  For[i = 1, i ≤ n, i++, T = {};
  para cada
   For[j = 1, j ≤ s, j++, If[subvariedad[variedades[[i]],
   para cada                 si
       variedadesparidad[[j, 1]]], T = AppendTo[T, variedadesparidad[[j, 2]]]]];
                                      añade al final
   If[Count[T, "par"] > Count[T, "impar"], P = AppendTo[P,
   si  conteo             conteo             añade al final
       {variedades[[i]], "par"}], P = AppendTo[P, {variedades[[i]], "impar"}]]];
                                     añade al final
  Return[P]]
  retorna
```

Ahora, **necesitamos determinar todos los planos y rectas afines**, así como **puntos**, de *EG*(m,2); definiendo funciones que determinan las correspondientes variedades afines. Una vez hecho esto, con la ayuda de estas, podemos generar el siguiente **conjunto que posee todos los planos, rectas y puntos** (estos últimos vistos como 0-variedades) **afines** de *EG*(m,2), en dicho orden.

In[10]:=
```
m = 5; r = 1; puntos := Tuples[{0, 1}, m]
                        tuplas
puntosvar[i_] := {Mod[puntos[[i]], 2]}
                 operación módulo
recta[i_, j_] :=
 Union[Mod[Table[puntos[[i]] + l (puntos[[j]] - puntos[[i]]), {l, 0, 1}], 2]]
 unión  op···  tabla
plano[i_, j_, h_] :=
 Union[Flatten[Mod[Table[puntos[[i]] + l (puntos[[j]] - puntos[[i]]) +
 unión  aplana  op···  tabla
       k (puntos[[h]] - puntos[[i]]), {k, 0, 1}, {l, 0, 1}], 2], 1]]
```

In[14]:=
```
variedades =
  {Union[Select[Flatten[Table[plano[i, j, h], {i, 2^m - 2},
   unión  selecc···  aplana  tabla
        {j, i + 1, 2^m - 1}, {h, i + 2, 2^m}], 2], Length[#] == 2^2 &]],
                                                    longitud
   Union[Select[Flatten[Table[recta[i, j], {i, 2^m - 1}, {j, i + 1, 2^m}], 1],
         selecc···  aplana  tabla
     Length[#] == 2 &]],
     longitud
   Union[Select[Table[puntosvar[i], {i, 2^m}], Length[#] == 1 &]]};
   unión  selecc···  tabla                    longitud
```

Hecho esto, estamos en condiciones de dar la **subrutina clave de este algoritmo**, que llamaremos ***decodificarReed*(x,orden,var)**. Esta, recibiendo la palabra **x** binaria a decodificar, un orden para los elementos de la geometría finita *EG*(m,2) y un **conjunto con las variedades necesarias** para realizar la decodificación (el que acabamos de construir), devuelve la decodificación de **x** a través del código de Reed-Muller binario $\mathcal{RM}$(1,m) empleando el **algoritmo de Reed**.

In[15]:=
```
decodificarReed[x_, orden_, var_] :=
 Module[{n = Length[var], S = casobase[x, orden, var[[1]]], s = 2, l = Length[x],
 módulo  longitud                                              longitud
   i, y = {}, j}, While[s ≠ n + 1, S = casogeneral[x, orden, var[[s]], S];
                  mientras
   s++];
  For[i = 1, i ≤ l, i++, j = Position[Union[orden], orden[[i]]][[1, 1]];
  para cada                 posición  unión
   If[S[[j, 2]] == "impar",
   si
    y = AppendTo[y, Mod[x[[i]] + 1, 2]], y = AppendTo[y, x[[i]]]]];
        añade al final  operación módulo  añade al final
  Return[y]]
  retorna
```

NOTA: Los programas presentados son válidos para cualquier código de Reed-Muller binario $\mathcal{RM}$(r,m) con 0 ≤ r ≤ m arbitrarios; salvo el correspondiente al **algoritmo de Reed** que, como se ha diseñado, solo es válido para r≐1. Este puede generalizarse a cualquier entero r en las condiciones usuales si se definen las correspondientes funciones para determinar las variedades afines de la geometría finita *EG*(m,2) con dimensión superior a 2 en orden ascendente. Basta incluir estas, por delante, en el conjunto que se define para guardar las variedades afines según su dimensión.

Para terminar, se incluyen aquí las líneas de código necesarias para ejecutar el **Ejemplo 10**.

In[16]:=
```
G = matgencaniter[1, 5];
G // MatrixForm
     forma de matriz

lista = formordstandar[1, 5];
orden = lista[[2]];
y = {1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
   0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1};
decodificarReed[y, orden, variedades]
```

Out[17]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Out[21]=
```
{1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1}
```