1 协议分析

本章节主要描述 NR 协议中 Polar 码的编码过程，包括上行 Polar 编码和下行 Polar 编码过程。Polar 码由于在短码时的优异性能，被采纳为 5G 控制信息的编码方案。具体应用于下行控制信息，上行控制信息以及 PBCH 所承载的广播信息。

表 1-1 NR 编码类型汇总

上下行	承载内容	物理信道	编码类型
下行	DCI 控制信息	PDCCH	Polar 编码
	MIB 信息	PBCH	Polar 编码
	SIB 信息	PDSCH	LDPC 编码
	Paging 信息	PDSCH	LDPC 编码
	下行数据	PDSCH	LDPC 编码
上行	UCI 信息	PUCCH/PUSCH	当 K>=12 时，Polar 编码； 当 3<=K<=11 时，RM 编码; 当 K<=2 时，重复编码；
	上行数据	PUSCH	LDPC 编码

下图给出了上下行 Polar 码的编码过程，并给出协议中对应的章节号。接下来对每个处理模块进行协议分析描述。

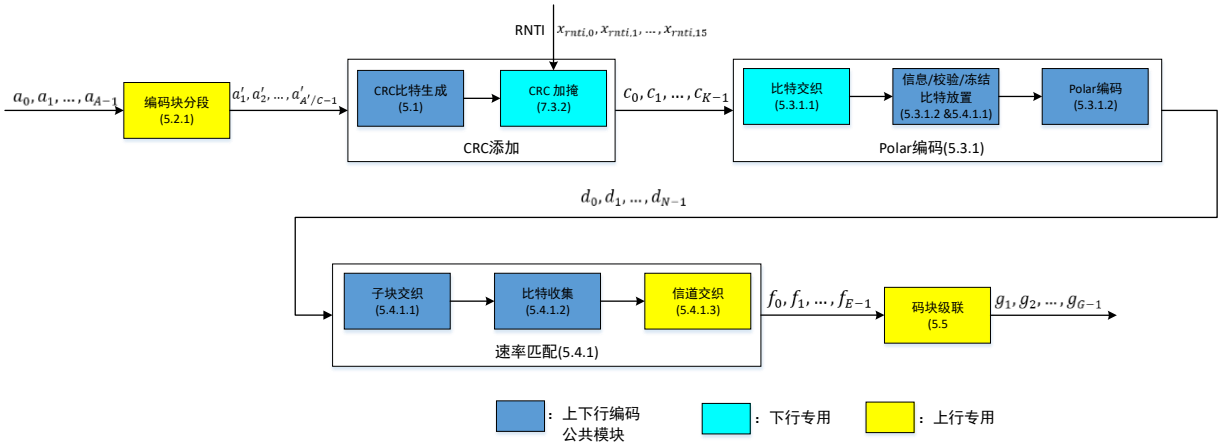


图 1-1 polar 码编码流程图

1.1 码块分段

由于 Polar 码的编译码复杂度随着母码码长 N 的增大明显增大，标准规定上行控制信道 Polar 码母码的最大码长为 N=1024 比特，超过 1024 比特的部分对母码进行重复，达到特定的码长。在一些特定情形下，上行控制信息 UCI 的比特长度可达到 500 比特以上(如承载 CSI 反馈信息)，而码率最低可达 0.1，因此速率匹配后的输出可能远大于 1024 比特，此情形下需要对循环缓冲器中码字的大部分进行重复，而重复只有重复带来的能量增益，没有编码增益，与信道编码相比重复编码会严重损失 BLER 性能。如果把信息比特分成两段，会降低信息比特长度，在母码长度仍为 1024 比特的情形下，相当于降低了编码码率，这样明显能提供编码增益，能带来 BLER 性能的提升。

但对于一些速率匹配输出长度较短的情况，分段会增加译码复杂度，但其性能改善可能很小或造成性能损失，此时没必要对 UCI 进行分段，因此需要一种分段方案，根据信息长度和码率来判断是否需要对信息比特进行分段。

通过对分段和不分段性能的仿真，最终确定了分段规则为：当信息比特长度（包括 CRC 比特）$K \geq 360$且速率匹配后比特长度$E \geq 1088$，UCI 被分成两段。考虑到 K 较大时，对应的 CRC 为 11 比特，如果$K \geq 1013(1024-11)$，不管E是否大于 1088 比特，都需要对 UCI 进行分段。所以最终的分段准则为：$K \geq 360$且$E \geq 1088$或$K \geq 1013$。为了保证两个码块采

用相同的编码参数，特别是相同的母码长度，需要信息比特和编码比特长度都必须等分。对于分段情况，两个码块分别添加 CB-CRC，不对整个码块添加 TB-CRC，两个码块编码后分别独立交织，然后串行级联。

下行最大的 DCI 长度为 140 比特，添加 24 比特最大长度为 164 比特，因此没必要对下行 DCI 进行分段，下行 DCI 只有一个编码块。码块分段仅适用于上行信息比特数较大的场景。

	K	码块数
下行	任意长度	1
上行	$K \geq 360$且$E \geq 1088$ 或$K \geq 1013$	2
	其他情况	1

1.2 CRC 添加

CRC 的作用是为了校验传输的信息比特的正确性。下行为了避免虚警问题，采用 L=24 比特的 CRC 校验码，上行由于不存在虚警问题，采用了 L=6(12<=A<=19)和 L=11(A>=20)两种 CRC 长度，其中 A 为输入的信息比特长度。

上行与常规 CRC 添加的方式相同，下行由于需要盲检信息比特的长度，而 Polar 码的冻结比特均为 0，为了区分信息比特为 0 和冻结比特的场景，需要将 CRC 寄存器的初始值设置为全 1，而不是全 0。

令$A(D)$为输入信息比特的码字多项式，$B(D)$为全 1 码字，CRC 寄存器的初始值，$g(D)$为 CRC 校验码的生成多项式

$$A(D) = a_0 D^{A+L-1} + a_1 D^{A+L-2} + \cdots + a_{A-1} D^{L}$$

$$B(D) = D^{L-1} + D^{L-2} + \cdots + 1$$

$$C(D) = B(D) \cdot D^{A} + A(D)$$

$$\boldsymbol{r(D) = mod\left(C(D) \cdot D^{L}, g(D)\right) = mod\left(B(D) \cdot D^{A+L} + A(D) \cdot D^{L}, g(D)\right) =}$$

$$\boldsymbol{= mod\left(B(D) \cdot D^{A+L}, g(D)\right) + mod\left(A(D) \cdot D^{L}, g(D)\right)}$$

令$r_2(D) = mod\left(B(D) \cdot D^{A+L}, g(D)\right)$，$r_1(D) = mod\left(A(D) \cdot D^{L}, g(D)\right)$

$$\boldsymbol{r(D) = r_1(D) + r_2(D)}$$

$$\boldsymbol{r_1(D) = r(D) + r_2(D)}$$

其中$r_1(D)$为 CRC 初始寄存器为全 0 时$A(D)$对应的 CRC 校验码字，$r_2(D)$为理想已知的码字，只跟信息比特长度 A 相关，$r(D)$为 CRC 寄存器初始值为全 1 时$A(D)$对应的 CRC 校验码字，也为实际发送的 CRC 校验码。

从上式可以看出，由于$r_2(D)$与信息比特长度相关，所以有效避免了盲检信息比特长度错误的现象。

下行 UE 为了盲检 DCI，还需要将 CRC 比特与 RNTI 值进行加扰，根据对应的 RNTI $x_{rnti,0}, x_{rnti,1}, \ldots, x_{rnti,15}$，对码块的最后 16 比特进行加扰。

1.3 Polar 编码

协议中的 Polar 编码模块包括比特交织，比特类型置位和 Polar 编码三个子模块。

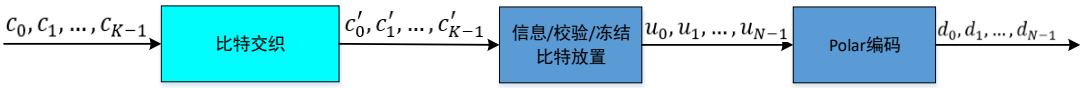


图 1-2 极化码编码流程图

1.3.1 比特交织

本章节对应的比特交织是指对 CRC 编码完后的数据比特进行交织，即分布式 CRC。该操作只针对下行控制信息和广播信息。通常 CRC 校验必须在 Polar 码译码结束并得到原始的信息比特和 CRC 比特的译码结果之后进行。对于下行 DCI 存在盲检的需求，盲检测的延

迟为一次完整的 Polar 译码时间乘以盲检次数。在盲检过程中，当盲检测的候选输入包括纯噪声时，若能提前识别失败的译码，并提前终止译码进程，就能够有效地降低盲检的延迟，并节省终端的功耗。而对于上行 UCI 不存在盲检的需求，因此对分布式 CRC 没有需求，分布式 CRC 只用于下行，用于提前终止译码，降低盲检的延迟并节省终端的功耗。

CRC 编码器是线性编码且生成的是系统码，所以 CRC 编码器必然对应一个生成矩阵 G_{CRC}，信息比特乘以生成矩阵即得到 CRC 比特，生成矩阵由 CRC 多项式确定且与信息比特长度相关。所以与 CRC 比特存在校验关系的信息比特也全部被置换到对应的 CRC 比特，这样译码部分信息比特就可以进行 CRC 了，如果校验失败即停止译码，实现 ET(Early Termination)功能。所以分布式 CRC 在 CRC 编码之后级联一个交织器，交织器只是改变了 CRC 的校验顺序，并没有改变原校验关系，可以证明，相同的 CRC 多项式，不同信息比特长度下的 CRC 生成矩阵具有嵌套关系，所以根据最大信息比特长度设计交织模式可以得到任意信息比特长度对应的交织模式。交织前后对应的生成矩阵如下式所示：

$$\boldsymbol{G}=\begin{pmatrix}\mathbf{1} & \cdots & \mathbf{0} & \boldsymbol{g}_{\mathbf{0},\mathbf{0}} & \cdots & \boldsymbol{g}_{\mathbf{0},\boldsymbol{k}-\mathbf{1}}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{1} & \boldsymbol{g}_{\boldsymbol{n}-\mathbf{1},\mathbf{0}} & \cdots & \boldsymbol{g}_{\boldsymbol{n}-\mathbf{1},\boldsymbol{k}-\mathbf{1}}\end{pmatrix}$$

$$\widehat{\boldsymbol{G}}=\begin{pmatrix}\boldsymbol{g}'_{\mathbf{0},\mathbf{0}} & \boldsymbol{g}'_{\mathbf{0},\mathbf{1}} & & \boldsymbol{g}_{\mathbf{0},\boldsymbol{k}-\mathbf{1}}\\ \boldsymbol{g}'_{\mathbf{1},\mathbf{0}} & \boldsymbol{g}'_{\mathbf{1},\mathbf{1}} & & \boldsymbol{g}_{\mathbf{1},\boldsymbol{k}-\mathbf{1}}\\ \vdots & \vdots & \cdots & \vdots\\ \boldsymbol{g}_{\boldsymbol{d}'(\mathbf{0}),\mathbf{0}} & \vdots & & \vdots\\ \mathbf{0} & \boldsymbol{g}_{\boldsymbol{d}'(\mathbf{1}),\mathbf{1}} & & \vdots\\ \mathbf{0} & \mathbf{0} & & \vdots\\ & \vdots & \ddots & \vdots\\ \mathbf{0} & \vdots\ \mathbf{0} & \cdots & \boldsymbol{g}_{\boldsymbol{d}'(\boldsymbol{k}-\mathbf{1}),\boldsymbol{k}-\mathbf{1}}\end{pmatrix}$$

区别于后置式 CRC，分布式 CRC 的 Polar 码对 CRC 编码后的码字先进行重新排列（交织），然后进行 Polar 编码。分布式 CRC 的交织特性满足以下特征：

- 不改变校验比特和信息比特之间的校验关系
- 每个校验比特均位于其检验的所有信息比特之后
- 若干校验比特位于信息比特之间

图 1-3 为不同信息长度下 CRC 校验比特与其对应校验的信息比特之间关系，其中蓝色部分表示 CRC 校验比特，黄色部分表示信息比特（即数据），由图可以看出 CRC 校验比特仅参与部分信息比特的校验，因此可以通过交织分布在信息比特中间。

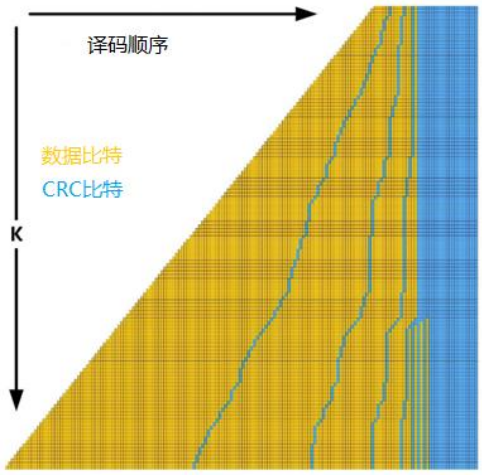


图 1-3 极化码 CRC 校验比特位置示意图

1.3.2 信息/校验/冻结比特置位

- 子信道可靠度排序

Polar 码为了支持 5G NR 灵活的码长和码率的需求，需要设计足够实用的子信道可靠排序序列。

在标准讨论之初，由于 DE 的方法计算得到各个子信道的可靠度信息比较准确，不同码率和码长的 Polar 码构造序列主要基于 DE 方法及其变种。为了获得特定码率和码长的 Polar 码排序序列，需要根据信道相关参数（如信噪比，信道 delay spread 类型），实时计算或读取用这个参数事先计算并预存的序列，复杂度过高且不实用。而 PW 方法（详见 2.2.1.2）的提出从标准化和实用化的角度澄清了 Polar 码排序序列和信道参数无关的特点，展示了可以用一个序列来提供任意码长和码率的 Polar 构造方法，并且在各种码率和码长配置下展现出良好且稳定的性能。最终，5G NR Polar 码采用单序列达成共识，不同长度的母码序列从

单码序列中抽取，以 PW 方法作为基础，然后用计算机搜索的方法对部分序列进行简单重排得到最终的序列。

3GPP 38.212 协议中基于 PW 方法给出了最长码长（1024）下极化子信道对应的可靠度排序表，该表分两列，分别是$W(Q_i^{N_{max}})$和$Q_i^{N_{max}}$，其中$Q_i^{N_{max}}$表示码长为N_{max}的 Polar 码子信道中可靠性排序为$W(Q_i^{N_{max}})$的子信道序号，$W(Q_i^{N_{max}})$表示第$Q_i^{N_{max}}$个子信道的可靠度在所有子信道中的排序。在表格中，序列以可靠度升序的方式顺序记录，即$W(Q_0^{N_{max}}) < W(Q_1^{N_{max}}) < W(Q_{N_{max}-1}^{N_{max}})$。对于长度为 N 的 Polar 码，从第二列$Q_i^{N_{max}}$中顺序抽取值小于 N 的元素构成子序列$Q_0^{N-1} = \{Q_0^N, Q_1^N, \dots, Q_{N-1}^N\}$即是 N 个子信道的可靠度排序，即$W(Q_0^N) < W(Q_1^N) < W(Q_{N-1}^N)$.

- 校验比特

引入奇偶校验比特校验的 Polar 码称为 PC-Polar (Parity check-Polar) 码, 研究发现，引入少量校验比特能增加 Polar 码的最小码字距离，提高 L 条幸存路径的译码成功率，当信息比特很短时，PC-CA Polar 码的误码性能优于 CA-polar 码，因此 PC-CA Polar 码被采纳为 NR 上行控制信息的编码方案。

与 CA-POLAR 码相比，PC-CA-Polar r（PC-Crc Aided-Polar）码在 CRC 编码器之后级联了 PC 编码器，在译码端 PC 与 CRC 都参与路径筛选，而只有 CRC 比特被用于差错检验。n_{PC}个 PC 比特的位置索引分为两个集合，其中$n_{PC} - n_{PC}^{wm}$个 PC 比特放置在非冻结比特位置索引中最不可靠的$n_{PC} - n_{PC}^{wm}$位置上，另外n_{PC}^{wm}个比特放置在除了$n_{PC} - n_{PC}^{wm}$个最不可靠位置索引外的其它非冻结比特位置对应的 Polar 码生成矩阵中行重最小的位置上, 如果行重最小的行对应的比特位置索引个数多于n_{PC}^{wm}个, 此时选择最可靠的n_{PC}^{wm}个位置索引放置这些 PC 比特。

校验方程的设计目标是提高 Polar 码的码距（即矩阵中任意两行之间“1”的差异），这可以从 Polar 码译码过程的错误传播模式入手，由于克罗内克积（⊗）具有以 2 的整数次幂为周期的递归结构，最常见的误差传播样式间隔为 1、2、4、8 个比特的位置，因此在构建 PC 校验方程时，需要使得方程内的元素尽量避免以 2 的幂次为间隔。一个简单有效的 PC 校验方程准则为：从 PC 比特位置开始，以 5 为间隔，向前选取信息比特位置作为 PC 校验方程的一部分。此时的编码过程可以用一个长度 5 的移位寄存器来实现。

奇偶校验比特仅适用于上行码长较短的场景，当$18 \le K_r \le 25$时，$n_{PC} = 3$。$n_{PC} = 3$的 PC 比特序列为通过一个长度为 5，初始值为 0 的循环移位寄存器计算得到。每个 PC 位由分配给前面子信道的消息位（模 5 得到）的异或结果决定，其中不包括先前计算的奇偶校验位。如图 1-4 所示，每个绿色节点为校验比特，连线表示其校验的信息比特或冻结比特。

$$\boldsymbol{u_i = \oplus u_{5j+p}}$$

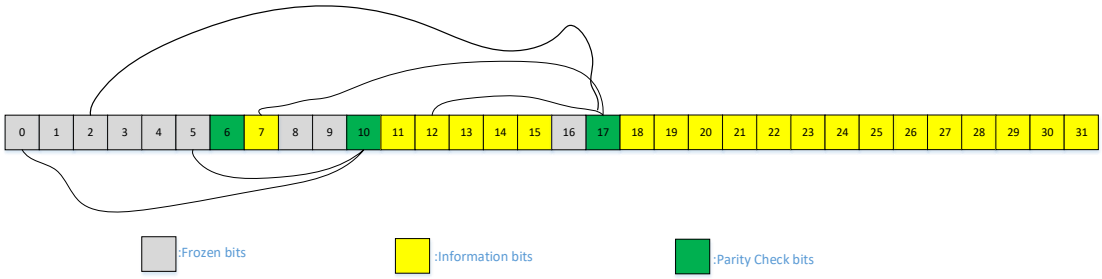


图 1-4 极化码 PC 校验示意图

1.3.3 Polar 编码

将信息比特放置在信息位，其它位放置事先确定好的冻结比特，组成信源序列u_1^N，经过生成矩阵生成码字$x_1^N = u_1^N G_N$。经过化简，$G_N = B_N F^{\otimes n}$，B_N为比特反转矩阵，矩阵$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$。编码过程可以用生成矩阵$G_N$表示，编码的复杂度为$O(N\,log\,N)$。图 1-5 为 Arikan 教授提出 Polar 码时的编码方案。

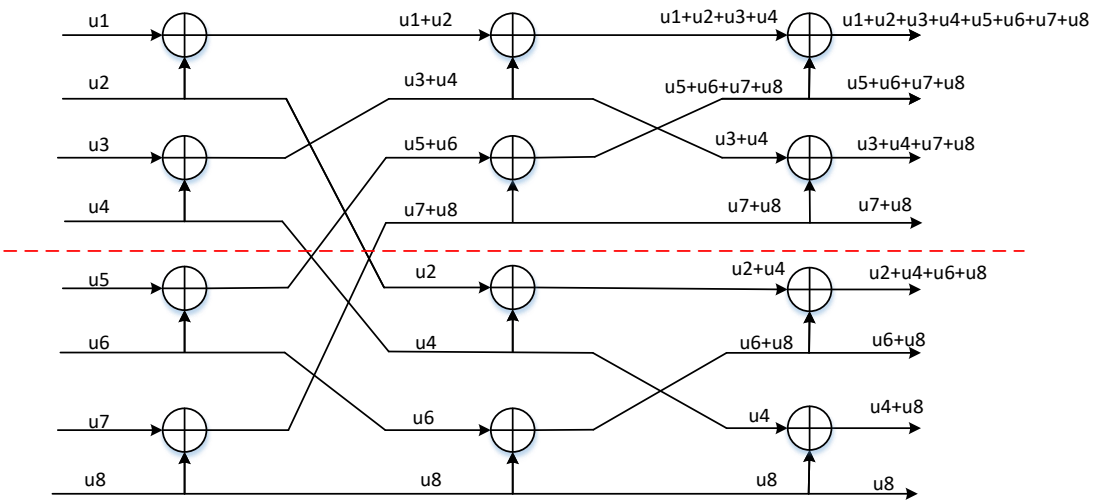


图 1-5 极化码编码比特流（以码长 8 比特为例）

$$d = uG_N = vF_N = (u \times B_N) \times F_N$$

$$G_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$B_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad F_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

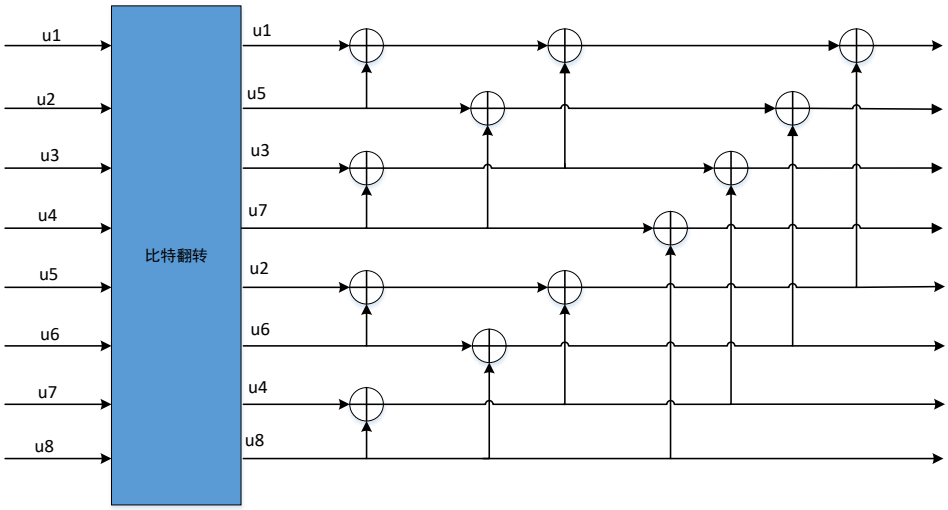


图 1-6 论文中极化码编码示意图

NR 标准定义的编码方案与该方案相比，去掉了B_N的操作，由于比特翻转只会影响构造

信道的顺序，而对实际性能没有影响，同时为了保证生成矩阵的下三角特性，便于速率匹配简单有效地实现，NR 没有对比特翻转部分进行标准化，等价于从右往左的 Arikan 编码器。NR 中使用的 Polar 码编码器如下图所示：

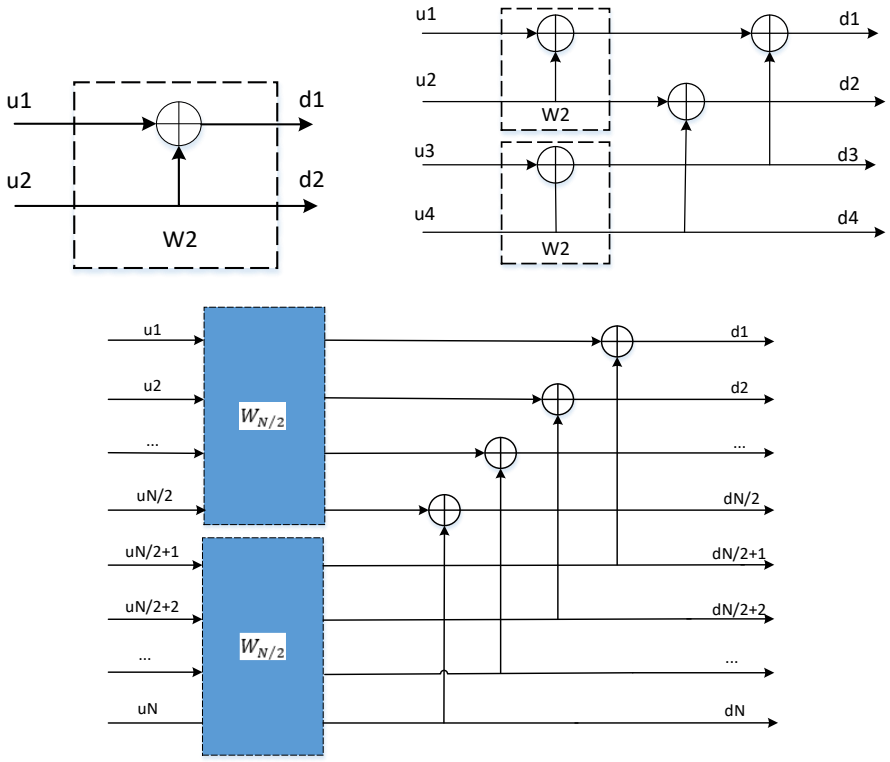


图 1-7 极化码分极示意图

$$d = u \cdot G_N$$

式中，u为$1 \times N$输入比特，G_N为$N \times N$生成矩阵，d为$1 \times N$编码后的比特。$G_N = F^{\otimes n}$，$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, F^{\otimes n} = \begin{bmatrix} F^{\otimes n-1} & 0 \\ F^{\otimes n-1} & F^{\otimes n-1} \end{bmatrix}$

1.4 速率匹配

3GPP 提出的 5G 标准中，速率匹配操作可以分为三个子操作：子块交织、循环缓冲和信道交织。

图 1-8 极化码速率匹配流程图

1.4.1 子块交织

子块交织的目的是对编码后的比特序列进行重排，确保 Polar 码纠错能力最强的（可靠度高）比特在接下来的循环缓冲操作中被保留下来。具体地，将信道比特分成长度为 32 的子块，形成 4 组，中间两组交替，同时也在打孔时对前部分信息比特进行预冻结。

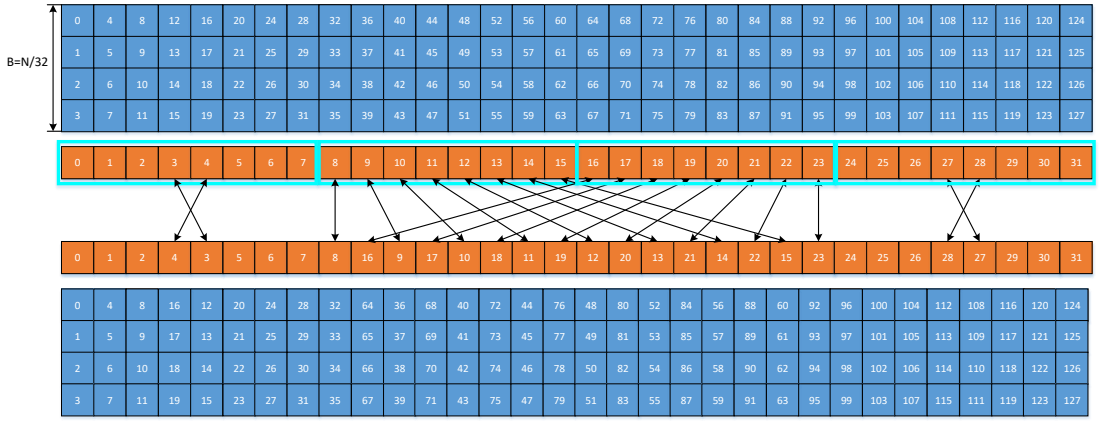
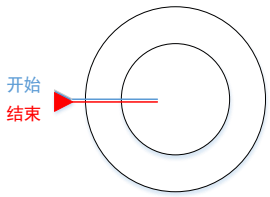


图 1-9 极化码子块交织示意图

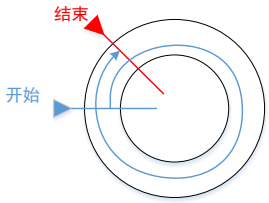
1.4.2 比特收集

该步骤的目的是将之前子块交织操作后得到输出序列长度从 N 变成 E，从而达到相应码率的要求。根据信息比特 K 和母码码长 N、速率匹配后的 E 可得到三种情况，分别是重复、打孔和缩短，由于打孔和缩短都是不传输比特，此时待传输的各个子信道的可靠性发生了变化。速率匹配的设计主要有两种方案：方案一是根据删除比特的位置，重新计算各个子信道的可靠度排序，由于需要重新评估子信道的可靠度，虽然保证了较优的性能，但是带来了极大的开销。方案二是不进行重新评估，虽然性能可能下降，但是极大地降低了实现复杂度。最终标准采用了方案二。



编码后数据

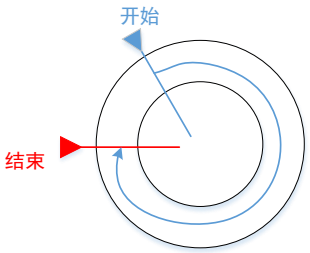
- 若$E \geq N$,速率匹配模式为重复，则从头（序号 0）顺序循环读取E个比特。



重复

- 若$E < N$且$K/E \leq 7/16$，速率匹配模式为打孔模式

采用打孔的方式，打掉$0\sim(E-N)-1$，传输的比特为$(E-N)\sim(N-1)$,译码时相应位置$0\sim(E-N)-1$对应接收比特的 LLR 系数置 0

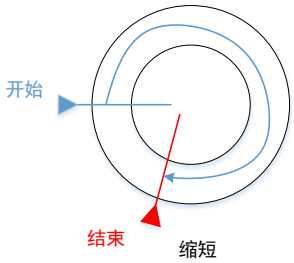


打孔

- 若$E < N$且$K/E > 7/16$，速率匹配模式为缩短模式

在缩短模式下，缩短后面的比特，即$E\sim(N-1)$比特不进行传输，传输的比特为$0\sim(E-1)$，译码时相应接收比特位置$E\sim(N-1)$的 LLR 系数置$+\infty$。

虽然缩短模式和打孔模式都是打掉部分比特，但是打孔模式下由于打掉的比特是未知的，因此对应位置的 LLR 系数需要置为 0，而在缩短模式下，考虑到 Polar 码生成矩阵的下三角特性，编码后的$E\sim(N-1)$比特只跟编码前的$E\sim(N-1)$个比特相关，而编码前的$E\sim(N-1)$个比特被设置为冻结比特，即传输 0 序列，则编码后的$E\sim(N-1)$个比特也为全 0 比特流，接收端已知，解速率匹配时其 LLR 值按最大值填充。



1.4.3 信道交织

信道交织的目的是为了让高阶调制的 Polar 编码具有更优的性能。Polar 码在评估子信道可靠度时，假设编码后的比特经历了可靠度相同的信道，而实际上当 Polar 码用于高阶调制时，该假设不再成立，高阶调制符号每个比特对应的可靠性是不相同的。一种方法是根据编码比特经历的信道评估子信道的可靠度，但是复杂度更高。另一种方法是引入比特交织，将编码和调制解调解耦，通过交织让高阶调制符号每个比特对应的解调性能更均衡。由于下行信道采用 Polar 编码只有 QPSK 调制，而 QPSK 各个比特的解调性能是一致的，因此下行控制信息或广播信息的 Polar 编码没必要引入信道交织的过程，信道交织只针对上行信道采用 Polar 编码。

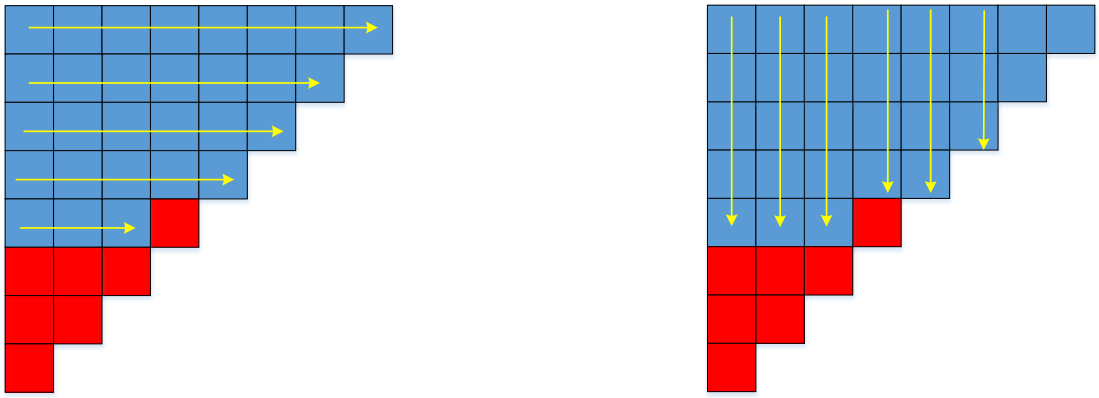


图 1-10 极化码信道交织示意图

信道交织器由长度为T比特的等腰三角形构成，其中 T 为满足$\frac{T(T+1)}{2} \geq E$的最小正整数。假设输入比特序列$e = [e_0, e_1, \ldots, e_{E-1}]$，得到相应的输出序列$f = [f_0, f_1, \ldots, f_{E-1}]$。交织操作通过将$e$序列逐行写入三角形，并将剩余的$\frac{T(T+1)}{2} - E$个比特填入 NULL 比特，然后通过逐列

读取三角形交织器每一列中的比特并跳过 NULL 比特，得到输出序列f。

1.5 码块级联

码块级联合并是与码块分割对应的逆操作，仅当上行 UCI 编码前经过了码块分割时编码后才需要激活码块级联。两段长度均为 E 的编码码块被合并形成一个长度为 G 的传输码块。若$G = 2E + 1$，则会在第二个码块的末尾添加一位 0 比特。

2 算法原理

本章节主要描述 Polar 码的基本原理及译码算法过程。

2.1 Polar 码概述

Polar 码是 Arikan 教授在 2008 年发明的一种基于信道极化的编码方法，是首个理论上可严格证明能在二进制输入离散无记忆信道上，渐进性能可逼近香农极限的信道编码方案。Polar 码具有严格的结构，生成矩阵不需要额外设计，并且具有较低的编码和译码复杂度，速率匹配简单，可得到几乎任何码率的码字，并被理论证明没有误码平层。Polar 码在 5G 通信系统中具有重要意义，被选定为 eMBB 场景下控制信道和广播信道的信道编码方案。

Polar 码是基于信道极化理论构造出来的，将一组二进制输入离散无记忆信道（B-DMC），通过信道合并和分裂的操作，得到一组新的二进制输入离散无记忆信道，该过程称为极化过程，得到的新的信道成为子信道。如图 2-1 所示，经过信道合并分裂后，原始的信道 W 会极化为一个"差"（信道容量小）的信道 W-和一个"好"的信道（信道容量大）的信道 W+。

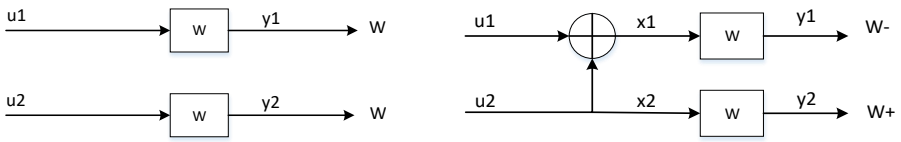


图 2-1 极化码信道极化过程示意图

图中W为原始信道，转移概率为$p(y|u)$，$W-$和$W+$为经过1级极化得到的子信道，其中$W-$子信道转移概率为$p(y_1|u_1 \oplus u_2)$，相对于 W 的性能变差，$W+$子信道假设的转移概率为$p(y_2|u_2,u_1)$，相对于 W 性能变好。

$$\boldsymbol{y_1 = x_1 = u_1 \oplus u_2}$$

$$\boldsymbol{y_2 = x_2 = u_2}$$

则对应可得到

$$u_1 = y_1 \oplus y_2$$

对于二进制删除信道，只有y_1和y_2都正确接收的时候，u_1才正确，假设y_1或y_2错误接收的概率为p，因此正确接收的概率为$1-p$，则u_1正确的概率为$(1-p)^2$。

假设u_1为校验比特，传输的信息预先已知，

$$u_2 = y_2$$
$$u_2 = y_1 \oplus u_1$$

只有y_1和y_2都错误接收的时候，u_2才错误接收，则u_2正确的概率为$1-p^2$，相比于极化之前的信道

$$y_1 = u_1$$
$$y_2 = u_2$$

u_1的错误概率从p变化至$1-(1-p)^2 = 2p-p^2$，u_2的错误概率从p变化至$1-(1-p^2) = p^2$。因此信道极化在不损失信道总容量的前提下，构造出的信道W^+相比于原来信道W好，而信道W^-相比于原来信道W差

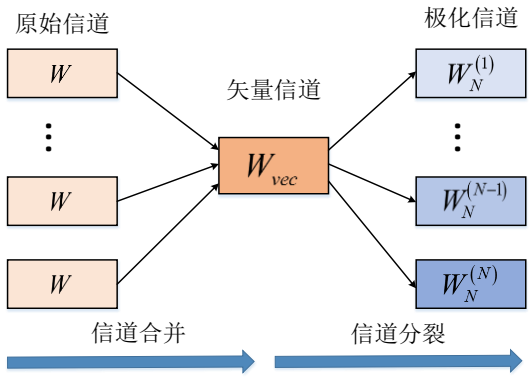
$$I(W^-) \le I(W) \le I(W^+)$$
$$I(W^-) + I(W^+) = 2I(W)$$
$$I(W^-) = 1-(2p-p^2)$$
$$I(W) = 1-p$$
$$I(W^+) = 1-p^2$$

当参与极化的信道足够多时，一部分子信道的容量趋向于1（可靠子信道），另一部分子信道的容量趋向于0（不可靠子信道）。码长为N的极化码，当N增大时有如下渐进特性：

$$\text{N个容量为I(W)的信道} \xrightarrow{\text{信道极化}} \begin{matrix} NI(W)\text{个容量为1的信道} \\ N(1-I(W))\text{个容量为0的信道} \end{matrix}$$

利用这一现象，可以将信息比特承载在可靠子信道上，在不可靠子信道上放置收发两端已知的固定比特（冻结比特），通过这种方式进行构造的编码就是极化码。

Polar 码的编码过程可以认为是，将N个独立且相同的 B-DMC 原始信道W，经过信道合并和分裂操作形成N个新的比特子信道$\left\{W_N^{(i)}: 1 \le i \le N\right\}$的过程，如下图所示：



信道合并是指N个独立且相同的 B-DMC 信道W以递归的方式组成一个新的矢量信道$W_{vec}: u_1^N \to y_1^N$，其中$N = 2^m, m \ge 0$。

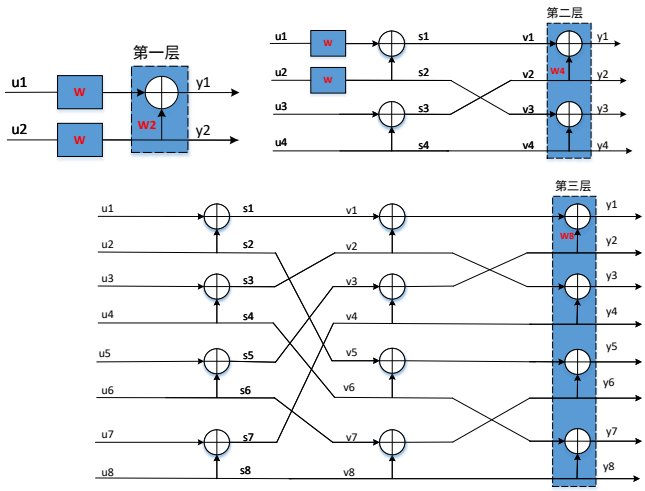


图 2-2 极化码分层极化示意图

在第一层，W_2由两个独立复制的信道W_1联合而成，$W_2: u_1^2 \to y_1^2$对应的转移概率为：

$$W_2(y_1, y_2|u_1, u_2) = W_2(y_1^2|u_1^2) = W_1(y_1|u_1 \oplus u_2) \cdot W_1(y_2|u_2)$$

在第二层，W_4由两个独立复制的信道W_2联合而成，$W_4: u_1^4 \to y_1^4$对应的转移概率为：

$$W_4(y_1^4|u_1^4) = W_2(y_1^2|u_1 \oplus u_2, u_3 \oplus u_4) \cdot W_2(y_3^4|u_2, u_4)$$

在第三层，W_8由两个独立复制的信道W_4联合而成，$W_8: u_1^8 \to y_1^8$对应的转移概率为：

$$W_8(y_1^8|u_1^8) = W_4(y_1^4|u_1 \oplus u_2, \oplus u_3 \oplus u_4, u_5 \oplus u_6, u_7 \oplus u_8) \cdot W_4(y_5^8|u_2, u_4, u_6, u_8)$$

一般地，每层递归都可以将两个独立复制的信道$W_{N/2}$通过概率转移关系式进行联合：

$$W_N(y_1^N|u_1^N) = W_{N/2}\left(y_1^{N/2}|u_{1,o}^N \oplus u_{1,e}^N\right) \cdot W_{N/2}\left(y_{N/2+1}^N|u_{1,e}^N\right)$$

式中，$u_{1,o}^N = (u_1, u_3, \dots, u_{N-1})$表示下标为奇数的子集，$u_{1,e}^N = (u_2, u_4, \dots, u_N)$表示下表为偶数的子集。

信道分裂的过程是指将矢量信道W_N分成N个二进制比特子信道$W_N^i: u_i \to y_1^N \times u_1^{i-1}, 1 \le i \le N$，第$i$个比特子信道的转移概率可以表示为矢量信道的转移概率

$$\boldsymbol{W_N^i = p(y_1, y_2, \dots, y_N, u_1, u_2, \dots, u_{i-1}|u_i) = p\left(y_1^N, u_1^{i-1}|u_i\right)}$$

$$\boldsymbol{W_N^i = p\left(y_1^N, u_1^{i-1}|u_i\right) = \frac{p\left(y_1^N, u_1^{i-1}, u_i\right)}{p(u_i)} = \frac{\sum_{u_{i+1}^N} p\left(y_1^N, u_1^{i-1}, u_i, u_{i+1}^N\right)}{p(u_i)} = \frac{\sum_{u_{i+1}^N} p\left(y_1^N, u_1^N\right)}{p(u_i)}}$$
$$\boldsymbol{= \frac{\sum_{u_{i+1}^N} p\left(y_1^N|u_1^N\right) \cdot p\left(u_1^N\right)}{p(u_i)}}$$

假设u_i为 0，1 均匀独立同分布，则$p(u_1^N) = \left(\frac{1}{2}\right)^N = \frac{1}{2^N}$，则

$$\boldsymbol{W_N^i = \frac{1}{2^{N-1}} \sum_{u_{i+1}^N} p\left(y_1^N|u_1^N\right)}$$

$$\boldsymbol{W_{2N}^{2i-1} = \frac{1}{2^{2N-1}} \sum_{u_{2i}^{2N}} p\left(y_1^{2N}|u_1^{2N}\right) = \frac{1}{2^{2N-1}} \sum_{u_{2i,o}^{2N}} \sum_{u_{2i,e}^{2N}} p\left(y_1^{2N}|u_{1,o}^{2N} \oplus u_{1,e}^{2N}, u_{1,e}^{2N}\right)}$$

$$\boldsymbol{= \frac{1}{2^{2N-1}} \sum_{u_{2i,o}^{2N}} \sum_{u_{2i,e}^{2N}} p\left(y_1^N|u_{1,o}^{2N} \oplus u_{1,e}^{2N}\right) \cdot p\left(y_{N+1}^{2N}|u_{1,e}^{2N}\right)}$$

$$\boldsymbol{= \frac{1}{2^{2N-1}} \sum_{u_{2i,e}^{2N}} \left(p\left(y_{N+1}^{2N}|u_{1,e}^{2N}\right) \sum_{u_{2i,o}^{2N}} p\left(y_1^N|u_{1,o}^{2N} \oplus u_{1,e}^{2N}\right) \right)}$$

其中，$\boldsymbol{u_{1,o}^{2N}}$表示序列$u_1, u_2, \dots, u_{2N}$中的偶数部分，$\boldsymbol{u_{1,e}^{2N}}$表示序列中奇数部分。无论$u_{1,e}^{2N}$取何值，$u_{1,o}^{2N} \oplus u_{1,e}^{2N}$的取值和$u_{1,o}^{2N}$遍历的值相同，因此$\sum_{u_{2i,o}^{2N}} p\left(y_{N+1}^{2N}|u_{1,o}^{2N} \oplus u_{1,e}^{2N}\right)$与$u_{1,e}^{2N}$的取值无关

$$\boldsymbol{W_{2N}^{2i-1}=\frac{1}{2^{2N-1}}\sum_{u_{2i,e}^{2N}}\left(p(y_{N+1}^{2N}|u_{1,e}^{2N})\sum_{u_{2i,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)}$$

$$=\frac{1}{2^{2N-1}}\left(\sum_{u_{2i,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})\right)\left(\sum_{u_{2i,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)$$

$$=\frac{1}{2}\sum_{u_{2i}}\left(\frac{1}{2^{N-1}}\sum_{u_{2i+1,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})\right)\left(\frac{1}{2^{N-1}}\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)$$

$\frac{1}{2^{N-1}}\sum_{u_{2i+1,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})$为下半部分的概率转移值，$\frac{1}{2^{N-1}}\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})$为上半部分的概率转移值

令上半部分$v_1=u_1\oplus u_2, v_2=u_3\oplus u_4, \dots, v_N=u_{2N-1}\oplus u_{2N}$

$$\frac{1}{2^{N-1}}\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})=\frac{1}{2^{N-1}}\sum_{v_{i+1}^N}p(y_1^N|v_1^N)=p(y_1,y_2,\dots,y_N,v_1,v_2,\dots,v_{i-1}|v_i)$$

$$=W_N^i(y_1^N,v_1^{i-1}|v_i)=W_N^i(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}|u_{2i-1}\oplus u_{2i})$$

令下半部分$z_1=u_2, z_2=u_4, \dots, z_N=u_{2N}$

$$\frac{1}{2^{N-1}}\sum_{u_{2i+1,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})=\frac{1}{2^{N-1}}\sum_{z_{i+1}^N}p(y_{N+1}^{2N}|z_1^N)=p(y_{N+1},y_{N+2},\dots,y_{2N},z_1,z_2,\dots,z_{i-1}|z_i)$$

$$=W_N^i(y_{N+1}^{2N},z_1^{i-1}|z_i)=W_N^i(y_{N+1}^{2N},u_{1,e}^{2i-2}|u_{2i})$$

$$W_{2N}^{2i}=\frac{1}{2^{2N-1}}\sum_{u_{2i+1,e}^{2N}}\left(p(y_{N+1}^{2N}|u_{1,e}^{2N})\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)$$

$$=\frac{1}{2^{2N-1}}\left(\sum_{u_{2i+1,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})\right)\left(\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)$$

$$=\frac{1}{2}\left(\frac{1}{2^{N-1}}\sum_{u_{2i+1,e}^{2N}}p(y_{N+1}^{2N}|u_{1,e}^{2N})\right)\left(\frac{1}{2^{N-1}}\sum_{u_{2i+1,o}^{2N}}p(y_1^N|u_{1,o}^{2N}\oplus u_{1,e}^{2N})\right)$$

因此

$$W_{2N}^{2i-1}=\frac{1}{2}\sum_{u_{2i}}W_N^i(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}|u_{2i-1}\oplus u_{2i})\cdot W_N^i(y_{N+1}^{2N},u_{1,e}^{2i-2}|u_{2i})$$

$$W_{2N}^{2i}=\frac{1}{2}W_N^i(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}|u_{2i-1}\oplus u_{2i})\cdot W_N^i(y_{N+1}^{2N},u_{1,e}^{2i-2}|u_{2i})$$

定义接收端u_i的似然比为：

$$LR_N^i(y_1^N, u_1^{i-1}) = \frac{p(y_1^N, u_1^{i-1}|u_i=0)}{p(y_1^N, u_1^{i-1}|u_i=1)} = \frac{W_N^i(y_1^N, u_1^{i-1}|u_i=0)}{W_N^i(y_1^N, u_1^{i-1}|u_i=1)}$$

$$LR_{2N}^{2i-1} = \frac{W_{2N}^{2i-1}(u_{2i-1}=0)}{W_{2N}^{2i-1}(u_{2i-1}=1)} = \frac{\frac{1}{2}\sum_{u_{2i}} W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|0 \oplus u_{2i}) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|u_{2i})}{\frac{1}{2}\sum_{u_{2i}} W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|1 \oplus u_{2i}) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|u_{2i})}$$

$$= \frac{\sum_{u_{2i}} W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|0 \oplus u_{2i}) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|u_{2i})}{\sum_{u_{2i}} W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|1 \oplus u_{2i}) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|u_{2i})}$$

令$(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}) = A, (y_{N+1}^{2N}, u_{1,e}^{2i-2}) = B$

$$LR_{2N}^{2i-1} = \frac{W_N^i(A|0 \oplus u_{2i}=0) \cdot W_N^i(B|u_{2i}=0) + W_N^i(A|0 \oplus u_{2i}=1) \cdot W_N^i(B|u_{2i}=1)}{W_N^i(A|1 \oplus u_{2i}=0) \cdot W_N^i(B|u_{2i}=0) + W_N^i(A|1 \oplus u_{2i}=1) \cdot W_N^i(B|u_{2i}=1)}$$

$$= \frac{W_N^i(A|0) \cdot W_N^i(B|0) + W_N^i(A|1) \cdot W_N^i(B|1)}{W_N^i(A|1) \cdot W_N^i(B|0) + W_N^i(A|0) \cdot W_N^i(B|1)}$$

$$= \frac{\frac{\boldsymbol{W_N^i(A|0) \cdot W_N^i(B|0)}}{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|1)}} + \frac{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|1)}}{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|1)}}}{\frac{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|0)}}{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|1)}} + \frac{\boldsymbol{W_N^i(A|0) \cdot W_N^i(B|1)}}{\boldsymbol{W_N^i(A|1) \cdot W_N^i(B|1)}}} = \frac{\frac{\boldsymbol{W_N^i(A|0)}}{\boldsymbol{W_N^i(A|1)}} \cdot \frac{\boldsymbol{W_N^i(B|0)}}{\boldsymbol{W_N^i(B|1)}} + \boldsymbol{1}}{\frac{\boldsymbol{W_N^i(B|0)}}{\boldsymbol{W_N^i(B|1)}} + \frac{\boldsymbol{W_N^i(A|0)}}{\boldsymbol{W_N^i(A|1)}}}$$

$$= \frac{\boldsymbol{LR_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}) \cdot LR_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}) + 1}}{\boldsymbol{LR_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}) + LR_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})}}$$

$$LR_{2N}^{2i} = \frac{W_{2N}^{2i}(u_{2i}=0)}{W_{2N}^{2i}(u_{2i}=1)} = \frac{\frac{1}{2}W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 0) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|0)}{\frac{1}{2}W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 1) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|1)}$$

$$= \frac{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 0) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|0)}{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 1) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|1)}$$

若$\hat{u}_{2i-1} = 0$

$$LR_{2N}^{2i} = \frac{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|0) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|0)}{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|1) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|1)} = \frac{W_N^i(A|0) \cdot W_N^i(B|0)}{W_N^i(A|1) \cdot W_N^i(B|1)}$$

$$= \frac{W_N^i(A|0)}{W_N^i(A|1)} \cdot \frac{W_N^i(B|0)}{W_N^i(B|1)} = LR_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}) \cdot LR_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})$$

若$\hat{u}_{2i-1} = 1$

$$LR_{2N}^{2i} = \frac{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|1) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|0)}{W_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|0) \cdot W_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2}|1)} = \frac{W_N^i(A|1) \cdot W_N^i(B|0)}{W_N^i(A|0) \cdot W_N^i(B|1)}$$

$$= \left(LR_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})\right)^{-1} \cdot LR_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})$$

因此

$$LR_{2N}^{2i} = \left(LR_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})\right)^{1-2\hat{u}_{2i-1}} \cdot LR_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})$$

综上所述:

$$LR_{2N}^{2i-1} = \frac{LR_N^i\left(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}\right) \cdot LR_N^i\left(y_{N+1}^{2N}, u_{1,e}^{2i-2}\right) + 1}{LR_N^i\left(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}\right) + LR_N^i\left(y_{N+1}^{2N}, u_{1,e}^{2i-2}\right)}$$

$$LR_{2N}^{2i} = \left(LR_N^i\left(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}\right)\right)^{1-2\hat{u}_{2i-1}} \cdot LR_N^i\left(y_{N+1}^{2N}, u_{1,e}^{2i-2}\right)$$

可以证明，随着N的增大，N个比特子信道中一部分会变成$C\left(W_N^i\right)$趋于 0 的纯噪声信道，另一部分变成$C\left(W_N^i\right)$趋于 1 的无噪声信道。在这样的极化信道上信道编码是非常简单的：只需要将所要传输的数据加载在$C(W)$趋近于 1 的那些信道上，而$C(W)$趋近于 0 的那些信道不使用（传输收发双方已知信号），就可以实现数据的可靠传输。

2.2 Polar 码编码方案

2.2.1 子信道可靠度评估和排序

Polar 码的关键就是将信息比特承载在经过信道合并和分裂得到的高可靠子信道上，因此子信道可靠度的评估和排序直接影响比特集合的选取，进而影响 Polar 码的性能。Polar 码子信道可靠度的评估方法有很多，目前主流的方法有密度演进以及极化权重构造等，本章节重点介绍 NR 标准中采用的极化权重构造法，对其它方法作简单介绍。

2.2.1.1 密度演进（DE）

DE 是一种跟踪消息概率密度在置信传输译码算法中进化情况的经典算法，而串行抵消译码算法是一种特性方向的 BP 算法，即自上而下的硬判决 BP。因此，DE 可以直接用于 Polar 码，通过跟踪译码过程评估子信道的可靠度。

基于 DE 的子信道可靠度计算最大的优点是可以利用当前信道的相关参数比较精确地估计每个子信道的可靠度。但是在递归操作中，需要对多维参数进行运算，复杂度高，不适用于实际的应用场景。针对一些特殊信道，DE 可以通过运算得到简化。

- 二进制擦除信道

对于极化信道W_N^i，其巴氏参数$Z(W_N^i)$可以通过递推公式计算得到：

$$\begin{cases} Z(W_{2N}^{2i-1}) = 2Z(W_N^i) - [Z(W_N^i)]^2 \\ Z(W_{2N}^{2i}) = [Z(W_N^i)]^2 \end{cases}$$

$$Z(W) = \sum_y \sqrt{p(y|x=0) \cdot p(y|x=1)}$$

对于二进制擦除信道，$Z(W) = p$

巴氏参数构造的核心思想是根据递推式获得$Z(W_N^i)$来估计子信道的可靠性，选取可靠性高的信道传输信息比特。若信息比特长度为K，则选取$Z(W)$最小的K个信道。但巴氏参数构造的缺点是适用范围很窄，只适用于二进制擦除信道。

- AWGN 信道

AWGN 信道的主要参数是噪声的均值和方差，对于一个噪声均值为 0，方差为σ^2的 AWGN 信道，假设发送端发送全 0 码字，且使用 BPSK 调制，通过 AWGN 信道接收得到 BPSK 符号为$y_i = 1 + n_i$，其中噪声n_i服从高斯分布$\mathcal{N}(0, \sigma^2)$，因此y_i服从高斯分布$\mathcal{N}(1, \sigma^2)$，因而接收到的y_i的对数似然比为：

$$L_N^i(y_i) = ln \frac{\frac{1}{\sqrt{2\pi}} e^{-(y_i-1)^2/2\sigma^2}}{\frac{1}{\sqrt{2\pi}} e^{-(y_i+1)^2/2\sigma^2}} = \frac{2y_i}{\sigma^2}$$

则$L_N^i(y_i)$服从$\mathcal{N}\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right)$，此处$L_N^i(y_i)$的方差是均值的 2 倍。

分裂后各个子信道的对数似然比的递推公式为：

$$L_{2N}^{2i-1} = ln \left(\frac{1 + tanh\left(\frac{L_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})}{2}\right) \cdot tanh\left(\frac{L_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})}{2}\right)}{1 - tanh\left(\frac{L_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})}{2}\right) \cdot tanh\left(\frac{L_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})}{2}\right)} \right)$$

$$= 2arctanh\left(tanh\left(\frac{L_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})}{2}\right) \cdot tanh\left(\frac{L_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})}{2}\right) \right)$$

$$L_{2N}^{2i} = (1 - 2\hat{u}_{2i-1}) \cdot L_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}) + L_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})$$

令$u = L_N^i(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2})$，$v = L_N^i(y_{N+1}^{2N}, u_{1,e}^{2i-2})$，则

$$L_{2N}^{2i-1} = 2arctanh\left(tanh\left(\frac{u}{2}\right)\cdot tanh\left(\frac{v}{2}\right)\right) \Longrightarrow$$

$$tanh\left(\frac{L_{2N}^{2i-1}}{2}\right) = tanh\left(\frac{u}{2}\right)\cdot tanh\left(\frac{v}{2}\right)$$

$$L_{2N}^{2i} = (1-2\hat{u}_{2i-1})\cdot u + v$$

由于u服从方差为均值两倍的高斯正态分布，令$E(u)=x$，则

$$E\left(tanh\left(\frac{u}{2}\right)\right) = \varphi\big(E(u)\big) = \varphi(x) = \frac{1}{\sqrt{4\pi x}}\int_{-\infty}^{+\infty} tanh\left(\frac{u}{2}\right) exp\left[-\frac{(u-x)^2}{4x}\right]du$$

注：高斯正态分布$\mathcal{N}(\mu,\sigma^2)$的概率密度函数为

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$tanh\left(\frac{L_{2N}^{2i-1}}{2}\right) = tanh\left(\frac{u}{2}\right)\cdot tanh\left(\frac{v}{2}\right)$$

此时u与v独立同分布，可得

$$E\left(tanh\left(\frac{L_{2N}^{2i-1}}{2}\right)\right) = \varphi\big(E(u)\big)\cdot\varphi\big(E(u)\big) \Longrightarrow$$

$$\varphi\left(E\left(L_{2N}^{2i-1}\right)\right) = \left[\varphi\big(E(u)\big)\right]^2$$

因此

$$E\left(L_{2N}^{2i-1}\right) = \varphi^{-1}\left(\left[\varphi\big(E(u)\big)\right]^2\right)$$

$$E\left(L_{2N}^{2i}\right) = 2E(u)$$

其中，函数$\varphi(x)$定义为

$$\varphi(x) = \begin{cases} \frac{1}{\sqrt{4\pi x}}\int_{-\infty}^{+\infty} tanh\left(\frac{u}{2}\right) exp\left[-\frac{(u-x)^2}{4x}\right]du, x>0 \\ 0, x=0 \end{cases}$$

$\varphi(x)$可进一步简化和近似为：

$$\varphi(x) = \begin{cases} 1-exp(-0.4527x^{0.86}+0.0218), 10>x>0 \\ 1-\sqrt{\frac{\pi}{x}}\left(1-\frac{9}{7x}\right)exp\left(-\frac{x}{4}\right), x>10 \end{cases}$$

对概率密度函数进行积分，即得到第i个子信道的错误概率为

$$P(i) = Q\sqrt{\frac{E\left(L_{2N}^{2i-1}\right)}{2}}$$

上述基于密度进化的评估方法可以精确地根据信道刻画子信道的可靠度，这些方法需要

准确的信道信息，如 AWGN 信道的接收信噪比等。在实现时，要么要求在每次编码前进行在线实时的可靠度计算，会给编译码带来额外的复杂度和延时，要么对不同码长、码率存储大量可靠度的计算结果。

2.2.1.2 极化权重（PW）

该方法通过一个简单的闭合公式对子信道可靠度相对关系进行表征，具有极低的复杂度，且评估过程与信道参数无关。另外，根据 PW 公式产生的可靠度排序序列，在相同母码长度和不同母码长度之间都具有嵌套性，可以以更低的复杂度支持在线或离线两种子信道可靠度评估。

对于第i个子信道W_N^i，其序号i的二进制表示为$B_0B_1 \dots B_{n-1}$，其中B_{n-1}是最高位，B_0是最低位，其可靠度可以由下式计算：

$$\boldsymbol{V\left(W_N^i\right) = \sum_{j=0}^{n-1} B_j \cdot 2^{j \times \frac{1}{4}}}$$

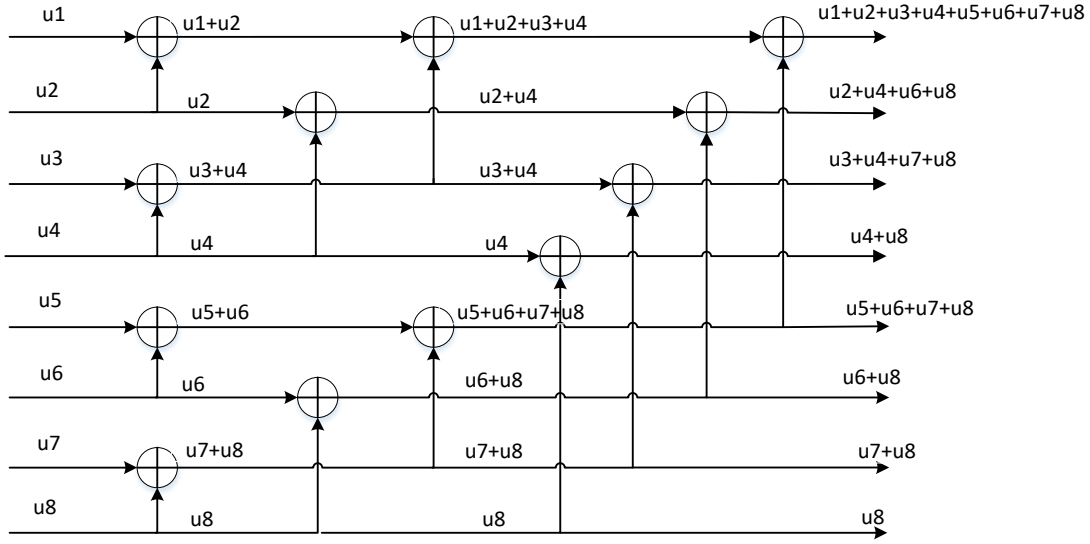
PW 方法中，每个子信道的可靠度由二进制表示中"1"的权重相加得到，且对应的权重与"1"在二进制表示中所在的位置相关。越靠近最低位，权重越低，反之，权重越高。这说明正向极化产生的子信道比反向极化产生的子信道具有更高的可靠度。通过仿真验证，为了保证有唯一解，$2^{1/4}$是最接近容量极限的。

PW 方式评估得到的子信道可靠度排序具有嵌套特性。嵌套特性之一是：针对(N, K_1)计算得到的信息比特集合是针对(N, K_2) $(K_2 > K_1)$的信息比特集合的一个子集。因此，可以针对给定的N，构造一个实用于$K = 1,2, \dots, N$的排序序列，序列的子集即响应的信息比特集合。嵌套特性之二是：子信道$i = 1,2, \dots, N/2$在针对$N/2$构造的排序序列与针对N构造的排序序列中的排序一致。PW 方法与信道参数无关，直接离线存储可靠度排序序列，可以避免 DE 方法中的在线构造，极大地降低了编译码地复杂度开销和时延。

2.2.2 编码

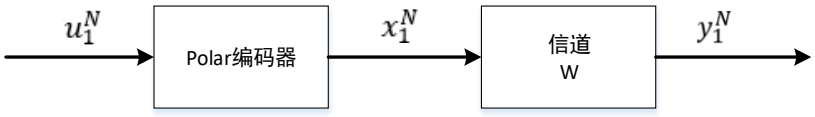
将信息比特放置在信息位，其它位放置事先确定好的冻结比特，组成信源序列u_1^N，经过生成矩阵生成码字$x_1^N = u_1^N G_N$。

$$G_N = F^{\otimes n},\ F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, F^{\otimes n} = \begin{bmatrix} F^{\otimes n-1} & 0 \\ F^{\otimes n-1} & F^{\otimes n-1} \end{bmatrix}$$



2.3 Polar 码译码方案

设 Polar 码的参数集合为(N, K, A, A^C)，N为编码后的比特长度，K为信息比特长度，A为信息比特集合，A^C为冻结比特集合，输入比特为$u_1^N = (u_1, u_2, \ldots, u_N)$，输出比特为$x_1^N = (x_1, x_2, \ldots, x_N)$，经过信道后的接收向量为$y_1^N = (y_1, y_2, \ldots, y_N)$。译码器的作用是根据接受序列$y_1^N$和编码参数得到信息序列$u_1^N$的估计值$\hat{u}_1^N$。由于译码器可以通过$\hat{u}_{A^C} = u_{A^C}$避免冻结比特的错误，则 Polar 译码器的真正任务是产生对信息比特u_A的估计值$\hat{u}_A$。



2.3.1 SC 译码

SC (successive cancellation) 串行抵消译码算法，是 Arikan 给出的 Polar code 译码算法，由于各个极化信道并不是相互独立的，而是具有确定的依赖关系，信道序号大的极化信道依赖于所有比其序号小的极化信道。基于极化信道之间的这一依赖关系，SC 译码算法对各个比特进行译码判决时，需要假设之前步骤译码得到的结果都是正确的。

SC 译码是将第$1,2,\dots,i-1$位译出的比特值用于第i位的译码，消除之前译出值的干扰，按照i从 1 到N的顺序计算：

$$\begin{aligned}&\boldsymbol{LR_N^i(u_i)}\\&=\frac{\boldsymbol{p(u_i=0|y_1,y_2,\dots,y_N,u_1,u_2,..,u_{i-1})}}{\boldsymbol{p(u_i=1|y_1,y_2,\dots,y_N,u_1,u_2,..,u_{i-1})}}\\&=\frac{\boldsymbol{p\left(u_i=0|y_1^N,u_1^{i-1}\right)}}{\boldsymbol{p\left(u_i=1|y_1^N,u_1^{i-1}\right)}}\\&=\frac{\boldsymbol{p\left(y_1^N,u_1^{i-1}|u_i=0\right)}}{\boldsymbol{p\left(y_1^N,u_1^{i-1}|u_i=1\right)}}\end{aligned}\qquad\text{公式（3-1）}$$

定义信道转移概率W_N^i为发送u_i时$y_1,y_2,\dots,y_N,u_1,u_2,..,u_{i-1}$的概率：

$$\begin{aligned}&\boldsymbol{W_N^i}\\&=\boldsymbol{p(y_1,y_2,\dots,y_N,u_1,u_2,\dots,u_{i-1}|u_i)}\\&=\boldsymbol{p\left(y_1^N,u_1^{i-1}|u_i\right)}\end{aligned}\qquad\text{公式（3-2）}$$

将公式 3-2 代入公式 3-1 中得到，

$$\boldsymbol{LR_N^i(u_i)=\frac{W_N^i(u_i=0)}{W_N^i(u_i=1)}}\qquad\text{公式（3-3）}$$

对公式 3-3 取对数得到 LLR 信息，

$$\begin{aligned}&\boldsymbol{LLR(u_i)}\\&=\boldsymbol{ln\frac{W_N^i(u_i=0)}{W_N^i(u_i=1)}}\\&=\boldsymbol{ln\frac{p\left(y_1^N,u_1^{i-1}|u_i=0\right)}{p\left(y_1^N,u_1^{i-1}|u_i=1\right)}}\end{aligned}\qquad\text{公式（3-4）}$$

对 LLR 进行应判决得到估计信息，

$$\widehat{\boldsymbol{u}}_{\boldsymbol{i}}=\begin{cases}\boldsymbol{0},\boldsymbol{u_i}\text{为冻结比特或}\boldsymbol{u_i}\text{为信息比特且}\boldsymbol{LLR}(\boldsymbol{u_i})>\boldsymbol{0}\\ \boldsymbol{1},\boldsymbol{u_i}\text{为信息比特且}\boldsymbol{LLR}(\boldsymbol{u_i})<\boldsymbol{0}\end{cases}$$

接下来给出一个最基本单元的 Polar 码译码器的过程：

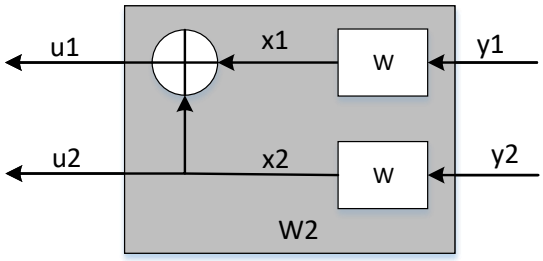


图 2-3 Polar 码基本单元译码流程示意图

一般假设$P(x_i=0)=P(x_i=1)=\frac{1}{2}$，在后续的描述过程中，$P(x)$为概率函数，$p(x)$为概率密度函数，$LR(x)$为似然比，$LLR(x)$为对数似然比。

$$\boldsymbol{LR}(\boldsymbol{x_1})=\frac{\boldsymbol{P}(\boldsymbol{x_1}=\boldsymbol{0}|\boldsymbol{y_1})}{\boldsymbol{P}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})}=\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0},\boldsymbol{y_1})/\boldsymbol{p}(\boldsymbol{y_1})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1},\boldsymbol{y_1})/\boldsymbol{p}(\boldsymbol{y_1})}=\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0},\boldsymbol{y_1})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1},\boldsymbol{y_1})}$$
$$=\frac{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{0})\cdot\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0})}{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{1})\cdot\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1})}=\frac{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{0})}{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{1})}$$

$$\boldsymbol{LR}(\boldsymbol{x_2})=\frac{\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}=\frac{\boldsymbol{p}(\boldsymbol{y_2}|\boldsymbol{x_2}=\boldsymbol{0})}{\boldsymbol{p}(\boldsymbol{y_2}|\boldsymbol{x_2}=\boldsymbol{1})}$$

$$\boldsymbol{LLR}(\boldsymbol{x_1})=\boldsymbol{ln}\left(\frac{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{0})}{\boldsymbol{p}(\boldsymbol{y_1}|\boldsymbol{x_1}=\boldsymbol{1})}\right),\boldsymbol{LLR}(\boldsymbol{x_2})=\boldsymbol{ln}\left(\frac{\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}\right)$$

当接收到y_1,y_2时，u_1,u_2的似然比计算过程如下：

$$\boldsymbol{u_1}=\boldsymbol{x_1}\oplus\boldsymbol{x_2}$$

$$\boldsymbol{u_2}=\boldsymbol{x_2}$$

$$\boldsymbol{LR}(\boldsymbol{u_1})=\frac{\boldsymbol{p}(\boldsymbol{u_1}=\boldsymbol{0}|\boldsymbol{y_1},\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{u_1}=\boldsymbol{1}|\boldsymbol{y_1},\boldsymbol{y_2})}=\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})+\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})+\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})}$$
$$=\frac{\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}+\boldsymbol{1}}{\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{0}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}+\frac{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{0}|\boldsymbol{y_2})}{\boldsymbol{p}(\boldsymbol{x_1}=\boldsymbol{1}|\boldsymbol{y_1})\cdot\boldsymbol{p}(\boldsymbol{x_2}=\boldsymbol{1}|\boldsymbol{y_2})}}$$
$$=\frac{\boldsymbol{LR}(\boldsymbol{x_1})\cdot\boldsymbol{LR}(\boldsymbol{x_2})+\boldsymbol{1}}{\boldsymbol{LR}(\boldsymbol{x_1})+\boldsymbol{LR}(\boldsymbol{x_2})}$$

在计算u_2时，此时u_1已经成功译码，则可作为u_2译码时的先验信息：
若$\hat{u}_1=0$，此时存在两种情况

Case1:$x_1=0,x_2=0,u_2=0$

Case2: $x_1=1,x_2=1,u_2=1$

$$\boldsymbol{LR(u_2)=\frac{p(u_2=0|y_1,y_2)}{p(u_2=1|y_1,y_2)}=\frac{p(x_1=0|y_1)\cdot p(x_2=0|y_2)}{p(x_1=1|y_1)\cdot p(x_2=1|y_2)}=LR(x_1)\cdot LR(x_2)}$$

若$\hat{u}_1=1$，此时存在两种情况

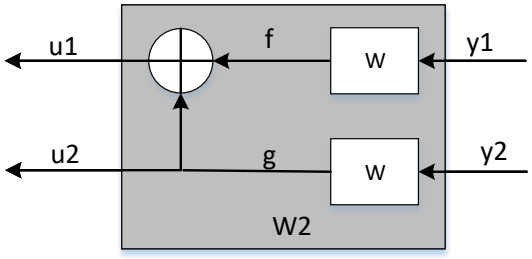
Case1:$x_1=0,x_2=1,u_2=1$

Case2:$x_1=1,x_2=0,u_2=0$

$$\boldsymbol{LR(u_2)=\frac{p(u_2=0|y_1,y_2)}{p(u_2=1|y_1,y_2)}=\frac{p(x_1=1|y_1)\cdot p(x_2=0|y_2)}{p(x_1=0|y_1)\cdot p(x_2=1|y_2)}=LR(x_2)/LR(x_1)}$$

$$\boldsymbol{LR(u_2)=LR(x_1)^{1-2\hat{u}_1}\cdot LR(x_2)}$$

为了便于后面描述简单起见，将x_1定义为f节点，将x_2定义为g节点，



$$LR(x_1)=LR(f),LR(x_2)=LR(g)$$

$$LR(u_1)=\frac{LR(f)\cdot LR(g)+1}{LR(f)+LR(g)}$$

$$\boldsymbol{LR(u_2)=LR(f)^{1-2\hat{u}_1}\cdot LR(g)}$$

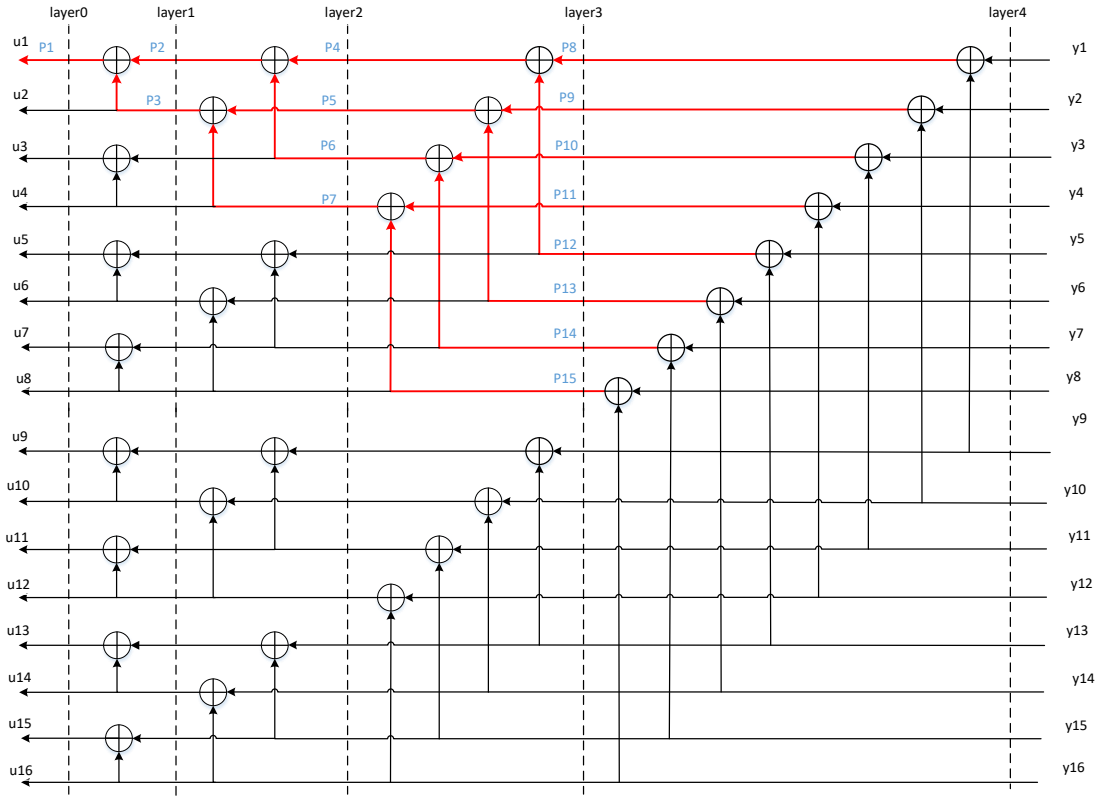
令$LR(f)=e^{LLR(f)}$，$LR(g)=e^{LLR(g)}$,则

$$LLR(u_1)=ln\frac{e^{(LLR(f)+LLR(g))}+1}{e^{LLR(f)}+e^{LLR(g)}}\approx sign\big(LLR(f)\big)\cdot sign\big(LLR(g)\big)\cdot min(|LLR(f)|,|LLR(g)|)$$

$$LLR(u_2)=(1-2\hat{u}_1)\cdot LLR(f)+LLR(g)$$

接下来以$N=16$为例，给出 SC 的译码过程：

Step1: 译码u_1；



$$P_8 = sign\big(LLR(y_1)\big) \times sign\big(LLR(y_9)\big) \times min(|LLR(y_1)|, |LLR(y_9)|)$$

$$P_9 = sign\big(LLR(y_2)\big) \times sign\big(LLR(y_{10})\big) \times min(|LLR(y_2)|, |LLR(y_{10})|)$$

$$P_{10} = sign\big(LLR(y_3)\big) \times sign\big(LLR(y_{11})\big) \times min(|LLR(y_3)|, |LLR(y_{11})|)$$

$$P_{11} = sign\big(LLR(y_4)\big) \times sign\big(LLR(y_{12})\big) \times min(|LLR(y_4)|, |LLR(y_{12})|)$$

$$P_{12} = sign\big(LLR(y_5)\big) \times sign\big(LLR(y_{13})\big) \times min(|LLR(y_5)|, |LLR(y_{13})|)$$

$$P_{13} = sign\big(LLR(y_6)\big) \times sign\big(LLR(y_{14})\big) \times min(|LLR(y_6)|, |LLR(y_{14})|)$$

$$P_{14} = sign\big(LLR(y_7)\big) \times sign\big(LLR(y_{15})\big) \times min(|LLR(y_7)|, |LLR(y_{15})|)$$

$$P_{15} = sign\big(LLR(y_8)\big) \times sign\big(LLR(y_{16})\big) \times min(|LLR(y_8)|, |LLR(y_{16})|)$$

$$P_4 = sign(P_8) \times sign(P_{12}) \times min(|P_8|, |P_{12}|)$$

$$P_5 = sign(P_9) \times sign(P_{13}) \times min(|P_9|, |P_{13}|)$$

$$P_6 = sign(P_{10}) \times sign(P_{14}) \times min(|P_{10}|, |P_{14}|)$$

$$P_7 = sign(P_{11}) \times sign(P_{15}) \times min(|P_{11}|, |P_{15}|)$$

$$P_2 = sign(P_4) \times sign(P_6) \times min(|P_4|, |P_6|)$$

$$P_3 = sign(P_5) \times sign(P_7) \times min(|P_5|, |P_7|)$$

$$LLR(u_1) = P_1 = sign(P_2) \times sign(P_3) \times min(|P_2|, |P_3|)$$

$$C(1,1) = \hat{u}_1$$

Step2:译码u_2；此时u_1已经成功译码，u_2处于u_1对应的 g 函数的位置。

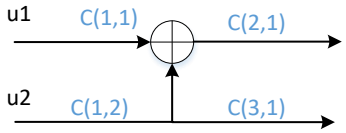
$$\boldsymbol{LLR(u_2) = (1 - 2\hat{u}_1) \cdot P_2 + P_3 = \big(1 - 2C(1,1)\big) \cdot P_2 + P_3}$$

$$\boldsymbol{C(1,2) = \hat{u}_2}$$

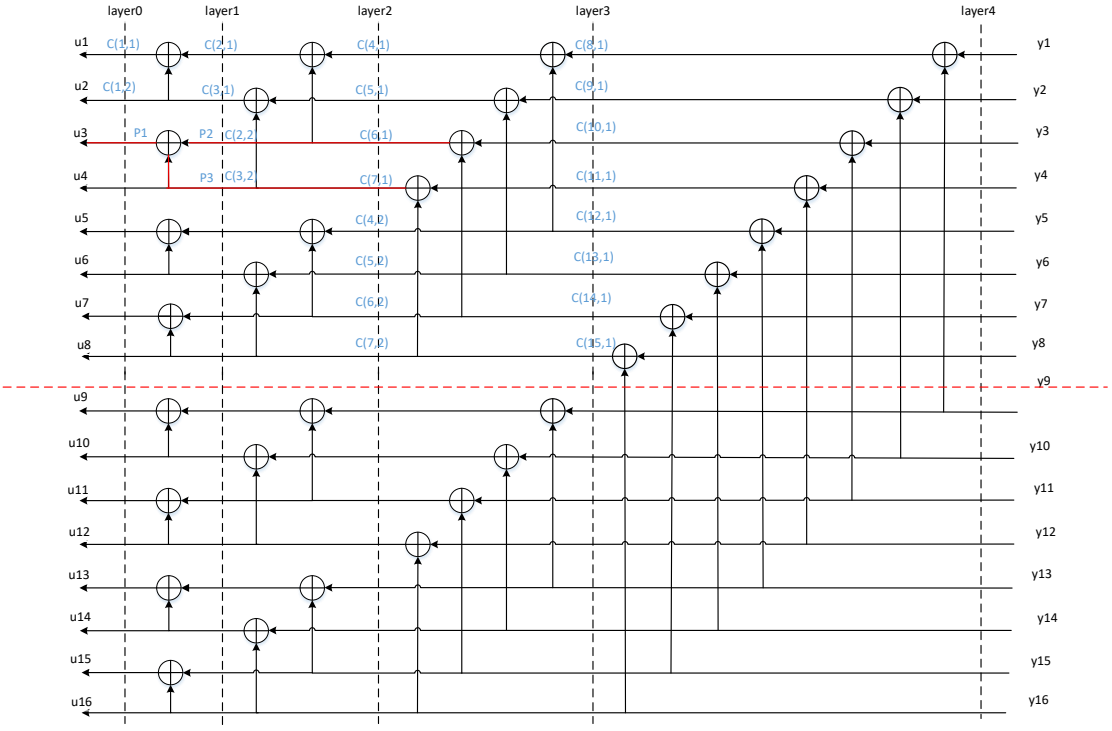
此时u_1和u_2均已成功译码，更新对应的编码节点位置的结果：

$$\boldsymbol{C(2,1) = mod(C(1,1) + C(1,2), 2)}$$

$$\boldsymbol{C(3,1) = C(1,2)}$$



Step3:译码u_3，此时已经获得$C(2,1)$和$C(3,1)$编码节点值，需重新更新对应P_2和P_3的对数似然比值，需要注意的是 P1 表示当前译码比特对应的软信息，P2、P3 表示计算当前译码比特所用到输入的软信息，因此 P1、P2、P3 在译码u_3时均需要计算更新：



$$\boldsymbol{P_2 = \big(1 - 2(\hat{u}_1 + \hat{u}_2)\big) \cdot P_4 + P_6 = \big(1 - 2C(2,1)\big) \cdot P_4 + P_6}$$

$$\boldsymbol{P_3 = (1 - 2\hat{u}_2) \cdot P_5 + P_7 = \big(1 - 2C(3,1)\big) \cdot P_5 + P_7}$$

$$\boldsymbol{LLR(u_3) = P_1 = sign(P_2) \times sign(P_3) \times min(|P_2|, |P_3|)}$$

$$\boldsymbol{C(1,1) = \hat{u}_3}$$

Step4:译码u_4，u_4处于u_3对应的 g 函数位置；

$$\boldsymbol{LLR(u_4) = P_1 = (1 - 2\hat{u}_3) \cdot P_2 + P_3 = \big(1 - 2C(1,1)\big) \cdot P_2 + P_3}$$

此时u_1, u_2, u_3, u_4均已成功译码，更新对应编码节点的结果；

$$\boldsymbol{C(1,2) = \hat{u}_4}$$

$$\boldsymbol{C(2,2) = C(1,1) + C(1,2)}$$

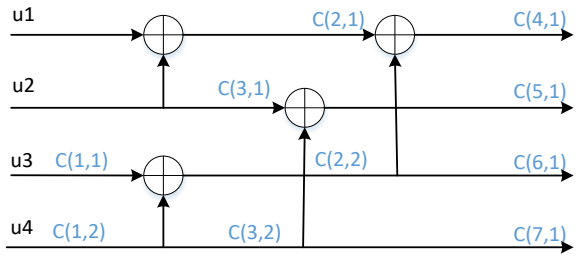
$$\boldsymbol{C(3,2) = C(1,2)}$$

$$\boldsymbol{C(4,1) = C(2,1) + C(2,2)}$$

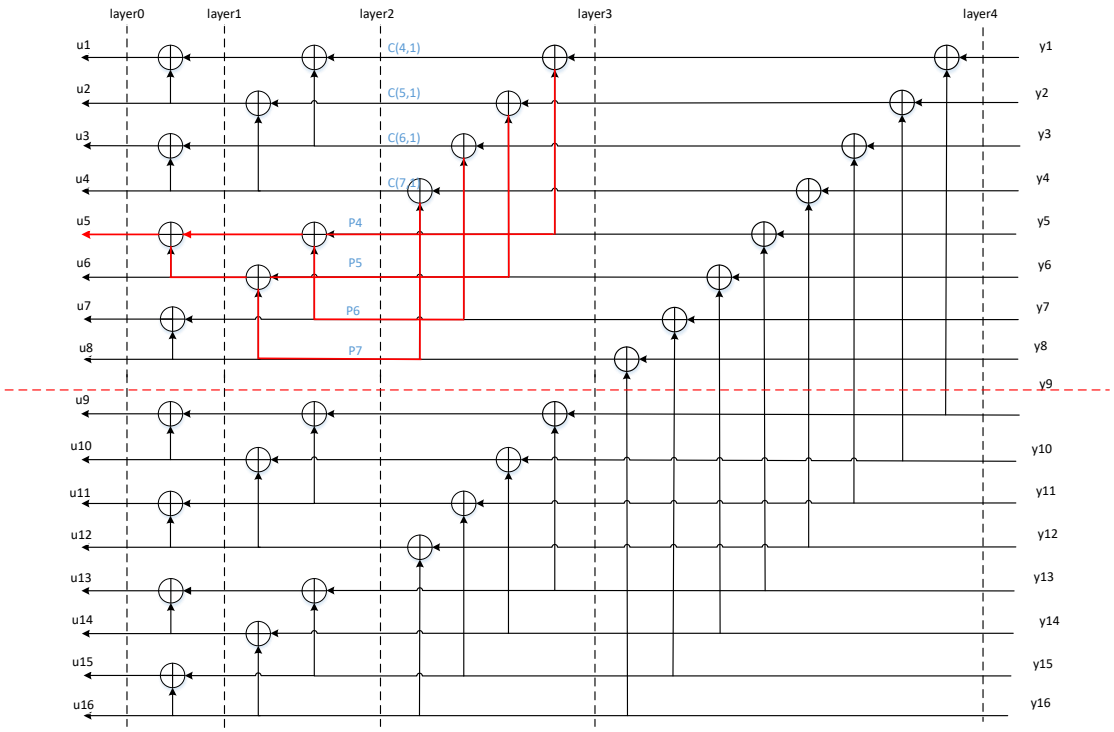
$$\boldsymbol{C(5,1) = C(3,1) + C(3,2)}$$

$$\boldsymbol{C(6,1) = C(2,2)}$$

$$\boldsymbol{C(7,1) = C(3,2)}$$



Step5:译码u_5；此时已经获得$C(4,1)$，$C(5,1)$，$C(6,1)$和$C(7,1)$编码节点值，需要重新更新对应P_4, P_5, P_6, P_7的对数似然比值：



$$\boldsymbol{P_4 = \left(1 - 2(\hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4)\right) \cdot P_8 + P_{12} = \left(1 - 2C(4,1)\right) \cdot P_8 + P_{12}}$$

$$\boldsymbol{P_5 = \left(1 - 2(\hat{u}_2 + \hat{u}_4)\right) \cdot P_9 + P_{13} = \left(1 - 2C(5,1)\right) \cdot P_9 + P_{13}}$$

$$\boldsymbol{P_6 = \left(1 - 2(\hat{u}_3 + \hat{u}_4)\right) \cdot P_{10} + P_{14} = \left(1 - 2C(6,1)\right) \cdot P_{10} + P_{14}}$$

$$\boldsymbol{P_7 = (1 - 2\hat{u}_4) \cdot P_{11} + P_{15} = \left(1 - 2C(7,1)\right) \cdot P_{11} + P_{15}}$$

$$\boldsymbol{P_2 = sign(P_4) \times sign(P_6) \times min(|P_4|, |P_6|)}$$

$$\boldsymbol{P_3 = sign(P_5) \times sign(P_7) \times min(|P_5|, |P_7|)}$$

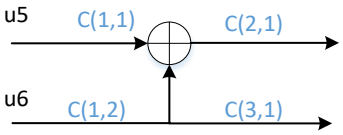
$$\boldsymbol{LLR(u_5) = P_1 = sign(P_2) \times sign(P_3) \times min(|P_2|, |P_3|)}$$

$$\boldsymbol{C(1,1) = \hat{u}_5}$$

Step6:译码u_6；u_6处于u_5对应的 g 函数位置；

$$\boldsymbol{LLR(u_5) = P_1 = (1 - 2\hat{u}_5) \cdot P_2 + P_3 = \left(1 - 2C(1,1)\right) \cdot P_2 + P_3}$$

此时$u_1, u_2, u_3, u_4, u_5, u_6$均已成功译码，更新对应编码节点的结果；



$$\boldsymbol{C(2,1) = C(1,1) + C(1,2)}$$

$$\boldsymbol{C(3,1) = C(1,2)}$$

Step7:译码u_7；此时已经获得$C(2,1)$和$C(3,1)$编码节点值，需要重新更新对应P_2和P_3的对数似然比值:

$$\boldsymbol{P_2 = \left(1 - 2(\hat{u}_5 + \hat{u}_6)\right) \cdot P_4 + P_6 = \left(1 - 2C(2,1)\right) \cdot P_4 + P_6}$$

$$\boldsymbol{P_3 = (1 - 2\hat{u}_6) \cdot P_5 + P_7 = \left(1 - 2C(3,1)\right) \cdot P_5 + P_7}$$

$$\boldsymbol{LLR(u_7) = P_1 = sign(P_2) \times sign(P_3) \times min(|P_2|, |P_3|)}$$

$$\boldsymbol{C(1,1) = \hat{u}_7}$$

Step8:译码u_8，u_8处于u_7对应的 g 函数位置;

$$\boldsymbol{LLR(u_8) = P_1 = (1 - 2\hat{u}_7) \cdot P_2 + P_3 = \left(1 - 2C(1,1)\right) \cdot P_2 + P_3}$$

此时$u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8$均已成功译码，更新对应编码节点的结果;

$$\boldsymbol{C(1,2) = \hat{u}_8}$$

$$\boldsymbol{C(2,2) = C(1,1) + C(1,2)}$$

$$\boldsymbol{C(3,2) = C(1,2)}$$

$$\boldsymbol{C(4,2) = C(2,1) + C(2,2)}$$

$$\boldsymbol{C(5,2) = C(3,1) + C(3,2)}$$

$$\boldsymbol{C(6,2) = C(2,2)}$$

$$\boldsymbol{C(7,2) = C(3,2)}$$

$$\boldsymbol{C(8,1) = C(4,1) + C(4,2)}$$

$$\boldsymbol{C(9,1) = C(5,1) + C(5,2)}$$

$$\boldsymbol{C(10,1) = C(6,1) + C(6,2)}$$

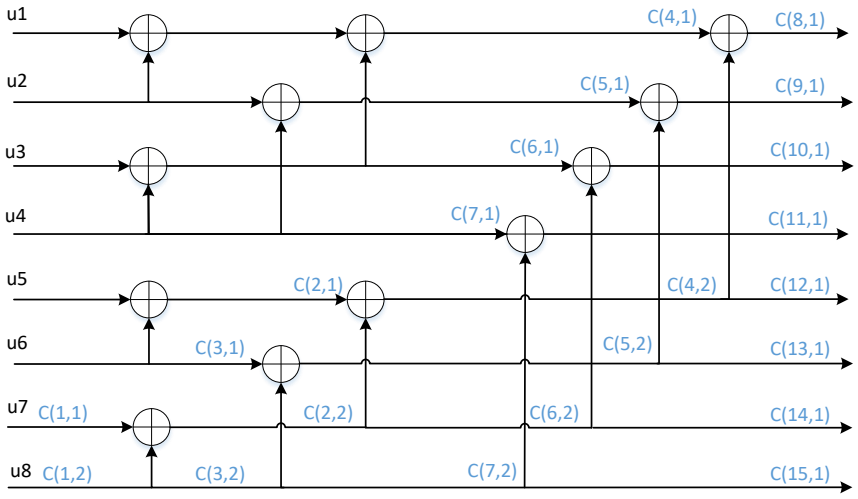
$$\boldsymbol{C(11,1) = C(7,1) + C(7,2)}$$

$$\boldsymbol{C(12,1) = C(4,2)}$$

$$\boldsymbol{C(13,1) = C(5,2)}$$

$$\boldsymbol{C(14,1) = C(6,2)}$$

$$\boldsymbol{C(15,1) = C(7,2)}$$



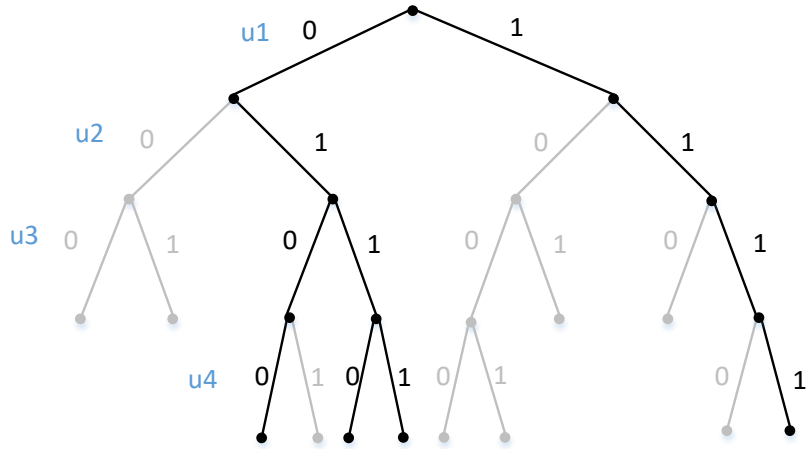
2.3.2 SCL 译码

虽然 Polar 码的 SC 译码算法非常简单，但在依次进行译码时需要假设前面译码得到的比特是正确的，当前面消息比特的译码发生错误之后，由于 SC 译码器在对后面的消息比特译码时需要用到之前的消息比特的估计值，这就会导致较为严重的错误传递，因此有了后来的 SCL 译码。SCL 译码相对于 SC 译码一个直接的改进就是，在每一层路径搜索后从允许保留最好的一条路径增加到允许保留 L 条候选路径用于下一步扩展，在译码过程，L 条路径可以同时存在，这样就增大了正确译码的可能性。通常，L 的值越大，误码性能越好，但译码的复杂度会增大，译码所需的时间也会更长，同时对应的存储器也会增加，另一方面也会增加下行控制信息的虚警概率。

在译码过程中要保持译码路径大小为 L，需要按照一定的规则对所有路径排序，从而对路径进行删除，或保留的操作。每个阶段译码都要保持最好的 L 条路径。在译码最后一个比特时，根据特定的规则选取最有可能正确的路径作为最终的译码路径。

下图给出了 L=4 非冻结比特的译码路径图。译码时，每一个比特最多保留 4 条译码路

径，并在这些留下的路径上继续译码，黑色路径为保留路径，灰色路径为舍弃路径。



接下来给出路径选择的度量函数PM_l^i的计算过程：

$$PM_l^i = p(u_1^i|y_1^N)$$

即在接收为$y_1, y_2, \dots, y_N$时，$u_1, u_2, \dots, u_i$的概率密度函数值

$$\boldsymbol{p(u_1^i|y_1^N) = \frac{p(u_1^i, y_1^N)}{p(y_1^N)}, p(u_1^{i-1}|y_1^N) = \frac{p(u_1^{i-1}, y_1^N)}{p(y_1^N)}}$$

$$\boldsymbol{\frac{p(u_1^{i-1}|y_1^N)}{p(u_1^i|y_1^N)} = \frac{p(u_1^{i-1}, y_1^N)}{p(u_1^i, y_1^N)}}$$

$$\boldsymbol{= \frac{p(u_1^{i-1}, y_1^N|u_i = 0) \cdot p(u_i = 0) + p(u_1^{i-1}, y_1^N|u_i = 1) \cdot p(u_i = 1)}{p(u_1^{i-1}, y_1^N|u_i) \cdot p(u_i)}}$$

$$\boldsymbol{= \frac{p(u_1^{i-1}, y_1^N|u_i = 0) + p(u_1^{i-1}, y_1^N|u_i = 1)}{p(u_1^{i-1}, y_1^N|u_i)}}$$

若$u_i = 0$

$$\boldsymbol{\frac{p(u_1^{i-1}|y_1^N)}{p(u_1^i|y_1^N)} = \frac{p(u_1^{i-1}, y_1^N)}{p(u_1^i, y_1^N)} = \frac{p(u_1^{i-1}, y_1^N|u_i = 0) + p(u_1^{i-1}, y_1^N|u_i = 1)}{p(u_1^{i-1}, y_1^N|u_i = 0)}}$$

$$\boldsymbol{= 1 + \frac{p(u_1^{i-1}, y_1^N|u_i = 1)}{p(u_1^{i-1}, y_1^N|u_i = 0)}}$$

若$u_i = 1$

$$\boldsymbol{\frac{p(u_1^{i-1}|y_1^N)}{p(u_1^i|y_1^N)} = \frac{p(u_1^{i-1}, y_1^N)}{p(u_1^i, y_1^N)} = \frac{p(u_1^{i-1}, y_1^N|u_i = 0) + p(u_1^{i-1}, y_1^N|u_i = 1)}{p(u_1^{i-1}, y_1^N|u_i = 1)}}$$

$$\boldsymbol{= 1 + \frac{p(u_1^{i-1}, y_1^N|u_i = 0)}{p(u_1^{i-1}, y_1^N|u_i = 1)}}$$

综上所述

$$\frac{p(u_1^{i-1}|y_1^N)}{p(u_1^i|y_1^N)}=\frac{p(u_1^{i-1},y_1^N)}{p(u_1^i,y_1^N)}=1+\left(\frac{p(u_1^{i-1},y_1^N|u_i=0)}{p(u_1^{i-1},y_1^N|u_i=1)}\right)^{-(1-2u_i)}$$

则

$$\frac{p(u_1|y_1^N)}{p(u_1^i|y_1^N)}=\frac{p(u_1|y_1^N)}{p(u_1^2|y_1^N)}\cdot\frac{p(u_1^2|y_1^N)}{p(u_1^3|y_1^N)}\cdots\frac{p(u_1^{i-1}|y_1^N)}{p(u_1^i|y_1^N)}=\prod_{j=2}^{i}1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}$$

$$\frac{1}{p(u_1^i|y_1^N)}=\frac{1}{p(u_1|y_1^N)}\prod_{j=2}^{i}1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}$$

$$\frac{1}{p(u_1|y_1^N)}=\frac{p(y_1^N)}{p(u_1,y_1^N)}=\frac{p(y_1^N|u_1=0)+p(y_1^N|u_1=1)}{p(y_1^N|u_1)}=1+\left(\frac{p(y_1^N|u_1=0)}{p(y_1^N|u_1=1)}\right)^{-(1-2u_1)}$$

$$\frac{1}{p(u_1^i|y_1^N)}=\prod_{j=1}^{i}1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}$$

取对数得

$$ln\frac{1}{p(u_1^i|y_1^N)}=\sum_{j=1}^{i}ln\left(1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}\right)$$

$$-ln\left(p(u_1^i|y_1^N)\right)=\sum_{j=1}^{i}ln\left(1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}\right)$$

$$ln\left(p(u_1^i|y_1^N)\right)=-\sum_{j=1}^{i}ln\left(1+\left(\frac{p(u_1^{j-1},y_1^N|u_j=0)}{p(u_1^{j-1},y_1^N|u_j=1)}\right)^{-(1-2u_j)}\right)$$

$$LR_N^i(y_1^N,u_1^{i-1})=\frac{p(y_1^N,u_1^{i-1}|u_i=0)}{p(y_1^N,u_1^{i-1}|u_i=1)}$$

其中$LR_N^i(y_1^N,u_1^{i-1})$的计算方案和 SC 译码过程中的$LR_N^i(y_1^N,u_1^{i-1})$计算方案相同。度量函数的定义为:

$$
\begin{aligned}
&\boldsymbol{PM_l^i} \\
&= \boldsymbol{ln}\left(\boldsymbol{p}\left(\boldsymbol{u_1^i}|\boldsymbol{y_1^N}\right)\right) \\
&= \sum_{\boldsymbol{j=1}}^{\boldsymbol{i}} \boldsymbol{ln}\left(\mathbf{1}+\boldsymbol{exp}\left(-\left(\mathbf{1}-\mathbf{2}\hat{\boldsymbol{u}}_{\boldsymbol{j}}[\boldsymbol{l}]\right)\boldsymbol{LLR_N^j}\right)\right)
\end{aligned}
\qquad \text{公式（4-1）}
$$

$$
\boldsymbol{LLR_N^j} = \boldsymbol{L_N^j} = \boldsymbol{ln}\left(\frac{\boldsymbol{p}\left(\boldsymbol{y_1^N}, \boldsymbol{u_1^{j-1}}|\boldsymbol{u_j}=\mathbf{0}\right)}{\boldsymbol{p}\left(\boldsymbol{y_1^N}, \boldsymbol{u_1^{j-1}}|\boldsymbol{u_j}=\mathbf{1}\right)}\right)
$$

度量函数越小，则概率值越高，简化后表达式的含义为：若根据 LLR 符号判断的比特与当前路径上的比特估计是一致的，即$L_N^i > 0, u_i = 0, L_N^i < 0, u_i = 1$，则$1 + exp\left(-\left(1 - 2\hat{u}_j[l]\right)\right) L_N^j \approx 1$;若 LLR 符号判断的比特与当前路径上的比特估计不一致,则$1 + exp\left(-\left(1 - 2\hat{u}_j[l]\right)\right) L_N^j \approx exp\left(\left|L_N^j\right|\right)$。

简化后的数据得：

$$
PM_l^i = \begin{cases} PM\left(u_1^{i-1}\right) & u_i\text{为信息比特或正确的冻结比特，} 1-2u_i = sign\left(L_N^i\right) \\ PM\left(u_1^{i-1}\right) + \left|L_N^i\left(y_1^N, u_1^{i-1}\right)\right| & u_i\text{为信息比特或正确的冻结比特，} 1-2u_i \neq sign\left(L_N^i\right) \\ -\infty & u_i\text{为错误的冻结比特} \end{cases}
$$

最终译码判决由最后一层 L 条路径的最大条件概率来选择，即$l_{min} = min(PM_l^N)$。

将第l_{min}路径上保存的各个译码结果取出来即为最终的译码结果。

2.3.3 特殊节点处理

译码过程中，存在一些特殊节点，可以在快速处理多个比特的判决，在不影响可靠性的前提下降低计算复杂度和时延。首先介绍极化码的树形图表示形式，第 i 层（stage i）包含$2^{\log N - i + 1}$个节点，每个节点包含2^{N-1}个比特，其中 stage1 的节点为叶子节点，每个叶子节点包含一个冻结比特或信息比特，stage$\log N + 1$为根节点，包含 N 个编码后的比特。

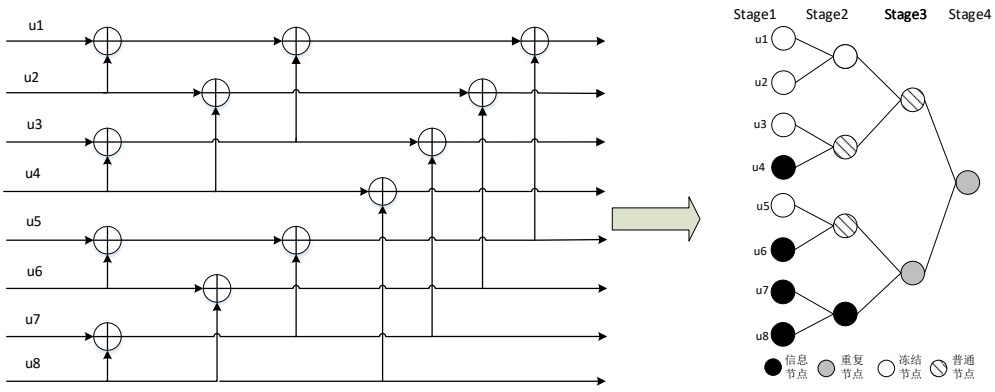
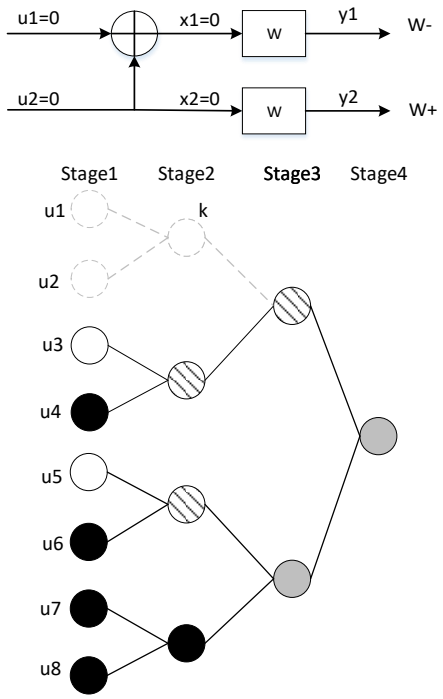


图 2-4 极化码树状图表示

2.3.3.1 冻结节点

如果每个节点的叶子节点均为冻结比特则该节点为冻结节点，即该节点包含的所有比特在译码前就可以确定。以基本的极化单元为例，如果一个极化单元的输入（u1，u2）为冻结比特（即输入均为“0”），则其输出（x1,x2）也均为 0。图中 stage2 的节点 k 为冻结节点。当译码至该节点时，可以对 u1、u2 直接进行判决（判决为“0”），不需要继续向左进行译码，可以节约 f 函数和 g 函数的运算。根据前文表述，比特判决之后需要向右更新每个节点的比特结果用来计算 g 函数，冻结节点的引入也节约了向右更新比特的计算。



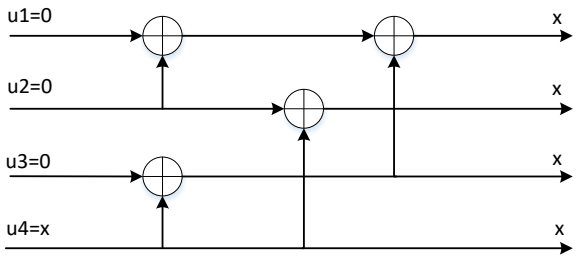
由于节点 k 包含叶子节点均为的冻结比特，因此也无需进行路径扩展。当前译码路径

的 PM 值按照下式更新，即对该节点所有非正的 LLR 值（即与比特"0"不符的 LLR 值）进行绝对值的累加作为当前路径的 PM 更新值。

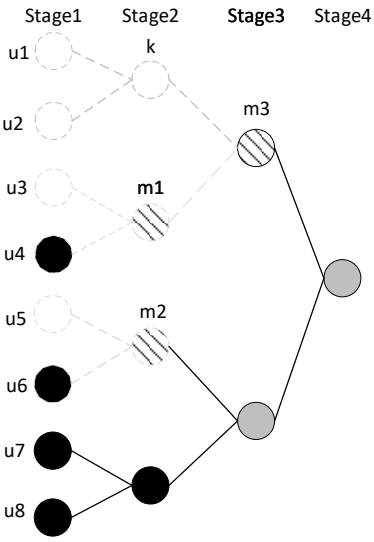
$$PM_l = PM_l + \sum_i \left(\frac{1}{2}|1 - sign(llr_i)| * |llr_i|\right)$$

2.3.3.2 重复节点

除冻结节点外还有一类特殊节点，即重复节点。如果一个节点的叶子节点仅包含一个信息比特（由极化特性可知，如果一个节点仅包含一个信息比特，那么该信息比特一定位于该节点的最后一个叶子节点），那么该节点包含的所有比特理论上全部等于该信息比特的值，即为该信息比特的重复，如下图所示。



需要注意的是，如果节点的两个子节点分别为冻结节点和重复节点，那么该节点所包含的所有比特仍为重复子节点中信息比特的重复，则该节点为更大的重复节点。下图中，stage1 的 u3 和 u4 在 stage2 构成重复节点 m1，u5 和 u6 在 stage2 构成重复节点 m2，而重复节点 m1 和冻结节点 k 又在 stage3 中构成更大的重复节点 m3。



因为重复节点仅包含一个信息比特，因此当前路径共有两条可能的扩展路径。一条为信息比特为"0"的路径，则这条路径下该重复节点包含的所有比特均判决为"0"，其叶子节点也判决为"0"。另一条为信息比特为"1"的路径，则这条路径下该重复节点包含的所有比特均判决"1"，其叶子节点中信息比特（最后一个叶子节点）判决为"1"，冻结比特（除最后一个叶子节点外的其他叶子节点）判决为"0"。两条扩展路径的度量值分别为：

$$PM_l = PM_l + \sum_i \left(\frac{1}{2} \left| (-1)^{\hat{u}_i} - sign(llr_i) \right| * |llr_i| \right)$$

2.3.4 CA-SCL 译码

虽然 SCL 译码算法较 SC 译码算法能明显提高 BLER 性能，但译码结束时的 L 条幸存路径中路径度量值最大的未必是正确的路径，由于编码中引入了 CB-CRC，所以可以利用 CRC 的校验功能辅助路径识别，从而进一步提升 Polar 码的纠错性能。

在 CRC-aided 辅助 SCL 译码算法中，最终的译码判决并非像 SCL 译码由最后一层 L 条路径的最大条件概率来选择，而是对 L 条候选路径进行校验，可靠度最大且通过 CRC 路径上的译码序列作为最终的译码输出。可以把 CRC 校验码和 Polar 码看作是级联关系，其中 Polar 码作为内码，CRC 校验码作为外码，通过 CRC 辅助选取最优路径能大大提高 Polar 码的译码性能。

1.3.1 节提到，极化码编码前的 CRC 校验比特经过比特交织后，有 7 比特校验比特分布在信息比特中间，成为分布式 CRC 比特。这些比特可以用来提前终止译码，分布式 CRC 校验又分为两种方法，即检查删除（CD）和检查保持（CK）。

2.3.4.1 检查删除

CA-SCL 译码算法中，CRC 校验的一个明显的用处就是用来对译码路径进行校验，删除非法路径，为其他潜在的合法路径腾出空间，这就是检查删除（CK，Check-Delete）。这种方法对于分布式 CRC 比特充分利用，没有通过校验的路径会在即时删除，如果在某一次校验中所有的路径都没有通过校验则停止译码。CRC-CK 的优点是降低了误码率，但同时由于不断引入新的译码路径，在一定程度上会提高译码的虚警概率。

2.3.4.2 检查保持

与检查删除相对的是检查保持（CK，Check-Keep），这种方法对于没有通过校验的译码路径并不删除，即仍然存在于幸存译码路径列表中，这些路径会标记为非法路径，在之后的译码过程中不再进行路径译码计算、路径分裂、路径惩罚值计算等操作，同时也不会引入新的译码路径。当所有的幸存路径都被标记为非法路径时则停止译码

由于 CRC-CK 在译码过程中不会引入新的译码路径，因此其误码率相对于 CRC-CD 会更高，换来的好处是更低的虚警概率。

3 Polar 译码算法实现

3.1 总述

本实现方案包含 NR PDCCH 链路从解扰开始到译码结束的整个外接收机处理流程，译

码算法实现的总体思路是基于译码树状图分层处理。主要模块包括预处理模块、LLR 计算模块、部分和计算、lazy copy 模块、路径处理模块、结果存储与回溯。具体信号处理流程如下图所示

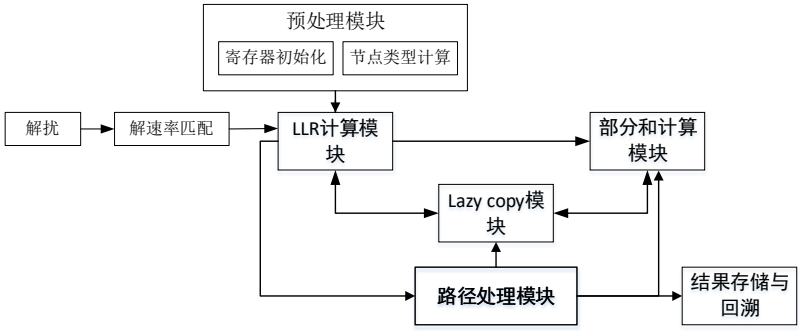


图 3-1 极化码译码器总体架构

3.1.1 整体接口

表 3-1 极化码译码算法整体接口

输入接口	C/D	位宽	来源	描述
K	C	us(10,0)	SW	信息比特长度，不含 CRC
E	C	us(11,0)	SW	速率匹配之后的码长
N	C	us(9,0)	SW	母码码长
L	C	us(3,0)	配置	译码路径列表宽度
dCrcType	C	us(2,0)	配置	分布式 CRC 类型指示 -1：不执行 DCRC； 0：CRC_CD；1：CRC_CK

输入接口	C/D	位宽	来源	描述
dCrcReg	D	us(3,0)	SW	DCRC 寄存器初始状态
deModLlr	D	S(6,2)	解调模块	输入的软值 LLR 信息
ILPattern	C	us(8,0)	SW	信息比特交织图样
jumpType	C	us(3,0)	SW	特殊节点类型指示
inforBitIdxCrc	C	us(8,0)	SW	分布式 CRC 校验比特校验的信息比特索引
输出接口	C/D	位宽	NA	描述
outputPM	D	s(6,0)		译码路径惩罚值
outputBits	D	us(1,0)		译码输出比特

3.2 预处理模块

3.2.1 极化码节点 stage 和 FG 函数类型配置

对于极化码每个节点进行 stage 标记和 f/g 函数类型标记，stage 用于表示当前译码节点的极化深度，f/g 函数标记用于选择更新节点 LLR 时选用的函数，1 表示该节点使用 g 函数更新 LLR，0 表示该节点使用 g 函数。

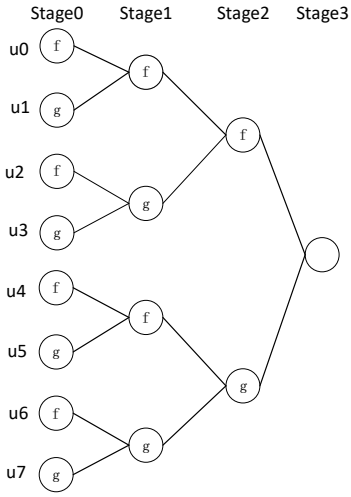


图 3-2 8 比特极化码节点计算类型

3.2.2 节点类型计算

在译码计算之前，需要根据极化码树状图确定每个节点的运算类型，即判断当前节点使用 f 函数还是 g 函数。另外，根据 3.3.3 小节的介绍，引入特殊节点以节约译码计算复杂度，所以需要在译码前确定译码节点的特殊类型。下图以码长 8 比特为例说明极化码节点类型计算。

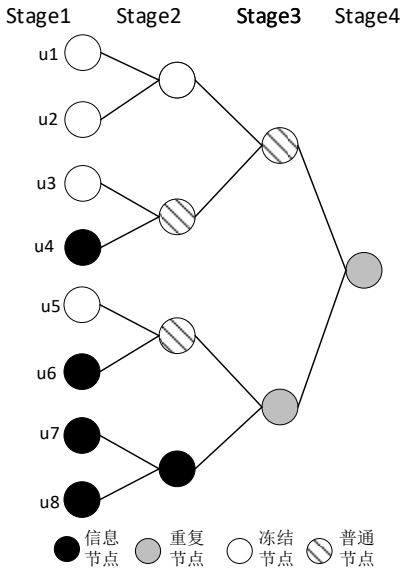


图 3-3 8 比特极化码特殊节点类型

3.2.3 节点译码深度计算

对于极化码 stage0 中的节点进行 depth 计算，作为更新u_s时回溯深度。需要注意的是，由于最后一个叶子节点的回溯深度理论为 4，不再需要进行u_s回溯计算，因此 depth 为 1。

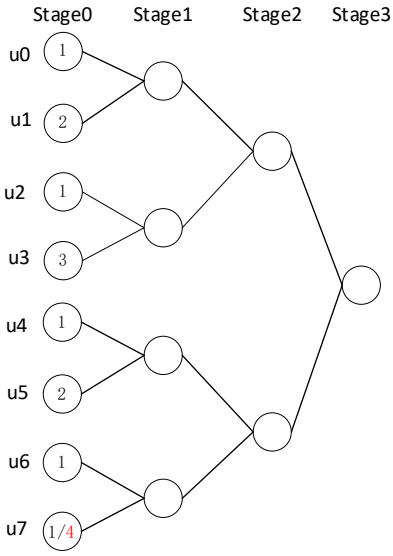


图 3-4 极化码 stage0 节点译码深度示意图

3.3 解扰模块

解扰模块位于整个译码的最前端，其输入为解调模块传输过来的软信息 LLR。译码器本地生成扰码序列，用于对 LLR 的解扰。

解扰过程可用如下公式表示：

$$deScrmLLR = LLR * \big(1 - 2c(i)\big), \quad i = 0 \cdots \tag{公式 3-1}$$

用于加扰的伪随机序列是一个 golden 序列，它由两个小 m 序列构成。

$$c(n) = \big(x_1(n + N_c) + x_2(n + N_c)\big) mod 2 \tag{公式 5-2}$$

$$x_1(n + 31) = \big(x_1(n + 3) + x_1(n)\big) mod 2 \tag{公式 5-3}$$

$$x_2(n + 31) = \big(x_2(n + 3) + x_2(n + 2) + x_2(n + 1) + x_2(n)\big) mod 2 \tag{公式 5-4}$$

其中，$\boldsymbol{N_c}$为加扰序列需要跳过的步数。小 m 序列移位寄存器的初始化可以用$\boldsymbol{C_{init}} = \sum_{i=0}^{30} \boldsymbol{x(i)} \cdot \boldsymbol{2^i}$表示。

3.4 LLR 计算模块

列表串行 SCL 译码中，各路径的 LLR 计算是串行流水进行的，单条路径的计算过程与 SC 译码一致。

设判决阶段的序号为 0，则信道计算阶段的序号为$log2(N)$，N为码长。其算法如伪代码所示。

LLR 计算算法

输入：比特计数值 i，路径序号 l

根据 i 计算起始阶段s_{start}（s_{start}为$i-1$二进制表示中第一个 1 的位置序号）：
for $s = s_{start}$ to 0 do
    读取路径 l 第 s+1 阶段的 LLR 指针$P_LLR_l^{s+1}$
    for $\varphi = 1$ to 2^s do
        根据指针$P_LLR_l^{s+1}$、节点序号φ读取 LLR 数LLR_{in}及部分和值$\hat{u}_{in}$
        依据公式 f 函数和 g 函数计算该节点的 LLR 值LLR_{out}，并进行存储
    end for
    更新路径 l 第 s 阶段的 LLR 指针$P_LLR_l^s$

9: end for

需要注意的是，使用多个 fg 计算单元，一个阶段内多个节点可并行计算。

在计算路径 l 的阶段 s 时需要路径 l 第 s+1 阶段 LLR 指针$P_LLR_l^{s+1}$，阶段 s 的所有节点计算完成后需要更新路径 l 的阶段 s 的 LLR 指针$P_LLR_l^s$。一条路径的 LLR 计算完成后，需要对路径处理进行使能。同时，路径序号加 1，后续进行下条路径的 LLR 计算。SCL 译码中列表大小为L（L为 2 的正整数幂），译码到第 i 个比特时，列表中有效路径数为L_a，当所有L_a有效路径 LLR 计算完成后，清除 LLR 计算使能，同时使能部分和的计算。

3.5 路径处理模块

路径处理是列表串行 SCL 译码的关键，主要分为两种情况：冻结比特处理和信息比特处理。设极化码的译码参数为(N, K, A)，其中N为码长，K 为信息比特的个数，A为信息比特索引集合。冻结比特的索引集合为A的补集，用符号A^C。

如果$i \in A^C$，则u_i是已知的。在列表串行结构译码器中，路径$P_l(1 \le l \le L_a - 1)$的 LLR 值$LLR_N^i[l]$计算完成后，将直接进行判决$\hat{u}_i[l] = u_i$，然后计算出对应的路径度量值PM_l^i。因为当前比特的判决值只能为u_i，所以无需进行路径的分化。原L_a条有效路径均为有效路径，无需进行路径选择，直接在原路径上更新判决比特和路径度量值即可。

如果$i \in A$,则分两种情况进行讨论：

1) $L_a < L$，即当前列表中的有效路径数小于列表大小的情况。在列表串行结构译码器中，路径P_l的$LLR_N^i[l]$计算完成后，立即分化为两条路径$P_{l,0}$和$P_{l,1}$。其中路径$P_{l,0}$第 i 个比特的判决值为 0，路径$P_{l,1}$第 i 个比特的判决值为 1。因为列表中还有空闲，$P_{l,0}$和$P_{l,1}$均会作为幸存路径保留下来。$P_{l,0}$将会被放在列表中第 2l 个位置，$P_{l,1}$将会被放在列表中第$2l + 1$的位置。两条路径的度量值按照公式 4-1 进行计算。列表中有效路径数L_a将变为 $2L_a$。

2) $L_a = L$，即列表为满时的情况。在列表串行结构译码器中，路径P_l的$LLR_N^i[l]$计算完成后，立即分化为两条路径$P_{l,0}$和$P_{l,1}$，计算出对应的路径度量值$PM_{l,0}^i$和$PM_{l,0}^i$存储下来。当路径$P_{L/2}$分化完成后，共有 L+2 条分化后的路径，此时便可以进行路径选择操作。从 L+2 条路径中根据路径度量值选择 2 条作为幸存路径，放置在列表中第 0 和第 1 个位置处。与此同时从原来 L+2 条路径中删除已经选择出去的路径，变为 L 条。当路径$P_{\frac{L}{2}+1}$扩展完成后，将新得到的路径$P_{\frac{L}{2}+1,0}$和$P_{\frac{L}{2}+1,1}$与上次选择剩余的 L 条路径合并，然后从这 L+2 个路径中依据路径度量值选择出 2 个最优的做为幸存路径，放置在列表中第 2 个和第 3 个位置处。 以此类推，选择出所有 L 条幸存路径。此时列表中有效路径数$L_a = L$。

在列表并行结构译码器中，L 条路径同时完成 LLR 计算，然后分化为 2L 条路径， 从中选出最优的 L 条作为幸存路径。而在列表串行结构中，L 条路径的 LLR 计算是依次进行的，因此路径分化的过程也是依次进行的。当累积的路径数为 L+2 条时，就从中选出 2 条作为幸存路径，直至完成所有路径的分化和选择。两者的操作是等价的。列表串行结构中的不需要等待 L 条路径全部计算完成，可以提前开启路径的选择操作，为后续计算做好准备。

存路径选择完毕后，需要对指针进行拷贝操作用于 lazy copy。同时将第 l 条路径中第 i 个比特的判决结果BD_l^i以及父路径的序号PP_l^i将存储下来，供后续译码结果的回溯使用。

3.6 部分和计算模块

列表串行 SCL 译码中，各路径的部分和计算同样是串行流水进行的，其过程与转移概率的计算方向相反，近似于极化码的编码过程。其算法如下所示

部分和计算算法

输入

1: 根据 i 计算终止阶段s_{end}（s_{end}为二进制表示中第一个 1 的位置序号）

2: for s=0 to s_{end} do

3: 读取路径 l 第 s 阶段的部分和指针$P_PS_l^s$;

4: for $\varphi = 1$ to 2^s do

5: 根据指针$P_{PS_l^s}$、节点序号φ读取部分和值$\hat{u}_{in}$;

6: 计算该节点下一阶段对应的部分和值$\hat{u}_{out}$，并进行存储;

7: end for

8: 更新该路径 l 第 s+1 阶段的部分和指针$P_PS_l^{s+1}$

9: end for

在路径 l 的阶段 s 计算时需要读取对应部分和指针$P_PS_l^s$，阶段 s 的所有节点计算完成后需要更新路径 l 阶段 s+1 的部分和指针$P_PS_l^{s+1}$。一条路径的部分和计算完成后，路径序号加 1，后续进行下条路径的部分和计算。当第一条路径的部分和计算完成后，使能 LLR 的计算。当所有L_a条有效路径部分和计算完成后，清除部分和计算使能。

3.7 Lazy copy 模块

码长为 N 的极化码在译码过程可以拆分为$n = \log_2(N)$个阶段，阶段$s(0 \le s \le n-1)$中节点的个数为2^s，所以对于每条路径而言需要存储的 LLR 和部分和的个数为 N-1。列表大小为 L 的 SCL 译码全部拷贝的复杂度为$O(LN)$，复杂度较高。

在译码过程中，阶段 s 被激活后，其内部所有节点都会被计算。采用共享内存的方式，为每个阶段设置一个指针。指针的内容为路径序号(逻辑编号)，指示实际数据存储的路径存储空间（物理编号）。由此可以将 N-1 个数据的拷贝变为$\log_2(N)$个指针的拷贝，复杂度降为$O(L\log_2 N)$。

指针的操作共用四种：初始化、拷贝、更新、读取。

1） 初始化：在开始译码时，路径$l(0 \le l \le L-1)$所有阶段的指针都会被设为 l;

2）拷贝：在路径选择完成后，设路径 l 的父路径编号为l_p，则将路径l_p所有阶段的指针

拷贝到路径 l 对应的位置；

3）更新：路径 l 在完成 LLR 更新阶段所有节点的计算后，因为新计算出的数据会被存储到路径 l 对应的空间中，所以需要将该阶段的指针更新为 l；

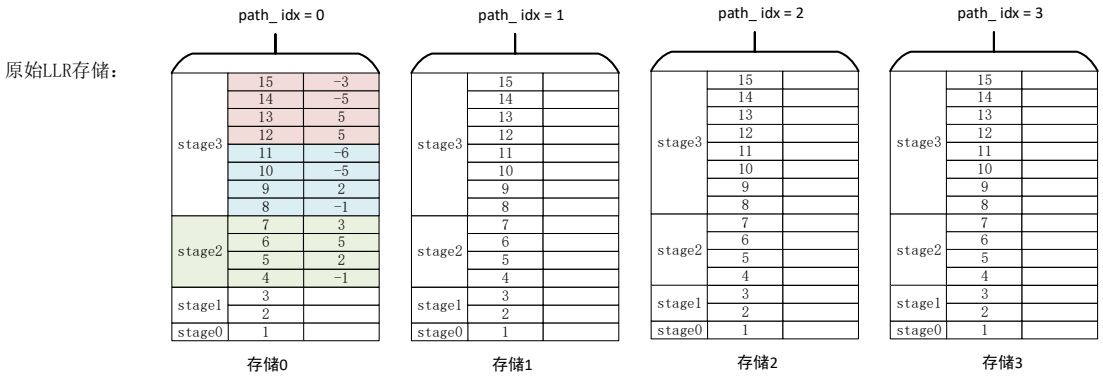
4）读取：根据路径序号 l 和阶段序号 s 进行索引得到对应指针。

由于 LLR 计算和部分和计算不是同步进行的，两者指针更新的时间不一致，所以需要两套指针。LLR 计算过程会读取 LLR 指针，在每个阶段计算完成后更新 LLR 指针。路径处理中，幸存路径选择完成后会进行 LLR 指针和部分和指针的拷贝操作。部分和计算过程中会读取部分和指针，在每个阶段计算完成后更新部分和指针。

L 条路径的选择过程不是同时进行的，所以路径的拷贝操作也需要顺序进行。为了防止拷贝操作覆盖有用数据，造成数据污染，需开辟两份存储空间进行乒乓操作。

3.7.1 Lazy Copy 示意

假设当前配置的 list 数目为 4，即分配 4 份存储用于存储过程中的 LLR，如下图所示：



假设计算到 stage2 的节点为"重复节点"，则存储两种假设（即假设信息比特为 0 和假设信息比特为 1）。

"假设信息比特为 0"的 path 会继承 path_idx=0 存储里的 LLR，下一个 node 会继续在 path_idx=0 所对应的存储中存储 LLR。

"假设信息比特为 1"的 path 会继承 path_idx=0 存储里的 LLR，下一个 node 会在

path_idx=1 所对应的存储中存储 LLR。

“常规 COPY”的做法是，“假设信息比特为 1”的 path 将 path_idx=0 中 LLR 拷贝到 path_idx=1 的存储中，继续后续的节点更新，如下图所示。读出和写入的延迟、功耗开销巨大。

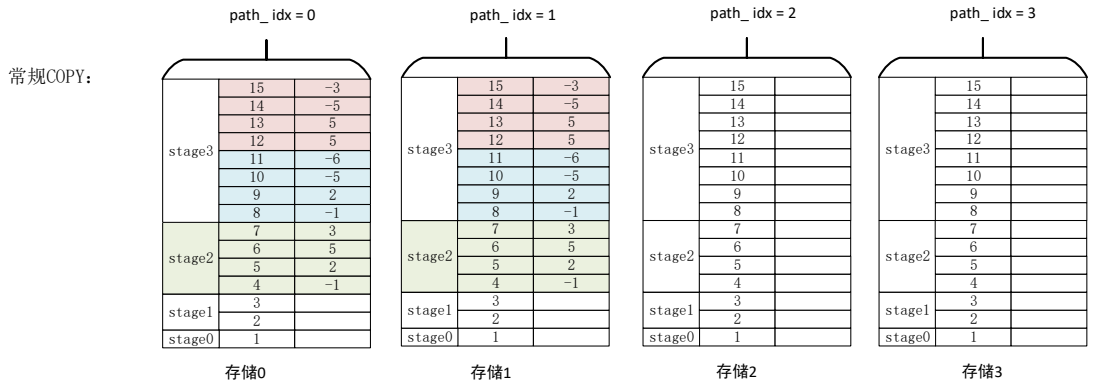


图 3-5 常规复制方法

“lazy copy”引入了 KEY（或者是指针），只进行 KEY 的拷贝，不进行实际内存中数据的拷贝，LLR 数据读取时，根据 KEY 索引到真实的存储地址，如下图所示，path_idx=0 的 KEY 存储中，stage3/2 的 LLR 指向存储 0，那么在提取数据时，stage3/2 的 LLR 会从“存储 0”中提取。

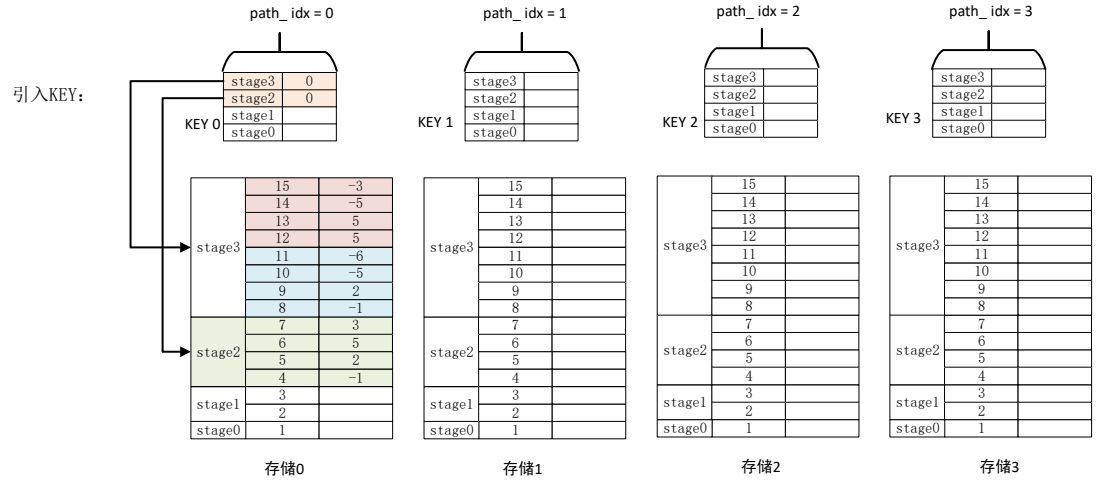


图 3-6 引入指针指示路径存储地址

对于上述“常规 COPY”，“lazy copy”的处理如下图所示，path_idx=1 的 KEY 存储中，stage3/2 的 LLR 指向存储 0，那么在提取数据时，stage3/2 的 LLR 会从“存储 0”中提取。

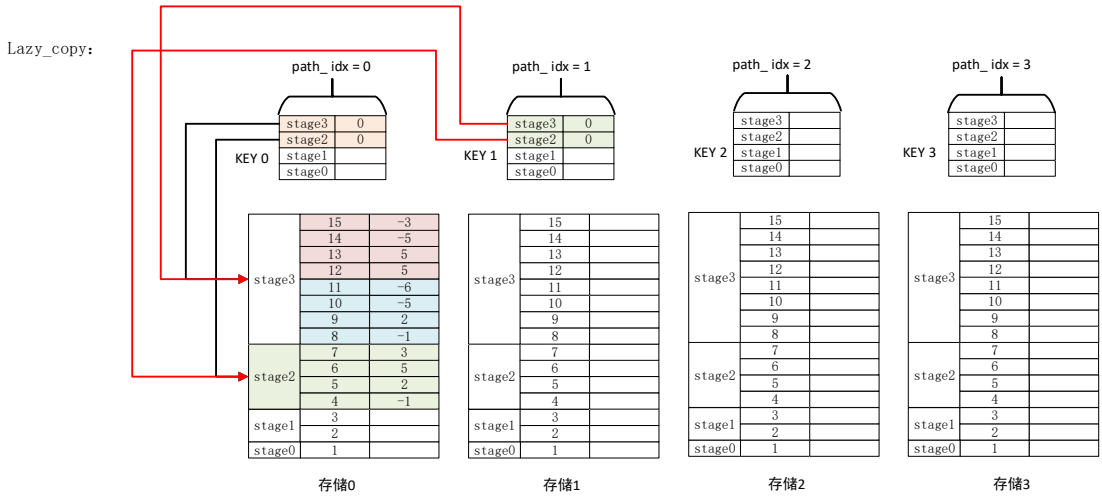


图 3-7Lazy copy 示意图

3.8 结果存储及回溯

对于一条路径而言，判决比特序列的长度为 N，L 条路径的拷贝需要一个大小为 L*N 的交换矩阵，会消耗大量的电路资源。为了避免这种大规模的拷贝，路径选择完成后，将 L 条路径的父路径序号和判决值均保存下来。待所有比特完成判决后，首先根据最终的路径度量值选择出最优路径，然后从后往前根据父路径序号依次回溯出译码结果序列。

译码结果回溯算法如下所示，其中$BD_{l_p}^{i}$表示第 i 个比特在路径l_p中的判决值，$PP_{l_p}^{i}$表示第 i 个比特的路径l_p对应的父路径：

结果回溯算法

输入：L 条路径的判决比特序列

初始化：l_p为$PM_l^N(0 \le l \le L-1)$中最小值对应的路径序号 l

1：for i=N to 1 do

2：　$\hat{u}_i = BD_{l_p}^{i}$

3：　$l_p = PP_{l_p}^{i}$

4：end for

输出：译码结果序列$\hat{u}_1^N$

回溯的计算复杂度为$O(N)$相较于 SCL 译码中其它计算的复杂度，可以忽略不计。存储单元消耗有少许增加，但与此同时，译码电路复杂度可以大幅度降低。

3.9 特殊节点处理

除 3.3.3 小节提到的两种特殊节点外，为降低时延还在 stage2 引入了新的特殊节点类型。由于 stage2 的节点包含 4 个叶子节点，因此按照叶子节点中信息比特的数目可分为 info2_0,info2_1,info3 和 info4 四类特殊节点。Info2_0 节点的叶子节点中第 1、3 比特为信息节点，info2_1 节点的叶子节点中第 2、3 比特为信息比特，info3 节点中第 1、2、3 比特为信息比特，info4 的所有 4 个叶子节点均为信息比特。每条译码路径的每个 stage2 特殊节点可分裂出最多 16 条译码路径，从中选择 4 条路径作为当前译码节点当前译码路径的幸存路径。

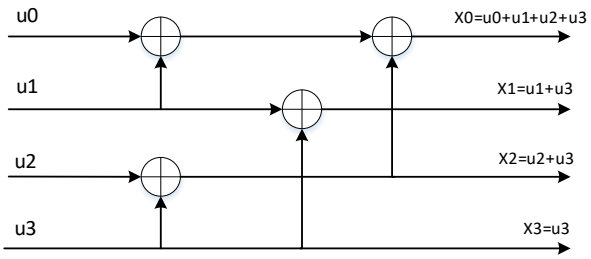


图 3-8 stag2 节点示意图

3.9.1 Info2_0/2_1 节点

如图 3-8 所示，info2_0 的第 1、3 比特为信息比特，则 x0==x1、x2==x3；而 info2_1 则应满足 x0==x2、x1==x3。因此 4 条幸存路径的选择可分为以下几种情况

1) 满足 x0==x1、x2==x3（info2_1:x0==x2、x1==x3）

$$PM0 = 0$$
$$PM1 = min(LLR0 + LLR1, LLR2 + LLR3)$$
$$PM2 = max(LLR0 + LLR1, LLR2 + LLR3)$$
$$PM3 = LLR0 + LLR1 + LLR2 + LLR3$$

2) x0==x1、x21!=x3/ x0!=x1、x2==x3（info2_1:x0==x2、x1!=x3）

$$PM0 = min(LLR2, LLR3)$$
$$PM1 = max(LLR2, LLR3)$$
$$PM2 = min(LLR2, LLR3) + LLR0 + LLR3$$
$$PM3 = max(LLR2, LLR3) + LLR0 + LLR3$$

3）x0!=x1、x21!=x3（info2_1:x0！=x2、x11!=x3）

$$PM0 = min(LLR2, LLR3) + min(LLR0, LLR1)$$
$$PM1 = max(LLR2, LLR3) + min(LLR0, LLR1)$$
$$PM2 = min(LLR2, LLR3) + max(LLR0, LLR1)$$
$$PM3 = max(LLR2, LLR3) + max(LLR0, LLR1)$$

3.9.2 info3 节点

对于 info3 节点，如果传输没有出错则有两种可能：x0==x2 且 x1==x3，x0==x2+1 且 x1==x3+1,因此 4 条幸存路径的选择可分为以下几种情况:

1） 满足上述条件

PM0=0

PM1-PM3 为（LL0+LLR1、LLR0+LLR2、LLR0+LLR3、LLR1+LLR2、LLR1+LLR3、LLR2+LLR3）6 个 PM 中 3 个最小的值;

2） 不满足上述条件则

PM0-PM3 为（LLR0、LLR1、LLR2、LLR3、LL0+LLR1、LLR0+LLR2、LLR0+LLR3、LLR1+LLR2、LLR1+LLR3、LLR2+LLR3、LL0+LLR1+LLR2、LL0+LLR1+LLR3、LL0+LLR2+LLR3、LL1+LLR2+LLR3）14 个 PM 中选择 4 个最小的 PM 值。

3.9.3 Info4 节点

对于 info4 节点，4 条幸存路径对应的 PM 值为：

PM0=0

PM1-3 为（LLR0、LLR1、LLR2、LLR3、LL0+LLR1+LLR2、LL0+LLR1+LLR3、LL0+LLR2+LLR3、LL1+LLR2+LLR3）8 个 PM 中选择 4 个最小的 PM 值。