Explaining the code

Part 1:reliaSeq (code used to generate the reliability sequence)

Why? The reliability sequence is a sequence of numbers that essentially sorts the channels by how well they perform at transmitting the data. It is sorted from worst to best channels. The leftmost (worst) channels will be frozen and the rightmost (best) channels will carry the message sequence

The reliability sequence must be obtained in order to perform polar decoding and encoding.

Normally, the reliability sequence is predefined by 5G standards.

However, this was difficult to come by, and instead I found an approximate algorithm that claims it is 98% accurate (will be provided in the end).

1) For any $N = 2^p$, the 1-st and the p-th sub-vectors are given by

$$\mathbf{A}_0^{(N)} = [1] \text{ and } \mathbf{A}_p^{(N)} = [N]. \tag{6}$$

So, for $p = 1$, we have

$$\mathbf{A}^{(2)} = [1, 2], \tag{7}$$

where $\mathbf{A}_0^{(2)} = [1]$ and $\mathbf{A}_1^{(2)} = [2]$;

2) For $p > 1$, and for $i = 1, \ldots, p-1$, we have

$$\mathbf{A}_i^{(N)} = [\mathbf{A}_i^{(N/2)} \quad \mathbf{B}_i^{(N)}], \tag{8}$$

where $|\mathbf{B}_i^{(N)}| = |\mathbf{A}_{p-i}^{(N/2)}| = \binom{p-1}{p-i} := z$.
The sub-vector $\mathbf{B}_i^{(N)}$ is given by

$$\mathbf{B}_i^{(N)} = [b_{i,1}^{(N)} \ldots b_{i,j}^{(N)} \ldots b_{i,z}^{(N)}], \tag{9}$$

where

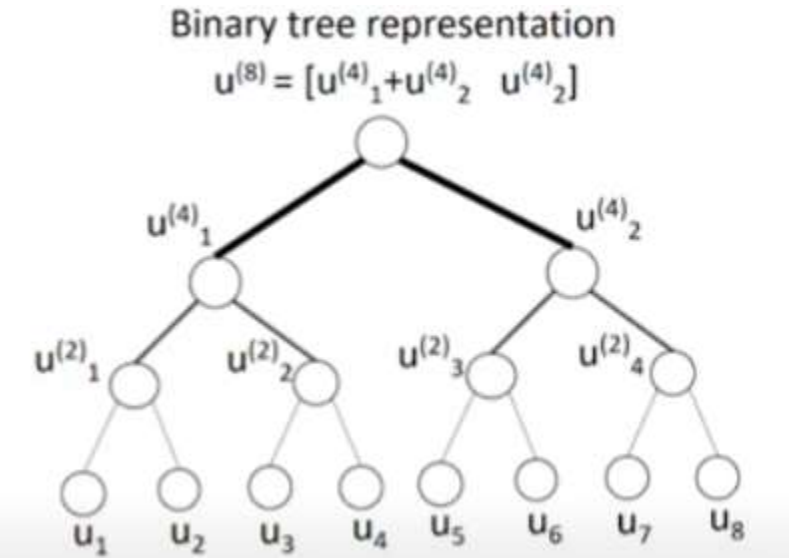
$$b_{i,j}^{(N)} = (N+1) - a_{p-i,z-j+1}^{(N/2)}. \tag{10}$$

Here is the matlab code implementation

```
function Avec=reliaSeq(NF)%this code generates
                          %the approximate reliability sequence
    pF=log2(NF);
    A={1;2};   %initial sequence for N=2
for k=2:pF  %iterations for each N until reaching desired
             %number of channels for specified NF
    N=2^k;
p=log2(N);
Aprev=A;
A={1};
for i=1:p-1
    z=nchoosek(p-1,p-i);
    B=zeros(z,1);
    AI=cell2mat(Aprev(i+1));
    AI2=cell2mat(Aprev(p-i+1));
    for j=1:z
        B(j)=(N+1-AI2(z-j+1));
    end
    res=[AI B'];
    A=[A {res}];
end
A=[A {N}];
end
Avec=[]; %now to convert the cell to a vector
L=length(A);
for i=1:L
Avec=[Avec cell2mat(A(i))];
end
end
```

Part 2: The encoder

First, after determining the frozen channels and message carrying channels, we put the message in the rightmost channels. Note that for a message length K, K must not exceed the codeword length N. Note that N must be a power of 2.



The algorithm is very simple. If we think of the codeword (u) as leaves in a binary tree, all that is needed to do is successive XOR and concatenation operations. For example, to get the bits at a certain nonleaf node, we XOR both of its children and concatenate the right child. This is done until the entirety of the codeword goes through this process. The maximum depth is determined by log2(N). Note that the number of bits XOR'd this way doubles every iteration.

This is the matlab code

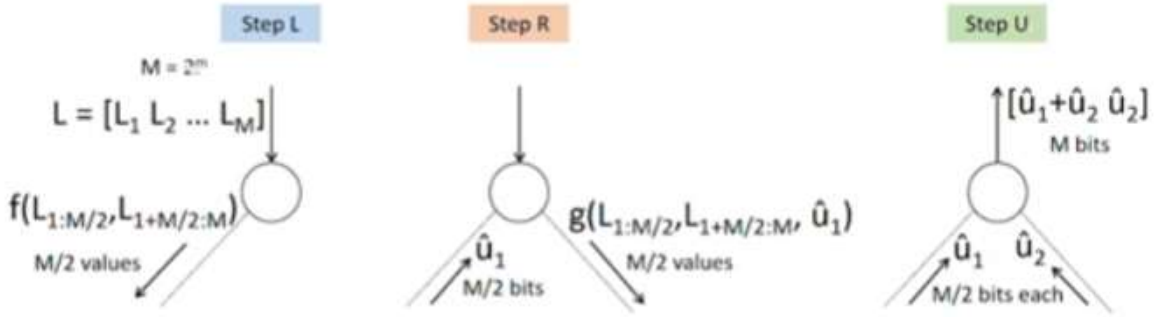
```
function u=encoder(msg,N,k)
n=log2(N);  %maximum depth+1
RS=reliaSeq(N); %generate reliability sequence
u=zeros(1,N); %initializing
u(RS(N-k+1:end))=msg; %putting the message on leftmost channels
m=1;%number of bits to be added per iteration

for d=n-1:-1:0 %maximum depth: at the leaf nodes to the
                %0th depth at the root node
   for i=1:2*m:N
        a=u(i:i+m-1); %left child
        b=u(i+m:i+2*m-1);%right child
        u(i:i+2*m-1)=[xor(a,b) b];%xor and concatenate
   end
   m=m*2;%as depth decreases number of bits to be operated on
        %increases also
end
end
```

Part 3: The decoder by Successive Cancellation

This one was tricky.

The decoder will be carried out in 3 main steps



Step L "left": the interior node is given an array of M incoming "beliefs". When direction to the left child the "min sum" function (f),specified as follows: $f(r_1,r_2) = \text{sgn}(r_1)\,\text{sgn}(r_2)\,\min(|r_1|, |r_2|)$

is applied on the two halves of the belief vector. This yields M/2 decisions on u1 to be sent back to the interior node and to the right child.

Step R "right": The interior node sends the same M beliefs to the right child in order to compute the "g" function specified as follows

$$g(r_1, r_2, b) = r_2 + (1 - 2b)\, r_1$$

Where r1 and r2 are as before, the two halves, and b is the left child decisions computed in the previous step. This yields another M/2 vector of decisions of the right node.

Step U"up": When the interior node gets both of the decision vectors, it carries out the XOR then concatenate function on them, the same one used in encoding. This will yield an M length vector. Then it sends it up back to the parent. This M length vector will act as the decision estimate for the parent interior node (either u1 or u2 depending on if its left or right child), and this continues iteratively until the root depth is reached and the entire codeword is decoded, and a message is obtained

In summary, the decoder works like this;

- Start at root node.
- If leaf node, make decision and go to parent.
- If not leaf node, do step L and go to left child.
- Wait until decision is received from left child. Start step R when done
- Wait until decision is received from right child. Start step U when done
- Go to parent
- Back to beginning until done

MATLAB implementation will be as follows

```
function msgcap=decoder(u,N,k)
n=log2(N);
RS=reliaSeq(N); %same initializations as in encoder
F=RS(1:N-k); %here defining the indexes of frozen bits will be useful
ucap=zeros(n+1,N); %estimated decoded message will be in the leftmost bits

%decoder
L=zeros(n+1,N); %initializing the belief vector for each depth
ns=zeros(1,2*N-1); %node state vector. 0 means unactivated, 1 means
                    %L is finished, 2 means R, 3 means U.
node=0;
depth=0;%starting at root
L(1,:)=u;%received signal act as beliefs (belief of root)
decoded=0;

f=@(a,b) (1-2*sign(a)).*(1-2*sign(b)).*min(abs(a),abs(b));
g=@(a,b,c) b+(1-2*c).*a;
%predefining the functions for ease of use
while (decoded==0) %keep going until decoded
    if depth==n %if leaf node start making decisions
        if (any(F==node+1))
           ucap(n+1,node+1)=0;
        else
            if (L(n+1,node+1)>=0)
                ucap(n+1,node+1)=0;
            else
                ucap(n+1,node+1)=1;
            end
        end
        if (node==N-1) %if last node then decoding is finished
            decoded=1;
        else
            node=floor(node/2);%if not keep going and go up in depth
            depth=depth-1;
        end
    else %if not leaf node
        npos=(2^depth-1)+node+1; %position of node in node state vector
        if ns(npos)==0 %not leaf; state zero, must do "f" function then send to left child
            val=2^(n-depth);%length of incoming belief vector
            Ln=L(depth+1,val*node+1:val*(node+1)); %beliefs from parent
            a=Ln(1:val/2); %left half
            b=Ln(val/2+1:end);%right half
            node=node*2; %next node which will be the
            depth=depth+1;%left child
            val=val/2;%the next one will be halved
            L(depth+1,val*node+1:val*(node+1))=f(a,b);%carry out minsum and store
            ns(npos)=1;%change state of current node to 1 (L done)
        else
```

```
            if ns(npos)==1 %previously state 1 node means now it is entering state 2
            val=2^(n-depth); %length of beliefs
            Ln=L(depth+1,val*node+1:val*(node+1)); %incoming beliefs
            a=Ln(1:val/2);
            b=Ln(val/2+1:end); %splitting to halves
            lnode=node*2; %left child
            ldepth=depth+1;%left child node
            lval=val/2; %length of beliefs of next node
            ucapn=ucap(ldepth+1,lval*lnode+1:lval*(lnode+1)); %incoming decisions
                                                              %from the left child
            node=node*2+1; %next node
            depth=depth+1;%right child
            val=val/2;
            L(depth+1,val*node+1:val*(node+1))=g(a,b,ucapn); %g and storage
            ns(npos)=2;
            else %finished step R, now entering step U and go to parent
             val=2^(n-depth);
             lnode=node*2; %left child
              rnode=node*2+1; %right child
          cdepth=depth+1;
          cval=val/2;
          ucapl=ucap(cdepth+1,cval*lnode+1:cval*(lnode+1)); %incoming decisions from left
          ucapr=ucap(cdepth+1,cval*rnode+1:cval*(rnode+1));%incomign decisions from right
          ucap(depth+1,val*node+1:val*(node+1))=[xor(ucapl,ucapr) ucapr]; %xor and concatenation
          node=floor(node/2); %go to parent
              depth=depth-1;%parent's depth
          end
      end
      end
  end

  msgcap=ucap(n+1,RS(N-k+1:end)); %assign the message from rightmost bits
  end
```