CENG 242
Applications

Daler Rakhmatov

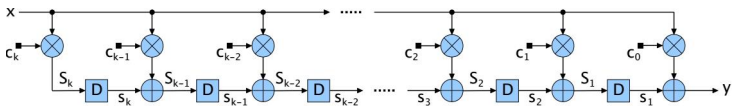
Electrical and Computer Engineering
University of Victoria
(daler@ece.uvic.ca)

Summer 2016

© 2016 Copyright Notice: All the materials pertaining to this document are provided solely for the use by a student registered in the University of Victoria course CENG 242 for educational purposes and private study. Any other use may be an infringement of copyright if done without securing the permission of the copyright holder.

Linear Discrete-Time Systems

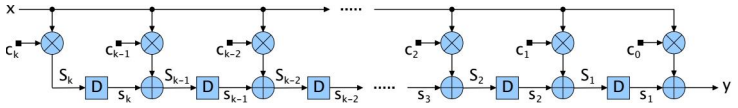
Linear feedforward shift register I



A k-stage **linear feedforward shift register** over $GF(2)$ is a structure shown above, where

- $c_0, c_1, ..., c_k \in GF(2)$ are *constants*;
- $x : \mathbb{Z}_+ \mapsto GF(2)$ is a function representing a *discrete-time input*;
- $y : \mathbb{Z}_+ \mapsto GF(2)$ is a function representing a *discrete-time output*;
- $\oplus$ and $\otimes$ represent "addition" and "multiplication" in $GF(2)$;
- D represents a **delay operator**: given a function $f : \mathbb{Z}_+ \mapsto GF(2)$ and some constant $\tau \in \mathbb{Z}_+$, we let $(D^\tau f)(t) = f(t-\tau)$ for every $t \geq \tau$, while setting $f(t-\tau) = 0$ whenever $t < \tau$.

Linear feedforward shift register II



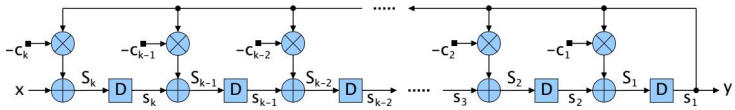
For every $t \geq k$, the output of a k-stage linear feedforward shift register is given by the following equation:

$$\boxed{y(t) = \sum_{j=0}^{k} c_j \cdot x(t-j) = (c_0 + c_1 \cdot D + c_2 \cdot D^2 + ... + c_k \cdot D^k)x(t)},$$

where the polynomial $c_0 + c_1 \cdot D + c_2 \cdot D^2 + ... + c_k \cdot D^k = T(D)$ is called a **transfer function** of our k-stage linear feedforward shift register.

A *logic circuit* of a linear feedforward shift register consists of XOR gates performing "+", AND gates performing "$\cdot$", and *clocked flip-flops* acting as delay operators.

Linear feedback shift registers



A k-stage **linear feedback shift register** over $GF(2)$ is a structure shown above, where the following equations hold for every $t \geq k$ ($t \in \mathbb{Z}_+$):

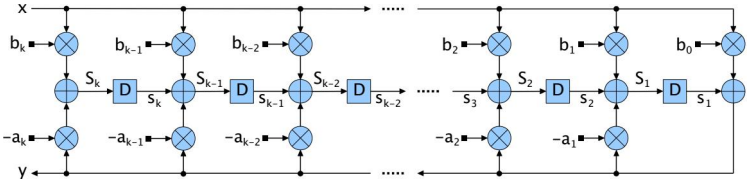
$$S_k(t) = x(t) - c_k \cdot y(t), \quad S_{k-1}(t) = s_k(t) - c_{k-1} \cdot y(t),$$
$$..., \quad S_1(t) = s_2(t) - c_1 \cdot y(t), \quad y(t) = s_1(t).$$

Since $s_j(t) = S_j(t-1) = DS_j(t)$ for every $j \in \{1, 2, ..., k\}$, we obtain:

$$y(t) = s_1(t) = D(s_2(t) - c_1 \cdot y(t)) = (-c_1 \cdot D)y(t) + D^2(s_3(t) - c_2 \cdot y(t)) =$$
$$... = (-c_1 \cdot D - c_2 \cdot D^2 - ... - c_k \cdot D^k)y(t) + D^k x(t), \text{ or}$$
$$\boxed{(1 + c_1 \cdot D + c_2 \cdot D^2 + ... + c_k \cdot D^k)y(t) = D^k x(t)}.$$

Note that $c_0 = 1$ and $-c_j = c_j$ for every $j \in \{1, 2, ..., k\}$ in the field $GF(2)$.

Rational transfer function



Putting our feedforward and feedback shift registers together, we obtain a k-stage **linear single-input single-output (SISO) machine** that satisfies the following equation for every $t \geq k$ ($t \in \mathbb{Z}_+$):

$$(1 + a_1 \cdot D + a_2 \cdot D^2 + ... + a_k \cdot D^k)y(t) =$$
$$(b_0 + b_1 \cdot D + b_2 \cdot D^2 + ... + b_k \cdot D^k)x(t),$$

which corresponds to the following **rational transfer function**:

$$\boxed{T(D) = \frac{b_0 + b_1 \cdot D + b_2 \cdot D^2 + ... + b_k \cdot D^k}{1 + a_1 \cdot D + a_2 \cdot D^2 + ... + a_k \cdot D^k} = \frac{b(D)}{a(D)}}.$$

GCD of two polynomials

Let $T(D) = \dfrac{b(D)}{a(D)}$ be some rational transfer function.

We can easily construct a linear SISO machine that implements our $T(D)$, but first we need to eliminate any common factors of $a(D)$ and $b(D)$.

In other words, we are to determine and eliminate (by polynomial division) the **GCD** of $a(D)$ and $b(D)$, which is the **monic polynomial** $d(D)$ of the highest degree such that $d(D)$ divides both $a(D)$ and $b(D)$.

We can determine such $d(D)$ using **Euclid's algorithm** for polynomials. It can be applied to any pair of non-zero polynomials over any field.

Euclid's algorithm for polynomials

1. Let $r_0(D) \leftarrow b(D)$, and let $k \leftarrow \max\{\deg a(D), \deg b(D)\}$.
2. Divide $a(D)$ by $r_0(D)$, which produces quotient $q_1(D)$ and remainder $r_1(D)$ such that $a(D) = r_0(D) \cdot q_1(D) + r_1(D)$.
3. For each $j = 1, 2, ..., k$ do:
   - If $r_j(D) = 0$, stop and output $d(D) \leftarrow c \cdot r_{j-1}(D)$, where our constant c is such that $d(D)$ is **monic**;
   - Else, divide $r_{j-1}(D)$ by $r_j(D)$, which produces quotient $q_{j+1}(D)$ and remainder $r_{j+1}(D)$ such that $r_{j-1}(D) = r_j(D) \cdot q_{j+1}(D) + r_{j+1}(D)$.

Example of polynomial division over $\mathbb{Q}$

Consider the following two polynomials in $\mathbb{Q}[D]$:

- $a(D) = 2 \cdot D^4 + 5 \cdot D^3 + 6 \cdot D^2 + 4 \cdot D + 1$,
- $b(D) = 2 \cdot D^2 + 5 \cdot D + 2$.

After dividing $a(D)$ by $b(D)$, we obtain $a(D) = b(D) \cdot q(D) + r(D)$, where $q(D) = D^2 + 2$ and $r(D) = -6 \cdot D - 3$.

$$
\begin{array}{r|l}
2D^2 + 5D + 2 & D^2 + 2 \\
\hline
2D^4 + 5D^3 + 6D^2 + 4D + 1 & \\
-2D^4 - 5D^3 - 2D^2 & \\
\hline
4D^2 + 4D + 1 & \\
-4D^2 - 10D - 4 & \\
\hline
-6D - 3 &
\end{array}
$$

Example: GCD of two polynomials over $\mathbb{Q}$

Consider the following two polynomials in $\mathbb{Q}[D]$:

- $a(D) = 2 \cdot D^4 + 5 \cdot D^3 + 6 \cdot D^2 + 4 \cdot D + 1$,
- $b(D) = 2 \cdot D^2 + 5 \cdot D + 2$.

The GCD of $a(D)$ and $b(D)$ is $d(D) = -\frac{1}{6} \cdot (-6 \cdot D - 3) = D + \frac{1}{2}$. Note that we have chosen the constant $c = -\frac{1}{6}$, which makes $d(D)$ monic.

$$2D^4 + 5D^3 + 6D^2 + 4D + 1 = \left(2D^2 + 5D + 2\right) \cdot \left(D^2 + 2\right) + \left(-6D - 3\right)$$
$$2D^2 + 5D + 2 = \left(-6D - 3\right) \cdot \left(-\tfrac{1}{3}D - \tfrac{2}{3}\right) + 0$$

Example of simplifying $\frac{b(D)}{a(D)}$ over $\mathbb{Q}$

Consider $\dfrac{b(D)}{a(D)} = \dfrac{2 \cdot D^2 + 5 \cdot D + 2}{2 \cdot D^4 + 5 \cdot D^3 + 6 \cdot D^2 + 4 \cdot D + 1}$ over $\mathbb{Q}$.

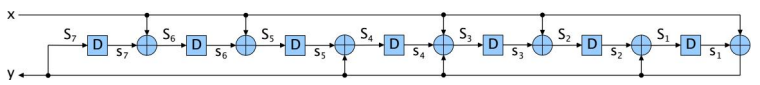
To simplify it, we divide both $a(D)$ and $b(D)$ by their GCD $d(D) = D + \frac{1}{2}$:

$$\frac{b(D)}{a(D)} = \frac{b(D)/d(D)}{a(D)/d(D)} = \frac{2 \cdot D + 4}{2 \cdot D^3 + 4 \cdot D^2 + 4 \cdot D + 2}.$$

$$
\begin{array}{r|l}
D + \frac{1}{2} & 2D^3 + 4D^2 + 4D + 2 \\
\hline
2D^4 + 5D^3 + 6D^2 + 4D + 1 & \\
-2D^4 - D^3 & \\
\hline
4D^3 + 6D^2 & \\
-4D^3 - 2D^2 & \\
\hline
4D^2 + 4D & \\
-4D^2 - 2D & \\
\hline
2D + 1 & \\
-2D - 1 & \\
\hline
0 &
\end{array}
\qquad
\begin{array}{r|l}
D + \frac{1}{2} & 2D + 4 \\
\hline
2D^2 + 5D + 2 & \\
-2D^2 - D & \\
\hline
4D + 2 & \\
-4D - 2 & \\
\hline
0 &
\end{array}
$$

Example: SISO machine over $GF(2)$

Consider the transfer function $T(D) = \dfrac{b(D)}{a(D)} = \dfrac{1 + D^2 + D^3 + D^5 + D^6}{1 + D + D^3 + D^4 + D^7}$, where $b(D)$ and $a(D)$ have no common factors. We need $k = 7$ flip-flops. As $a_2 = a_5 = a_6 = 0$ and $b_1 = b_4 = b_7 = 0$, we remove the corresponding AND gates. The remaining nine AND gates are replaced with wires, since $1 \cdot x = x$ and $-1 \cdot y = y$ in $GF(2)$. Note that the leftmost XOR gate has been replaced with a wire as well, since $S_7 = b_7 \cdot x - a_7 \cdot y = y$ in $GF(2)$. The figure below shows the resulting linear SISO machine.



Important: We can construct linear SISO machines over fields other than $GF(2)$. We need to properly negate our feedback coefficients $a_1, a_2, ..., a_k$, use an appropriate delay element for D, and replace the XOR/AND gates with their respective analogues implementing "addition"/"multiplication" in the chosen field.

Linear multiple-input multiple-output (MIMO) machines

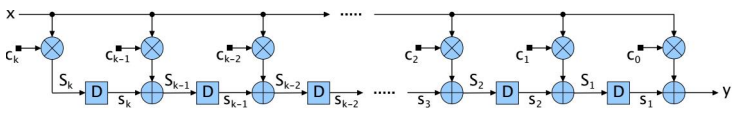
A k-stage **linear multiple-input multiple-output (MIMO) machine** over a field F satisfies the following pair of equations for every $t \in Z_+$:

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

- $\boldsymbol{s}(t) = [s_j \in F]_{k\times 1} = [\,s_1(t)\ \ s_2(t)\ \cdots\ s_k(t)\,]^\top$ is the **state vector**,
- $\boldsymbol{x}(t) = [x_j \in F]_{m\times 1} = [\,x_1(t)\ \ x_2(t)\ \cdots\ x_m(t)\,]^\top$ is the **input vector**,
- $\boldsymbol{y}(t) = [y_j \in F]_{n\times 1} = [\,y_1(t)\ \ y_2(t)\ \cdots\ y_n(t)\,]^\top$ is the **output vector**,

- $\boldsymbol{A} = [a_{ij} \in F]_{k\times k}$ is the **state (evolution) matrix**,
- $\boldsymbol{B} = [b_{ij} \in F]_{k\times m}$ is the **input (control) matrix**,
- $\boldsymbol{C} = [c_{ij} \in F]_{n\times k}$ is the **output (observation) matrix**,
- $\boldsymbol{D} = [d_{ij} \in F]_{n\times m}$ is the **transmission (feedthrough) matrix.**

Example: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ for linear feedforward shift register

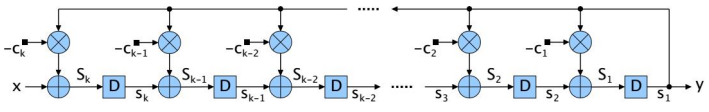


$$\boldsymbol{x}(t) = \begin{bmatrix} x(t) \end{bmatrix}, \qquad \boldsymbol{y}(t) = \begin{bmatrix} y(t) \end{bmatrix},$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{k-2} \\ c_{k-1} \\ c_k \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{D} = \begin{bmatrix} c_0 \end{bmatrix}.$$

Example: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ for linear feedback shift register



$$\boldsymbol{x}(t) = \begin{bmatrix} x(t) \end{bmatrix}, \qquad \boldsymbol{y}(t) = \begin{bmatrix} y(t) \end{bmatrix},$$

$$\boldsymbol{A} = \begin{bmatrix} -c_1 & 1 & 0 & \cdots & 0 & 0 \\ -c_2 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -c_{k-2} & 0 & 0 & \cdots & 1 & 0 \\ -c_{k-1} & 0 & 0 & \cdots & 0 & 1 \\ -c_k & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix}.$$

Linear MIMO machine equations

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

Note that the equation for $\boldsymbol{s}(t)$ is the **first-order NHLR**, and it requires one initial condition, namely the *initial state* $\boldsymbol{s}(0)$**.** Thus, a linear MIMO machine is completely specified by $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, and $\boldsymbol{s}(0)$**.**

The machine's state and output at time $t > 0$ can be computed as follows:

$$\begin{aligned} \boldsymbol{s}(t) &= \boldsymbol{A}^t\boldsymbol{s}(0) + \textstyle\sum_{r=0}^{t-1} \boldsymbol{A}^{t-1-r}\boldsymbol{B}\boldsymbol{x}(r), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{A}^t\boldsymbol{s}(0) + \textstyle\sum_{r=0}^{t-1} \boldsymbol{C}\boldsymbol{A}^{t-1-r}\boldsymbol{B}\boldsymbol{x}(r) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

How do we calculate $\boldsymbol{A}^t$ efficiently?

Calculating $\boldsymbol{A}^t$

Problem:
Given a matrix $\boldsymbol{A} = [a_{ij}]_{k\times k}$ and some $t \in \mathbb{N}$, calculate $\boldsymbol{A}^t$, letting $\boldsymbol{A}^0 = \boldsymbol{I}$ where $\boldsymbol{I}$ denotes the **identity matrix.**

Solution:
Use the *characteristic polynomial* of **A** and the *Cayley-Hamilton theorem.*

The **characteristic equation** of $\boldsymbol{A}$ is defined as $\det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = 0$, where $\det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = \lambda^k + h_{k-1} \cdot \lambda^{k-1} + ... + h_1 \cdot \lambda + h_0 = g(\lambda)$**.** We call $g(\lambda)$ the **characteristic polynomial** of $\boldsymbol{A}$, whose coefficients $h_0, h_1, ..., h_{k-1}$ are obtained as a result of computing $\det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I})$**.**

It can be shown that the **roots** of the characteristic equation $g(\lambda) = 0$ are the **eigenvalues** of $\boldsymbol{A}$**.**

Cayley-Hamilton theorem

Theorem (Cayley-Hamilton)

Every $k \times k$ matrix $\boldsymbol{A}$ satisfies its own characteristic equation, i.e.,

$$\boldsymbol{A}^k + h_{k-1} \cdot \boldsymbol{A}^{k-1} + ... + h_1 \cdot \boldsymbol{A} + h_0 \cdot \boldsymbol{I} = \boldsymbol{0},$$

where $\boldsymbol{0} = [0]_{k\times k}$, and $h_0, h_1, ..., h_{k-1}$ are coefficients of the characteristic polynomial $g(\lambda) = \det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = \lambda^k + h_{k-1} \cdot \lambda^{k-1} + ... + h_1 \cdot \lambda + h_0$.

Consequently, if $\lambda_1, \lambda_2, ..., \lambda_k$ are eigenvalues of $\boldsymbol{A}$, then:

$$\textstyle\prod_{j=1}^{k} (\boldsymbol{A} - \lambda_j \cdot \boldsymbol{I}) = \boldsymbol{0}.$$

Important: The Cayley-Hamilton theorem tells us that every $k \times k$ matrix $\boldsymbol{A}^t$ can be represented as a **linear combination** of $\boldsymbol{A}^{k-1}$, $\boldsymbol{A}^{k-2}$, ..., $\boldsymbol{A}^2$, $\boldsymbol{A}$, and $\boldsymbol{I}$, where $t, k \in \mathbb{N}$ and $t \geq k$**.**

Putzer's algorithm

Putzer's algorithm calculates $\boldsymbol{A}^t$ at $t \geq k$. It applies the Cayley-Hamilton theorem to k eigenvalues of $\boldsymbol{A}$ (denoted by $\lambda_1, \lambda_2, ..., \lambda_k$) as follows:

1. Let $\boldsymbol{M}_0 \leftarrow \boldsymbol{I}$, and let $\boldsymbol{M}_1 \leftarrow \boldsymbol{A} - \lambda_1 \cdot \boldsymbol{I}$.
2. For each index $j = 2, 3, .., k-1$, calculate $\boldsymbol{M}_j = (\boldsymbol{A} - \lambda_j \cdot \boldsymbol{I}) \cdot \boldsymbol{M}_{j-1}$.
   - **Note:** For $j = k$, we have $\boldsymbol{M}_k = \boldsymbol{0}$, where $\boldsymbol{0} = [0]_{k \times k}$.
3. Given $t \geq k$, let $\mu_1(t) \leftarrow \lambda_1^t$, and let $\mu_j(t) \leftarrow \sum_{r=0}^{t-1} \lambda_j^{t-1-r} \cdot \mu_{j-1}(r)$ for each index $j = 2, 3, ..., k$.
   - **Note:** $\mu_j(t)$ satisfies the recursion $\mu_j(t+1) - \lambda_j \cdot \mu_j(t) = \mu_{j-1}(t)$.
4. Let $\boldsymbol{A}^t \leftarrow \sum_{j=1}^{k} \mu_j(t) \cdot \boldsymbol{M}_{j-1}$.

Example: applying Putzer's algorithm I

Let a linear MIMO machine over $\mathbb{Q}$ have the following 2×2 state matrix:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ -2 & 4 \end{bmatrix} \longrightarrow \boldsymbol{A} - \lambda \cdot \boldsymbol{I} = \begin{bmatrix} 1-\lambda & 1 \\ -2 & 4-\lambda \end{bmatrix}.$$

Its characteristic polynomial is $g(\lambda) = \det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = (\lambda - 3) \cdot (\lambda - 2)$. The equation $g(\lambda) = 0$ has two roots $\lambda_1 = 3$ and $\lambda_2 = 2$. Therefore,

$$\begin{aligned} \mu_1(t) &= \lambda_1^t = 3^t, \\ \mu_2(t) &= \textstyle\sum_{r=0}^{t-1} \lambda_2^{t-1-r} \cdot \mu_1(r) = 2^{t-1} \cdot \sum_{r=0}^{t-1} \left(\frac{3}{2}\right)^r \\ &= 2^{t-1} \cdot \frac{1 - \left(\frac{3}{2}\right)^t}{1 - \frac{3}{2}} = 3^t - 2^t. \end{aligned}$$

Useful formula:

$$\textstyle\sum_{r=0}^{t-1} c^r = 1 + c + c^2 + ... + c^{t-1} = \begin{cases} t, & c = 1; \\ \frac{1-c^t}{1-c}, & c \neq 1. \end{cases}$$

Example: applying Putzer's algorithm II

We also have:

$$\boldsymbol{M}_1 = \boldsymbol{A} - \lambda_1 \cdot \boldsymbol{I} = \begin{bmatrix} 1 & 1 \\ -2 & 4 \end{bmatrix} - \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ -2 & 1 \end{bmatrix}.$$

Consequently, we obtain the following closed-form expression for $\boldsymbol{A}^t$:

$$\begin{aligned} \boldsymbol{A}^t &= \mu_1(t) \cdot \boldsymbol{I} + \mu_2(t) \cdot \boldsymbol{M}_1 \\ &= 3^t \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + (3^t - 2^t) \cdot \begin{bmatrix} -2 & 1 \\ -2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 2^{t+1} - 3^t & 3^t - 2^t \\ 2^{t+1} - 2 \cdot 3^t & 2 \cdot 3^t - 2^t \end{bmatrix}. \end{aligned}$$

Another example: applying Putzer's algorithm I

Let a linear MIMO machine over $\mathbb{Q}$ have the following 3×3 state matrix:

$$\boldsymbol{A} = \begin{bmatrix} 4 & 1 & 2 \\ 0 & 2 & -4 \\ 0 & 1 & 6 \end{bmatrix} \longrightarrow \boldsymbol{A} - \lambda \cdot \boldsymbol{I} = \begin{bmatrix} 4-\lambda & 1 & 2 \\ 0 & 2-\lambda & -4 \\ 0 & 1 & 6-\lambda \end{bmatrix}.$$

Its characteristic polynomial is $g(\lambda) = \det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = (4 - \lambda) \cdot (\lambda - 4)^2$. The equation $g(\lambda) = 0$ has three roots $\lambda_1 = \lambda_2 = \lambda_3 = 4$. Therefore,

$$\begin{aligned} \mu_1(t) &= \lambda_1^t = 4^t, \\ \mu_2(t) &= \textstyle\sum_{r=0}^{t-1} \lambda_2^{t-1-r} \cdot \mu_1(r) = \sum_{r=0}^{t-1} 4^{t-1-r} \cdot 4^r = t \cdot 4^{t-1}, \\ \mu_3(t) &= \textstyle\sum_{r=0}^{t-1} \lambda_3^{t-1-r} \cdot \mu_2(r) = \sum_{r=0}^{t-1} 4^{t-1-r} \cdot (r \cdot 4^{r-1}) = \frac{t^2-t}{2} \cdot 4^{t-2}. \end{aligned}$$

Another example: applying Putzer's algorithm II

We also have:

$$\boldsymbol{M}_1 = \boldsymbol{A} - \lambda_1 \cdot \boldsymbol{I} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & -2 & -4 \\ 0 & 1 & 2 \end{bmatrix}, \ \boldsymbol{M}_2 = (\boldsymbol{A} - \lambda_2 \cdot \boldsymbol{I}) \cdot \boldsymbol{M}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Consequently, we obtain the following closed-form expression for $\boldsymbol{A}^t$:

$$\begin{aligned} \boldsymbol{A}^t &= \mu_1(t) \cdot \boldsymbol{I} + \mu_2(t) \cdot \boldsymbol{M}_1 + \mu_3(t) \cdot \boldsymbol{M}_2 \\ &= 4^t \cdot \boldsymbol{I} + t \cdot 4^{t-1} \cdot \boldsymbol{M}_1 + \tfrac{t^2-t}{2} \cdot 4^{t-2} \cdot \boldsymbol{M}_2 \\ &= \begin{bmatrix} 4^t & t \cdot 4^{t-1} & 2 \cdot t \cdot 4^{t-1} \\ 0 & 4^t - 2 \cdot t \cdot 4^{t-1} & -t \cdot 4^{t-1} \\ 0 & t \cdot 4^{t-1} & 4^t + 2 \cdot t \cdot 4^{t-1} \end{bmatrix}. \end{aligned}$$

Discrete-time linear systems I

A **kth-order discrete-time linear system**, with a single input $x(t)$ and a single output $y(t)$, satisfies the following **difference equation** over $\mathbb{C}$, for every integer $t \geq 0$:

$$\sum_{j=0}^{k} a_j \cdot y(t-j) = \sum_{j=0}^{k} b_j \cdot x(t-j), \text{ where } a_0 = 1, a_k \neq 0.$$

Important: We assume that $y(t) = x(t) = 0$ for every integer $t < 0$.

The above difference equation can be rewritten as follows:

$$(1 + a_1 \cdot D + ... + a_k \cdot D^k) y(t) = (b_0 + b_1 \cdot D + ... + b_k \cdot D^k) x(t),$$

where D is the familiar **delay operator.**[1]

[1] Given a function $f : \mathbb{Z}_+ \mapsto GF(2)$ and some constant $\tau \in \mathbb{Z}_+$, $(D^\tau f)(t) = f(t - \tau)$ for every $t \geq \tau$, with $f(t - \tau) = 0$ whenever $t < \tau$.

Discrete-time linear systems II

Let

- $a(D) = 1 + a_1 \cdot D + a_2 \cdot D^2 + ... + a_k \cdot D^k$,
- $b(D) = b_0 + b_1 \cdot D + b_2 \cdot D^2 + ... + b_k \cdot D^k$.

Then, the output of our kth-order discrete-time linear system is given by

$$y(t) = T(D)x(t), \text{ where } T(D) = \frac{b(D)}{a(D)}.$$

Note that $T(D) = \frac{b(D)}{a(D)} = \frac{b_0 + b_1 \cdot D + b_2 \cdot D^2 + ... + b_k \cdot D^k}{1 + a_1 \cdot D + a_2 \cdot D^2 + ... + a_k \cdot D^k}$, i.e., it is the familiar **rational transfer function** of a linear SISO machine.

We already know how to implement a SISO machine over the field $GF(2)$, given a rational transfer function $T(D)$. Since we are working in $\mathbb{C}$ rather than $GF(2)$, we replace the XOR/AND gates with *adders*/*multipliers* that perform additions and multiplications in $\mathbb{C}$.

Discrete-time linear systems as NHLRs

Recall: NHLR

Let F be a field. Given k initial values $c_0, c_1, ..., c_{k-1} \in F$ and coefficients $a_1, a_2, ..., a_k \in F$ with $a_k \neq 0$, we define a **kth-order nonhomogeneous linear recursion (NHLR)** as follows:

$$\begin{cases} u_0 = c_0, \quad u_1 = c_1, \quad ..., \quad u_{k-1} = c_{k-1} & \text{(initial conditions);} \\ u_{n+k} + a_1 \cdot u_{n+k-1} + ... + a_k \cdot u_n = g(n) & \text{(for every } n \in \mathbb{Z}_+\text{),} \end{cases}$$

where function $g : \mathbb{Z}_+ \mapsto F$ maps every $n \in \mathbb{Z}_+$ to some unique $g(n) \in F$.

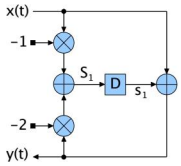
Essentially, any given kth-order discrete linear system can be expressed as the corresponding kth-order NHLR over $\mathbb{C}$:

$$\begin{cases} u_0 = y(0), \quad u_1 = y(1), \quad ..., \quad u_{k-1} = y(k-1) & \text{(initial conditions);} \\ u_{t+k} + a_1 \cdot u_{t+k-1} + ... + a_k \cdot u_t = g(t) & \text{(for every } t \in \mathbb{Z}_+\text{),} \end{cases}$$

where $u_t = y(t)$ for every $t \in \mathbb{Z}_+$, and $g(t) = \sum_{j=0}^{k} b_j \cdot x(t+k-j)$.

Example: first-order discrete-time linear system

Consider a discrete-time linear system given by the equation $y(t) + 2 \cdot y(t-1) = x(t) - x(t-1)$ for every integer $t \geq 0$, where $x(t) = t^2$, letting $y(t) = x(t) = 0$ for $t < 0$. We have $a_1 = 2$, $b_0 = 1$, $b_1 = -1$, and $T(D) = \frac{1-D}{1+2 \cdot D}$. The structure of the corresponding linear SISO machine is as shown here. (The complex multiplier for $b_0 = 1$ is optimized away.)



We rewrite $y(t) + 2 \cdot y(t-1) = x(t) - x(t-1)$ as the following NHLR:

$$\begin{cases} u_0 = y(0) & \text{(initial conditions);} \\ u_{t+1} + 2 \cdot u_t = g(t) & \text{(for every } t \in \mathbb{Z}_+\text{),} \end{cases}$$

where $u_t = y(t)$ for every $t \in \mathbb{Z}_+$, and $g(t) = x(t+1) - x(t) = 2 \cdot t + 1$. We know how to solve this NHLR.[2] After combining its homogeneous and particular solutions, we obtain $u_t = y(t) = -\frac{1}{9} \cdot (-2)^t + \frac{2}{3} \cdot t + \frac{1}{9}$.

[2] To determine the initial condition $u_o = y(0)$, we use our original difference equation. Letting $t = 0$ yields $y(0) + 2 \cdot y(-1) = x(0) - x(-1)$. Thus, $y(0) = x(0) = 0^2 = 0$.

Unilateral $\mathcal{Z}$-transform

Let $f : \mathbb{Z}_+ \mapsto \mathbb{C}$ be a discrete-time function from $\mathbb{Z}_+$ (time) to $\mathbb{C}$ (signal), and set $f(t) = 0$ for every integer $t < 0$. The **unilateral $\mathcal{Z}$-transform** of f, denoted by $\mathcal{Z}[f]$ or $F(z^{-1})$, is defined as follows:

$$\boxed{F(z^{-1}) = \mathcal{Z}[f] = \sum_{t=0}^{\infty} f(t) \cdot z^{-t}}.$$

If we interpret $f(t)$ as an infinite sequence of coefficients from the field $\mathbb{C}$, then we can view $F(z^{-1})$ as an OGF of $f(t)$, i.e., it is a **power series** in the ring $\mathbb{C}[[z^{-1}]]$.

Example: Let $t \in \mathbb{Z}_+$, $c \in \mathbb{C}$, and $\delta(t) = \begin{cases} 1, & t = 0; \\ 0, & t \neq 0. \end{cases}$

- $f(t) = \delta(t) \longleftrightarrow F(z^{-1}) = \mathcal{Z}[f] = \sum_{t=0}^{\infty} \delta(t) \cdot z^{-t} = 1$;
- $f(t) = c^t \longleftrightarrow F(z^{-1}) = \mathcal{Z}[f] = \sum_{t=0}^{\infty} c^t \cdot z^{-t} = \frac{1}{1 - c \cdot z^{-1}}$;
- $f(t) = tc^t \longleftrightarrow F(z^{-1}) = \mathcal{Z}[f] = \sum_{t=0}^{\infty} tc^t \cdot z^{-t} = \frac{c \cdot z^{-1}}{1 - c \cdot z^{-1}}$.

$\mathcal{Z}$-transform as OGF I

Expression for u_n	OGF $U(x) = \sum_{n=0}^{\infty} u_n \cdot x^n$
$u_n = 1$	$\frac{1}{1-x}$
$u_n = c^n$	$\frac{1}{1-c \cdot x}$
$u_n = nc$	$\frac{c \cdot x}{(1-x)^2}$
$u_n = nc^n$	$\frac{c \cdot x}{(1-c \cdot x)^2}$
$u_n = \binom{m+n-1}{n} c^n$	$\frac{1}{(1-c \cdot x)^m}$

Expression for $f(t)$	$\mathcal{Z}$-transform $F(z^{-1}) = \mathcal{Z}[f] = \sum_{t=0}^{\infty} f(t) \cdot z^{-t}$
$f(t) = 1$	$\frac{1}{1-z^{-1}}$
$f(t) = c^t$	$\frac{1}{1-c \cdot z^{-1}}$
$f(t) = tc$	$\frac{c \cdot z^{-1}}{(1-z^{-1})^2}$
$f(t) = tc^t$	$\frac{c \cdot z^{-1}}{(1-c \cdot z^{-1})^2}$
$f(t) = \binom{m+t-1}{t} c^t$	$\frac{1}{(1-c \cdot z^{-1})^m}$

$\mathcal{Z}$-transform as OGF II

Modified OGF	Resulting Sequence
$\alpha \cdot U(x) + \beta \cdot V(x)$	$..., \alpha \cdot u_n + \beta \cdot v_n, ...$
$U(x) \cdot V(x)$	$..., \sum_{k=0}^{n} u_k \cdot v_{n-k}, ...$
$\frac{1}{1-x} \cdot U(x)$	$..., \sum_{k=0}^{n} u_k, ...$
$x^m \cdot U(x)$	$\underbrace{0, 0, ..., 0}_{m}, u_0, u_1, u_2, ..., u_n, ...$
$x^{-1} \cdot (U(x) - u_0)$	$u_1, u_2, ..., u_n, ...$

Modified $\mathcal{Z}$-transform ($\alpha, \beta \in \mathbb{C}$, $\tau \in \mathbb{Z}_+$)	Resulting Function
$\alpha \cdot F(z^{-1}) + \beta \cdot G(z^{-1})$	$\alpha \cdot f(t) + \beta \cdot g(t)$
$F(z^{-1}) \cdot G(z^{-1})$	$\sum_{r=0}^{t} f(r) \cdot g(t-r)$
$\frac{1}{1-z^{-1}} \cdot F(z^{-1})$	$\sum_{r=0}^{t} f(r)$
$z^{-\tau} \cdot F(z^{-1})$	$f(t-\tau)$
$z \cdot (F(z^{-1}) - f(0))$	$f(t+1)$

Rational transfer function revisited I

Given a discrete-time function $f : \mathbb{Z}_+ \mapsto \mathbb{C}$ and some constant $\tau \in \mathbb{Z}_+$, we can express $f(t-\tau)$ as $(D^\tau f)(t)$, where D is the *delay operator*. Then,

$$\mathcal{Z}[D^\tau f] = z^{-\tau} \cdot \mathcal{Z}[f] = z^{-\tau} \cdot F(z^{-1}).$$

In other words, applying D^τ to $f : \mathbb{Z}_+ \mapsto \mathbb{C}$ corresponds to multiplying its $\mathcal{Z}$-transform $F(z^{-1})$ by $z^{-\tau}$.

Given a discrete-time linear system with a rational transfer function $T(D)$, let $X(z^{-1}) = \mathcal{Z}[x]$ and $Y(z^{-1}) = \mathcal{Z}[y]$. Then,

$$\boxed{Y(z^{-1}) = H(z^{-1}) \cdot X(z^{-1}), \text{ where } H(z^{-1}) = \frac{b_0 + b_1 \cdot z^{-1} + \ldots + b_k \cdot z^{-k}}{1 + a_1 \cdot z^{-1} + \ldots + a_k \cdot z^{-k}}}.$$

Essentially, $H(z^{-1}) = \frac{b(z^{-1})}{a(z^{-1})}$ is $T(D) = \frac{b(D)}{a(D)}$ with z^{-1} instead of D.

Note: $T(D)$ is applied to $x(t)$ to obtain $y(t)$, whereas $H(z^{-1})$ is applied to $X(z^{-1})$ to obtain $Y(z^{-1})$.

Rational transfer function revisited II

A rational transfer function $H(z^{-1})$ is often written as

$$H(z) = H(z^{-1}) \cdot \frac{z^k}{z^k} = \frac{b_0 \cdot z^k + b_1 \cdot z^{k-1} + \ldots + b_k}{z^k + a_1 \cdot z^{k-1} + \ldots + a_k} = \frac{b(z)}{a(z)}.$$

Since we are working in $\mathbb{C}$, both $a(z)$ and $b(z)$ can always be expressed as products of linear factors, allowing us to write $H(z)$ in the following form:

$$H(z) = \frac{b_0 \cdot (z-\beta_1) \cdot (z-\beta_2) \cdot \ldots \cdot (z-\beta_k)}{(z-\alpha_1) \cdot (z-\alpha_2) \cdot \ldots \cdot (z-\alpha_k)},$$

where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are called **poles** of $H(z)$, and $\beta_1, \beta_2, \ldots, \beta_k$ are called **zeros** of $H(z)$. Poles and zeros may or may not be distinct, and they may or may not be equal to 0. Since poles are zeros are complex numbers, they can be visualized as points on the complex z-plane.

Stability Criterion Revisited

A kth order discrete-time linear system is stable if and only if every **pole** of its transfer function $H(z)$ lies inside the **unit circle** on the complex z-plane.

Example I

Consider a second-order system with $H(z^{-1}) = \dfrac{2 + z^{-2}}{1 - 4 \cdot z^{-1} + 4 \cdot z^{-2}}$.

We rewrite $H(z^{-1})$ as follows:

$$H(z) = \frac{2 \cdot z^2 + 1}{z^2 - 4 \cdot z + 4} = \frac{2 \cdot (z - \frac{\mathrm{i}}{\sqrt{2}}) \cdot (z + \frac{\mathrm{i}}{\sqrt{2}})}{(z-2) \cdot (z-2)}.$$

There are two zeros $\beta_{1,2} = \pm\frac{\mathrm{i}}{\sqrt{2}}$ and two poles $\alpha_{1,2} = 2$ (one distinct pole repeated twice). Since we have a pole whose magnitude is not strictly less than 1, our system given by $H(z^{-1})$ is *unstable*.

Example II

Consider the input $x(t) = 1 \longleftrightarrow X(z^{-1}) = \frac{1}{1-z^{-1}}$. The resulting output $y(t)$ of our system with $H(z^{-1}) = \frac{2+z^{-2}}{1-4\cdot z^{-1}+4\cdot z^{-2}}$ is such that:

$$\begin{aligned} y(t) \longleftrightarrow Y(z^{-1}) &= H(z^{-1}) \cdot X(z^{-1}) = \frac{2 + z^{-2}}{(1 - 4 \cdot z^{-1} + 4 \cdot z^{-2}) \cdot (1 - z^{-1})} \\ &= \frac{11}{2} \cdot \underbrace{\frac{2 \cdot z^{-1}}{(1 - 2 \cdot z^{-1})^2}}_{t2^t} - \underbrace{\frac{1}{(1 - 2 \cdot z^{-1})^2}}_{(t+1)2^t} + \underbrace{\frac{3}{1 - z^{-1}}}_{3}. \end{aligned}$$

Therefore, $y(t) = \frac{11}{2} \cdot t2^t - (t+1)2^t + 3 = \frac{9}{2} \cdot t2^t - 2^t + 3$. Observe that the output $y(t)$ is *unbounded*, even though the input $x(t)$ is *bounded*.

Note: Partial fraction decomposition of a function $F(z^{-1}) = \frac{b(z^{-1})}{a(z^{-1})}$ is applicable when $\deg a(z^{-1}) > \deg b(z^{-1})$. If $\deg a(z^{-1}) \leq \deg b(z^{-1})$, we divide $b(z^{-1})$ by $a(z^{-1})$ first, which yields $F(z^{-1}) = q(z^{-1}) + \frac{r(z^{-1})}{a(z^{-1})}$, and then decompose $\frac{r(z^{-1})}{a(z^{-1})}$.

State-variable model

Any kth-order discrete-time linear system (SISO or MIMO) over $\mathbb{C}$ can be specified using a **state-variable model** (involving matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$):

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}\boldsymbol{x}(t), \end{aligned}$$

where $\mathbf{s}(t)$ is a k-element state vector, $\boldsymbol{x}(t)$ is an input vector, and $\boldsymbol{y}(t)$ is an output vector.

The state-variable model has two special representations for SISO systems: **controllable canonical form** and **observable canonical form.**

State-variable model: controllable canonical form

Difference-Equation Model: $\sum_{j=0}^{k} a_j \cdot y(t-j) = \sum_{j=0}^{k} b_j \cdot x(t-j)$

State-Variable Model — Controllable Canonical Form:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_k & -a_{k-1} & -a_{k-2} & \cdots & -a_2 & -a_1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} b_k - b_0 \cdot a_k & b_{k-1} - b_0 \cdot a_{k-1} & \cdots & b_2 - b_0 \cdot a_2 & b_1 - b_0 \cdot a_1 \end{bmatrix},$$

$$\boldsymbol{D} = [b_0].$$

$$\boxed{\textbf{Note: } \det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = (-1)^k(\lambda^k + a_1 \cdot \lambda^{k-1} + \ldots + a_{k-1} \cdot \lambda + a_k).}$$

State-variable model: observable canonical form

Difference-Equation Model: $\sum_{j=0}^{k} a_j \cdot y(t-j) = \sum_{j=0}^{k} b_j \cdot x(t-j)$

State-Variable Model — Observable Canonical Form:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & -a_k \\ 1 & 0 & \cdots & 0 & 0 & -a_{k-1} \\ 0 & 1 & \cdots & 0 & 0 & -a_{k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & -a_2 \\ 0 & 0 & \cdots & 0 & 1 & -a_1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} b_k - b_0 \cdot a_k \\ b_{k-1} - b_0 \cdot a_{k-1} \\ b_{k-2} - b_0 \cdot a_{k-2} \\ \vdots \\ b_2 - b_0 \cdot a_2 \\ b_1 - b_0 \cdot a_1 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{D} = [b_0].$$

Note: $\det(\boldsymbol{A} - \lambda \cdot \boldsymbol{I}) = (-1)^k(\lambda^k + a_1 \cdot \lambda^{k-1} + ... + a_{k-1} \cdot \lambda + a_k)$.

State-variable model: stability

Theorem

A discrete-time linear system is **stable** *if and only if every eigenvalue of its state matrix* $\boldsymbol{A}$ *has a magnitude strictly less than 1.*

If we represent our kth-order SISO system in one of the canonical forms (controllable or observable), the characteristic equation for its matrix $\boldsymbol{A}$ becomes $g(\lambda) = \lambda^k + a_1 \cdot \lambda^{k-1} + ... + a_{k-1} \cdot \lambda + a_k = 0$.

The *roots* of the equation $g(\lambda) = 0$ are, in fact, the *eigenvalues* of $\boldsymbol{A}$. To guarantee system stability, we must ensure that the magnitude of those roots is strictly less than 1.

Example: For a system given by $y(t) - y(t-1) + c \cdot y(t-2) = x(t)$, we have $\lambda^2 - \lambda + c = 0$. The roots are $\lambda_{1,2} = \frac{1 \pm \sqrt{1-4 \cdot c}}{2}$. If $c = 0$, the system is *unstable*. We must have $0 < c < 1$ to ensure that $|\lambda_1| < 1$ and $|\lambda_2| < 1$.

State-variable model: transfer matrix $\boldsymbol{H}(z^{-1})$

We start from the state-variable model equations:

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

Next, using $\boldsymbol{s}(t) \longleftrightarrow \boldsymbol{S}(z^{-1})$, $\boldsymbol{x}(t) \longleftrightarrow \boldsymbol{X}(z^{-1})$, and $\boldsymbol{y}(t) \longleftrightarrow \boldsymbol{Y}(z^{-1})$, we obtain:

$$\begin{aligned} z \cdot (\boldsymbol{S}(z^{-1}) - \boldsymbol{s}(0)) &= \boldsymbol{A}\boldsymbol{S}(z^{-1}) + \boldsymbol{B}\boldsymbol{X}(z^{-1}), \\ \boldsymbol{Y}(z^{-1}) &= \boldsymbol{C}\boldsymbol{S}(z^{-1}) + \boldsymbol{D}\boldsymbol{X}(z^{-1}). \end{aligned}$$

Consequently, we have $(z \cdot \boldsymbol{I} - \boldsymbol{A})\boldsymbol{S}(z^{-1}) = z \cdot \boldsymbol{s}(0) + \boldsymbol{B}\boldsymbol{X}(z^{-1})$, which is equivalent to $\boldsymbol{S}(z^{-1}) = (z \cdot \boldsymbol{I} - \boldsymbol{A})^{-1}(z \cdot \boldsymbol{s}(0) + \boldsymbol{B}\boldsymbol{X}(z^{-1}))$. Substituting this expression into the equation for $\boldsymbol{Y}(z^{-1})$ and letting $\boldsymbol{s}(0) = \boldsymbol{0}$ yields the following result:

$$\boldsymbol{Y}(z^{-1}) = \boldsymbol{H}(z^{-1})\boldsymbol{X}(z^{-1}), \text{ where } \boldsymbol{H}(z^{-1}) = \boldsymbol{C}(z \cdot \boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}.$$

What is $\boldsymbol{T}(t)$ such that $\boldsymbol{H}(z^{-1}) \longleftrightarrow \mathcal{Z}[\boldsymbol{T}(t)]$?

We can rewrite $\boldsymbol{H}(z^{-1}) = \boldsymbol{C}(z \cdot \boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}$ as follows:

$$\begin{aligned} \boldsymbol{H}(z^{-1}) &= \boldsymbol{C}\left(z^{-1} \cdot \frac{1}{\boldsymbol{I} - z^{-1} \cdot \boldsymbol{A}}\right)\boldsymbol{B} + \boldsymbol{D} \\ &= \boldsymbol{C}(z^{-1} \cdot \mathcal{Z}[\boldsymbol{A}^t])\boldsymbol{B} + \boldsymbol{D}. \end{aligned}$$

Since $\mathcal{Z}[\boldsymbol{A}^t]$ corresponds to the sequence $(\boldsymbol{A}^0, \boldsymbol{A}^1, \boldsymbol{A}^2..., \boldsymbol{A}^t, ...)$, we obtain the sequence $(\boldsymbol{0}, \boldsymbol{A}^0, \boldsymbol{A}^1, ..., \boldsymbol{A}^{t-1}, ...)$ from $z^{-1} \cdot \mathcal{Z}[\boldsymbol{A}^t]$. Our $\boldsymbol{H}(z^{-1})$ also contains $\boldsymbol{D}$: it is the $\mathcal{Z}$-transform of the sequence $(\boldsymbol{D}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}, ...)$. Thus,

$$\boldsymbol{H}(z^{-1}) \longleftrightarrow \boldsymbol{T}(t) = \begin{cases} \boldsymbol{D}, & t = 0; \\ \boldsymbol{C}\boldsymbol{A}^{t-1}\boldsymbol{B}, & t > 0. \end{cases}$$

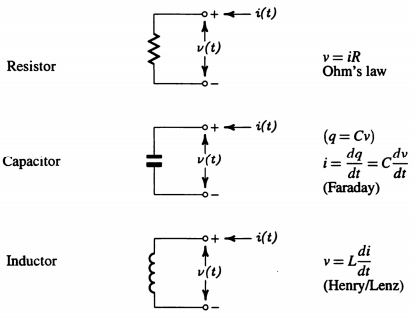
Since $\boldsymbol{H}(z^{-1})\boldsymbol{X}(z^{-1}) \longleftrightarrow \sum_{r=0}^{t} \boldsymbol{T}(r)\boldsymbol{x}(t-r) = \sum_{r=0}^{t} \boldsymbol{T}(t-r)\boldsymbol{x}(r)$, we obtain the following result, under the assumption that $\boldsymbol{s}(0) = \boldsymbol{0}$:

$$\begin{aligned} \boldsymbol{Y}(z^{-1}) \longleftrightarrow \boldsymbol{y}(t) &= \sum_{r=0}^{t-1} \boldsymbol{T}(t-r)\boldsymbol{x}(r) + \boldsymbol{T}(0)\boldsymbol{x}(t) \\ &= \sum_{r=0}^{t-1} \boldsymbol{C}\boldsymbol{A}^{t-1-r}\boldsymbol{B}\boldsymbol{x}(r) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

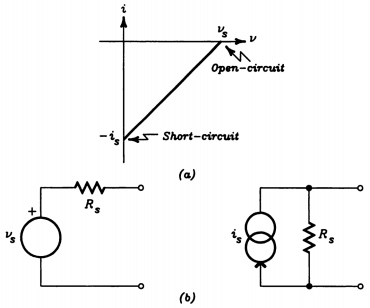
Linear Electric Networks

Linear electric networks as digraphs I

Passive elements R, C, L:



Non-ideal constant sources with internal resistance $R_s = v_s/i_s$:



* Graphics from *Circuit Analysis* by Whitehouse (web).

Linear electric networks as digraphs II

Any linear electric network with m elements and n nodes can be modeled as a *weighted digraph* $\mathcal{D} = (V, E, \boldsymbol{i}, \boldsymbol{v})$:

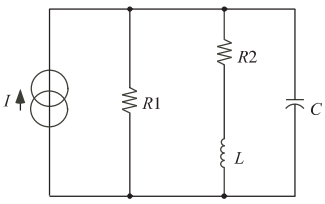
- Nodes in $V = \{v_1, v_2, ..., v_n\}$ represent n electric network nodes,
- Arcs in $E = \{e_1, e_2, ..., e_m\}$ represent m electric network elements,
  - **Note:** Arc directions are chosen arbitrarily.
- $\boldsymbol{i} = [\,\mathrm{I}_1(t)\ \mathrm{I}_2(t)\ ...\ \mathrm{I}_m(t)\,]^\top$ is a vector of arc **currents**,
- $\boldsymbol{v} = [\,\mathrm{V}_1(t)\ \mathrm{V}_2(t)\ ...\ \mathrm{V}_m(t)\,]^\top$ is a vector of arc **voltages.**

If we replace our (arbitrarily oriented) arcs in $\mathcal{D}$ with *undirected edges*, we obtain a *graph* $\mathcal{G}$ that depicts the underlying *network topology* and has the following properties:

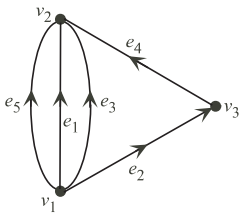
- $\mathcal{G}$ is *connected* and has *no self-loops*;
- Every edge in $\mathcal{G}$ belongs to some cycle in $\mathcal{G}$;
- There may be multiple edges connecting the same pair of nodes in $\mathcal{G}$.

Network example: digraph

Electric network:



Its digraph $\mathcal{D}$:



Edge mapping:

$$\begin{array}{rcl} R1 & \to & e_1 \\ L & \to & e_2 \\ C & \to & e_3 \\ R2 & \to & e_4 \\ I & \to & e_5 \end{array}$$

Suppose we have a linear electric network shown above, where:

- $I(t) = 10\cos(\omega t)$ A, where $\omega = 1000$ rad/s;
- $R1 = R2 = 10\ \Omega$, $C = 100\ \mu$F, $L = 10$ mH.

We wish to find the current vector $\boldsymbol{i}$ and the voltage vector $\boldsymbol{v}$. Note that $\mathrm{I}_5(t) = I(t)$ is given.

* Graphics from *Graph Theory* by Ruohonen (web).

A bit of network theory I

Kirchhoff's Current Law (KCL):
The algebraic sum of currents flowing out of any network node is zero.

Kirchhoff's Voltage Law (KVL):
The algebraic sum of voltages around any network loop is zero.

The steady-state network response to the excitation $A\cos(\omega t + \phi)$ may be found in three steps:

1. Determine the network response to the **complex** excitation $e^{\mathrm{i}\omega t}$;
2. Multiply this response by $Ae^{\mathrm{i}\phi}$;
3. Take the **real part** of the result as the final answer.

The steady-state network response to the excitation $A\sin(\omega t + \phi)$ may be found in three steps:

1. Determine the network response to the **complex** excitation $e^{\mathrm{i}\omega t}$;
2. Multiply this response by $Ae^{\mathrm{i}\phi}$;
3. Take the **imaginary part** of the result as the final answer.

A bit of network theory II

To determine the network response to $e^{\mathrm{i}\omega t}$, we express Ohm's, Faraday's, and Lenz's laws in terms of **phasors** $\tilde{\mathrm{I}}(t)$ and $\tilde{\mathrm{V}}(t)$ as shown below:

$$\begin{array}{rcl} \mathrm{V}(t) = R \cdot \mathrm{I}(t) & \longrightarrow & \tilde{\mathrm{V}}(t) = R \cdot \tilde{\mathrm{I}}(t), \\ \mathrm{I}(t) = C \cdot \frac{d\mathrm{V}(t)}{dt} & \longrightarrow & \tilde{\mathrm{V}}(t) = \frac{1}{\mathrm{i}\omega C} \cdot \tilde{\mathrm{I}}(t) = \tilde{R}_C \cdot \tilde{\mathrm{I}}(t), \\ \mathrm{V}(t) = L \cdot \frac{d\mathrm{I}(t)}{dt} & \longrightarrow & \tilde{\mathrm{V}}(t) = \mathrm{i}\omega L \cdot \tilde{\mathrm{I}}(t) = \tilde{R}_L \cdot \tilde{\mathrm{I}}(t). \end{array}$$

For our network example, we have $\tilde{\mathrm{I}}_5(t) = 10e^{\mathrm{i}1000t}$, $R1 = 10$, $R2 = 10$, $\tilde{R}_C = \frac{1}{\mathrm{i}\omega C} = -\mathrm{i}10$, and $\tilde{R}_L = \mathrm{i}\omega L = \mathrm{i}10$. Therefore,

$$\begin{array}{rcl} \tilde{\mathrm{V}}_1(t) & = & 10 \cdot \tilde{\mathrm{I}}_1(t), \\ \tilde{\mathrm{V}}_2(t) & = & \mathrm{i}10 \cdot \tilde{\mathrm{I}}_2(t), \\ \tilde{\mathrm{V}}_3(t) & = & -\mathrm{i}10 \cdot \tilde{\mathrm{I}}_3(t), \\ \tilde{\mathrm{V}}_4(t) & = & 10 \cdot \tilde{\mathrm{I}}_4(t). \end{array}$$

Network example: solution

$$\begin{array}{rl} \text{KCL at } v_1: & \tilde{\mathrm{I}}_1(t) + \tilde{\mathrm{I}}_2(t) + \tilde{\mathrm{I}}_3(t) + \tilde{\mathrm{I}}_5(t) = 0; \\ \text{KCL at } v_2: & -\tilde{\mathrm{I}}_1(t) - \tilde{\mathrm{I}}_3(t) - \tilde{\mathrm{I}}_4(t) - \tilde{\mathrm{I}}_5(t) = 0; \\ \text{KCL at } v_3: & -\tilde{\mathrm{I}}_2(t) + \tilde{\mathrm{I}}_4(t) = 0; \\ \text{KVL around } (e_3, e_1): & \tilde{\mathrm{V}}_3(t) - \tilde{\mathrm{V}}_1(t) = 0; \\ \text{KVL around } (e_5, e_1): & \tilde{\mathrm{V}}_5(t) - \tilde{\mathrm{V}}_1(t) = 0; \\ \text{KVL around } (e_5, e_3): & \tilde{\mathrm{V}}_5(t) - \tilde{\mathrm{V}}_3(t) = 0; \\ \text{KVL around } (e_2, e_4, e_1): & \tilde{\mathrm{V}}_2(t) + \tilde{\mathrm{V}}_4(t) - \tilde{\mathrm{V}}_1(t) = 0; \\ \text{KVL around } (e_2, e_4, e_3): & \tilde{\mathrm{V}}_2(t) + \tilde{\mathrm{V}}_4(t) - \tilde{\mathrm{V}}_3(t) = 0; \\ \text{KVL around } (e_5, e_4, e_2): & \tilde{\mathrm{V}}_5(t) - \tilde{\mathrm{V}}_4(t) - \tilde{\mathrm{V}}_2(t) = 0. \end{array}$$

$$\begin{array}{ll} \tilde{\mathrm{I}}_1(t) = 6.3246e^{\mathrm{i}(1000t+2.8198)}, & \tilde{\mathrm{V}}_1(t) = 63.2456e^{\mathrm{i}(1000t+2.8198)}; \\ \tilde{\mathrm{I}}_2(t) = 4.4721e^{\mathrm{i}(1000t+2.0344)}, & \tilde{\mathrm{V}}_2(t) = 44.7214e^{\mathrm{i}(1000t-2.6779)}; \\ \tilde{\mathrm{I}}_3(t) = 6.3246e^{\mathrm{i}(1000t-1.8925)}, & \tilde{\mathrm{V}}_3(t) = 63.2456e^{\mathrm{i}(1000t+2.8198)}; \\ \tilde{\mathrm{I}}_4(t) = 4.4721e^{\mathrm{i}(1000t+2.0344)}, & \tilde{\mathrm{V}}_4(t) = 44.7214e^{\mathrm{i}(1000t+2.0344)}; \\ \tilde{\mathrm{I}}_5(t) = 10e^{\mathrm{i}1000t}\ \text{(given)}, & \tilde{\mathrm{V}}_5(t) = 63.2456e^{\mathrm{i}(1000t+2.8198)}. \end{array}$$

Incidence and circuit matrices

Incidence Matrix: $\boldsymbol{A} = [a_{jk}]_{n\times m}$, where

$$a_{jk} = \begin{cases} 1 & \text{if arc } e_k \text{ is incident out of node } v_j, \\ -1 & \text{if arc } e_k \text{ is incident into node } v_j, \\ 0 & \text{if arc } e_k \text{ is not incident on node } v_j. \end{cases}$$

Circuit Matrix: $\boldsymbol{B} = [b_{jk}]_{r\times m}$, where r is the number of **distinct circuits** $C_1, C_2, ..., C_r$ in $\mathcal{D}$ (i.e., the number of distinct loops in a network), and

$$b_{jk} = \begin{cases} 1 & \text{if arc } e_k \text{ belongs to circuit } C_j \text{ of the same orientation}, \\ -1 & \text{if arc } e_k \text{ belongs to circuit } C_j \text{ of the opposite orientation}, \\ 0 & \text{if arc } e_k \text{ does not belong to circuit } C_j. \end{cases}$$

Given $\boldsymbol{A}$ and $\boldsymbol{B}$, we can write our network equations in the following form:

$$\text{KCL: } \boldsymbol{Ai} = \boldsymbol{0}, \qquad \text{KVL: } \boldsymbol{Bv} = \boldsymbol{0}.$$

Network example: $\boldsymbol{A}$ and $\boldsymbol{B}$

In our network example, we have $n = 3$, $m = 5$, and $r = 6$. We choose the column ordering $[\, e_1 \;\; e_2 \;\; e_3 \;\; e_4 \;\; e_5 \,]$ for both $\boldsymbol{A}$ and $\boldsymbol{B}$. Then,

$$\text{KCL: } \boldsymbol{A}\boldsymbol{i} = \boldsymbol{0} \longrightarrow \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ -1 & 0 & -1 & -1 & -1 \\ 0 & -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{\mathrm{I}}_1(t) \\ \tilde{\mathrm{I}}_2(t) \\ \tilde{\mathrm{I}}_3(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_5(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\text{KVL: } \boldsymbol{B}\boldsymbol{v} = \boldsymbol{0} \longrightarrow \begin{bmatrix} -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_2(t) \\ \tilde{\mathrm{V}}_3(t) \\ \tilde{\mathrm{V}}_4(t) \\ \tilde{\mathrm{V}}_5(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Cutset matrices I

Solving equations $\boldsymbol{A}\boldsymbol{i} = \boldsymbol{0}$ and $\boldsymbol{B}\boldsymbol{v} = \boldsymbol{0}$ can be computationally expensive for large networks. However, it is possible to obtain a reduced system of network equations, thus decreasing the complexity of finding $\boldsymbol{i}$ and $\boldsymbol{v}$.

For that purpose, we need to introduce so-called *cutset matrices*.

What is a cut?

Given a **connected graph** $\mathcal{G} = (V, E)$, let $V_1 \subseteq V$ and $V_2 = V \setminus V_1$ form a **partition** of the node set V. Then, the set of edges having one incident node in V_1 and the other in V_2, written as $\langle V_1, V_2 \rangle$, are called a **cut** in $\mathcal{G}$. Removing those edges effectively breaks $\mathcal{G}$ into two connected components.

The above definition is applicable to linear electric networks, because their underlying graphs are *connected*.

Cutset matrices II

Cutset Matrix: $\boldsymbol{Q} = [q_{jk}]_{s \times m}$, where s is the number of **distinct cutsets** $K_1, K_2, ..., K_s$ in $\mathcal{D}$, and

$$q_{jk} = \begin{cases} 1 & \text{if arc } e_k \text{ belongs to cutset } K_j \text{ of the same orientation,} \\ -1 & \text{if arc } e_k \text{ belongs to cutset } K_j \text{ of the opposite orientation,} \\ 0 & \text{if arc } e_k \text{ does not belong to cutset } K_j. \end{cases}$$

Stating that some $K_j = \langle V_1, V_2 \rangle$ and $e_k \in K_j$ have the same orientation means that e_k is directed from V_1 to V_2. If e_k is directed from V_2 to V_1 instead, we say that e_k and K_j are of the opposite orientation.

Note: Any *digraph* $\mathcal{D}$ representing a given network is simply an arbitrarily oriented graph $\mathcal{G}$ of that network. Therefore, arcs forming *distinct cutsets* in $\mathcal{D}$ correspond to the edges forming *unique cuts* in $\mathcal{G}$.

Network example: $\boldsymbol{Q}$

In our network example, we have $s = 3$. Our distinct cutsets are as follows:

$$\begin{aligned} K_1&: \quad \langle \{v_1\}, \{v_2, v_3\} \rangle = \{e_1, e_2, e_3, e_5\}; \\ K_2&: \quad \langle \{v_1, v_3\}, \{v_2\} \rangle = \{e_1, e_3, e_4, e_5\}; \\ K_3&: \quad \langle \{v_1, v_2\}, \{v_3\} \rangle = \{e_2, e_4\}. \end{aligned}$$

Other possible cutsets have different orientations but not different arc sets, e.g., $\langle \{v_1, v_2\}, \{v_3\} \rangle = \langle \{v_3\}, \{v_1, v_2\} \rangle$. As such they are not distinct.

Consequently, for our chosen column ordering $[\, e_1 \;\; e_2 \;\; e_3 \;\; e_4 \;\; e_5 \,]$, we obtain:

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \end{bmatrix}.$$

Note that $\boldsymbol{Q} = \boldsymbol{A}$ in our example. In general, an incidence matrix $\boldsymbol{A}$ of a given network will be a *submatrix* of a cutset matrix $\boldsymbol{Q}$ of that network.

Fundamental circuits

Given a *connected graph* $\mathcal{G} = (V, E)$, let $\mathcal{T} = (V, E_1)$ be some **spanning tree** of $\mathcal{G}$, and let $\mathcal{T}^* = (V, E_2)$ denote its **spanning co-tree** having the edge set $E_2 = E \setminus E_1$. Every edge in $\mathcal{T}^*$ is called a **chord** of $\mathcal{T}$.

Adding a chord e^* from $\mathcal{T}^*$ to $\mathcal{T}$ produces a **unique cycle** in $\mathcal{G}$. If $\mathcal{G}$ is a model of some linear electric network, such a cycle (with respect to e^*) corresponds to a **fundamental circuit** in a digraph $\mathcal{D}$ of that network.

We know that $|V| = n$, $|E| = m$, and $|E_1| = |V| - 1$. In other words, there are as many as $|E_2| = |E| - |E_1| = m - n + 1$ fundamental circuits in $\mathcal{D}$.

Therefore, we need only $m - n + 1$ rows from our circuit matrix $\boldsymbol{B}$, which yields an $(m - n + 1) \times m$ **fundamental circuit matrix**, written as $\boldsymbol{B}_f$.

Fundamental cutsets

While each co-tree edge e^* in $\mathcal{T}^*$ produces a *unique cycle* in $\mathcal{G}$, each tree edge e in $\mathcal{T}$ produces a **unique cut** in $\mathcal{G}$ (with respect to e), which is a *minimum-size cut* that contains e and no other edge of $\mathcal{T}$. There are as many as $|E_1| = n - 1$ such cuts in $\mathcal{G}$.

If $\mathcal{G}$ is a model of some linear electric network, a unique cut (with respect to some e) corresponds to a distinct **fundamental cutset** in a digraph $\mathcal{D}$ of that network.

Therefore, we need only $n - 1$ rows from our cutset matrix $\boldsymbol{Q}$, which yields an $(n - 1) \times m$ **fundamental cutset matrix**, written as $\boldsymbol{Q}_f$.

Constructing $\boldsymbol{B}_f$

We start with the *identity matrix* $\boldsymbol{I}$ of size $|E_2| \times |E_2|$, where each column represents an oriented edge e^* from $\mathcal{T}^*$ (an arc in $\mathcal{D}$), and its matching row represents the fundamental circuit in $\mathcal{D}$ with respect to e^*. We let that circuit's orientation agree with the orientation of e^*.

Next, we *prepend* the columns representing the oriented edges of $\mathcal{T}$ (arcs in $\mathcal{D}$), and fill missing row entries in accordance with the corresponding fundamental circuits in $\mathcal{D}$.

As a result, we obtain $\boldsymbol{B}_f = [\ \boldsymbol{B}_{f\mathcal{T}} \mid \boldsymbol{I}\ ]$, where $\boldsymbol{B}_{f\mathcal{T}}$ denotes the portion of $\boldsymbol{B}_f$ defined by the edges of our chosen spanning tree $\mathcal{T}$.

Constructing $\boldsymbol{Q}_f$

We start with the *identity matrix* $\boldsymbol{I}$ of size $|E_1| \times |E_1|$, where each column represents an oriented edge e from $\mathcal{T}$ (an arc in $\mathcal{D}$), and its matching row represents the fundamental cutset in $\mathcal{D}$ with respect to e. We let that cutset's orientation agree with the orientation of e.

Next, we *append* the columns representing the oriented edges of $\mathcal{T}^*$ (arcs in $\mathcal{D}$), and fill missing row entries in accordance with the corresponding fundamental cutsets in $\mathcal{D}$.

As a result, we obtain $\boldsymbol{Q}_f = [\ \boldsymbol{I} \mid \boldsymbol{Q}_{f\mathcal{T}^*}\ ]$, where $\boldsymbol{Q}_{f\mathcal{T}^*}$ denotes the portion of $\boldsymbol{Q}_f$ defined by the edges of our chosen spanning co-tree $\mathcal{T}^*$.

Network example: $\boldsymbol{B}_f$ and $\boldsymbol{Q}_f$

In our network example, we choose the spanning tree $\mathcal{T}$ with the edge set $E_1 = \{e_1, e_2\}$, whose co-tree $\mathcal{T}^*$ has the edge set $E_2 = \{e_3, e_4, e_5\}$. Using the column ordering $[\, e_1\ e_2 \mid e_3\ e_4\ e_5\,]$ for both $\boldsymbol{B}_f$ and $\boldsymbol{Q}_f$, we obtain:

$$\boldsymbol{B}_f = \left[\begin{array}{cc|ccc} -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{array}\right] = [\ \boldsymbol{B}_{f\mathcal{T}} \mid \boldsymbol{I}\ ],$$

$$\boldsymbol{Q}_f = \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \end{array}\right] = [\ \boldsymbol{I} \mid \boldsymbol{Q}_{f\mathcal{T}^*}\ ].$$

Important: Both $\boldsymbol{B}_f$ and $\boldsymbol{Q}_f$ depend on our choice of a spanning tree $\mathcal{T}$ for a given network graph $\mathcal{G}$.

Reduced KCL and KVL

Reduced KCL: $\boldsymbol{Q}_f \boldsymbol{i} = \boldsymbol{0}$.

Reduced KVL: $\boldsymbol{B}_f \boldsymbol{v} = \boldsymbol{0}$.

The number of rows in our reduced system of KCL and KVL equations is smaller than that in $\boldsymbol{A}\boldsymbol{i} = \boldsymbol{0}$ combined with $\boldsymbol{B}\boldsymbol{v} = \boldsymbol{0}$ (m versus $n + r$), but the number of columns remains unchanged (equal to m).

Note: The element ordering for vectors $\boldsymbol{i}$ and $\boldsymbol{v}$ must match the column ordering used for matrices $\boldsymbol{Q}_f$ and $\boldsymbol{B}_f$.

Network example: reduced KCL and KVL

$$\text{KCL: } \boldsymbol{Q}_f \boldsymbol{i} = \boldsymbol{0} \quad \longrightarrow \quad \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \end{array}\right] \begin{bmatrix} \tilde{\mathrm{I}}_1(t) \\ \tilde{\mathrm{I}}_2(t) \\ \tilde{\mathrm{I}}_3(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_5(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\text{KVL: } \boldsymbol{B}_f \boldsymbol{v} = \boldsymbol{0} \quad \longrightarrow \quad \left[\begin{array}{cc|ccc} -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_2(t) \\ \tilde{\mathrm{V}}_3(t) \\ \tilde{\mathrm{V}}_4(t) \\ \tilde{\mathrm{V}}_5(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Observe that the element ordering for vectors $\boldsymbol{i}$ and $\boldsymbol{v}$ matches the column ordering $[\, e_1\ e_2 \mid e_3\ e_4\ e_5\,]$ used for matrices $\boldsymbol{B}_f$ and $\boldsymbol{Q}_f$.

Loop and cutset transformations

Given a spanning tree $\mathcal{T}$ and its co-tree $\mathcal{T}^*$, we introduce the following current and voltage subvectors:

- $\boldsymbol{i}_{\mathcal{T}^*}$ is a current subvector associated with the edges of $\mathcal{T}^*$,
- $\boldsymbol{v}_{\mathcal{T}}$ is a voltage subvector associated with the edges of $\mathcal{T}$.

We can obtain the entire current vector $\boldsymbol{i}$ from its subvector $\boldsymbol{i}_{\mathcal{T}^*}$; similarly, we can obtain the entire voltage vector $\boldsymbol{v}$ from its subvector $\boldsymbol{v}_{\mathcal{T}}$, as shown below.

Loop Transformation: $\boldsymbol{i} = \boldsymbol{B}_f^{\top} \boldsymbol{i}_{\mathcal{T}^*}$.

Cutset Transformation: $\boldsymbol{v} = \boldsymbol{Q}_f^{\top} \boldsymbol{v}_{\mathcal{T}}$.

Network example: loop transformation

Recall our network example, where:

- Our chosen spanning tree $\mathcal{T}$ has the edge set $\{e_1, e_2\}$, and its co-tree $\mathcal{T}^*$ has the edge set $\{e_3, e_4, e_5\}$;
- $\boldsymbol{B}_f$ has the column ordering $[\, e_1 \;\; e_2 \mid e_3 \;\; e_4 \;\; e_5 \,]$;
- $\boldsymbol{i}_{\mathcal{T}^*} = [\, \tilde{\mathrm{I}}_3(t) \;\; \tilde{\mathrm{I}}_4(t) \;\; \tilde{\mathrm{I}}_5(t) \,]^\top$.

Loop Transformation: $\boxed{\boldsymbol{i} = \boldsymbol{B}_f^\top \boldsymbol{i}_{\mathcal{T}^*}}$

$$\underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{B}_f^\top} \underbrace{\begin{bmatrix} \tilde{\mathrm{I}}_3(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_5(t) \end{bmatrix}}_{\boldsymbol{i}_{\mathcal{T}^*}} = \begin{bmatrix} -\tilde{\mathrm{I}}_3(t) - \tilde{\mathrm{I}}_4(t) - \tilde{\mathrm{I}}_5(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_3(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_5(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{\mathrm{I}}_1(t) \\ \tilde{\mathrm{I}}_2(t) \\ \tilde{\mathrm{I}}_3(t) \\ \tilde{\mathrm{I}}_4(t) \\ \tilde{\mathrm{I}}_5(t) \end{bmatrix}}_{\boldsymbol{i}}$$

Network example: Cutset transformation

Recall our network example, where:

- Our chosen spanning tree $\mathcal{T}$ has the edge set $\{e_1, e_2\}$, and its co-tree $\mathcal{T}^*$ has the edge set $\{e_3, e_4, e_5\}$;
- $\boldsymbol{Q}_f$ has the column ordering $[\, e_1 \;\; e_2 \mid e_3 \;\; e_4 \;\; e_5 \,]$;
- $\boldsymbol{v}_{\mathcal{T}} = [\, \tilde{\mathrm{V}}_1(t) \;\; \tilde{\mathrm{V}}_2(t) \,]^\top$.

Cutset Transformation: $\boxed{\boldsymbol{v} = \boldsymbol{Q}_f^\top \boldsymbol{v}_{\mathcal{T}}}$

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & -1 \\ 1 & 0 \end{bmatrix}}_{\boldsymbol{Q}_f^\top} \underbrace{\begin{bmatrix} \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_2(t) \end{bmatrix}}_{\boldsymbol{v}_{\mathcal{T}}} = \begin{bmatrix} \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_2(t) \\ \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_1(t) - \tilde{\mathrm{V}}_2(t) \\ \tilde{\mathrm{V}}_1(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{\mathrm{V}}_1(t) \\ \tilde{\mathrm{V}}_2(t) \\ \tilde{\mathrm{V}}_3(t) \\ \tilde{\mathrm{V}}_4(t) \\ \tilde{\mathrm{V}}_5(t) \end{bmatrix}}_{\boldsymbol{v}}$$

Tellegen's theorem

Theorem

If $\boldsymbol{B}_f$ and $\boldsymbol{Q}_f$ have the same column ordering, then they are **orthogonal**: *$\boldsymbol{B}_f \boldsymbol{Q}_f^\top = \boldsymbol{0}$, which implies that $\boldsymbol{B}_{f\mathcal{T}} = -\boldsymbol{Q}_{f\mathcal{T}^*}^\top$. Similarly, we can write $\boldsymbol{Q}_f \boldsymbol{B}_f^\top = \boldsymbol{0}$, which implies that $\boldsymbol{Q}_{f\mathcal{T}^*} = -\boldsymbol{B}_{f\mathcal{T}}^\top$.*

Theorem (Tellegen)

Suppose electric networks X and Y have the same digraph representation. Let $\boldsymbol{i}_X$ and $\boldsymbol{i}_Y$ denote their respective current vectors, and let $\boldsymbol{v}_X$ and $\boldsymbol{v}_Y$ denote their respective voltage vectors. Then,

- $\boldsymbol{i}_X^\top \boldsymbol{v}_Y = 0$,
- $\boldsymbol{v}_X^\top \boldsymbol{i}_Y = 0$.

If X is the same network as Y, with the current vector $\boldsymbol{i}$ and the voltage vector $\boldsymbol{v}$, then $\boldsymbol{i}^\top \boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{i} = 0$ (i.e., network power is conserved).

Logic Circuits

Boolean ring I

Let R be a **ring with unity**:

- R forms a *commutative group* under "addition" $\oplus$, where the additive identity is denoted by 0;
- R forms a *semigroup* under "multiplication" $\cdot$, and R also includes the multiplicative identity denoted by 1 (unity);
- $x \cdot (y \oplus z) = (x \cdot y) \oplus (x \cdot z)$ and $(x \oplus y) \cdot z = (x \cdot z) \oplus (y \cdot z)$ for every $x, y, z \in R$.

When R is **idempotent** (meaning that $x \cdot x = x$ for every $x \in R$), we call it a **Boolean ring**, and for every $x, y, z \in R$, we have:

- $x \oplus x = 0$,
- $x \cdot y = y \cdot x$,
- $x \oplus y = z$ if and only if $x = y \oplus z$,
- $x \oplus y = x \oplus z$ if and only if $y = z$.

Boolean ring II

Let R be a **Boolean ring.** We define the following *new operations* on R:

- Binary "alternation" denoted by $+$, whereby for every $x, y \in R$
$$\boxed{x + y = x \oplus y \oplus (x \cdot y)},$$
- Unary "complementation" denoted by $\bar{}$, whereby for every $x \in R$
$$\boxed{\overline{x} = x \oplus 1}.$$

Then, for every $x, y, z \in R$:

- $\overline{x} \oplus 1 = x$;
- $\overline{x \oplus y} = x \oplus \overline{y} = \overline{x} \oplus y$;
- $x + y = x \oplus y$ if and only if $x \cdot y = 0$ (i.e., x and y are **orthogonal**).

A **Boolean ring** and its operations $\cdot$ ("multiplication"), $+$ ("alteration"), and $\bar{}$ ("complementation") form a so-called **Boolean algebra.**

Boolean algebra I

A **Boolean algebra** is a set R with a pair of binary operations $\cdot$ and $+$, a unary operation $\bar{}$, and at least two distinct special elements, denoted by 0 and 1, such that the following laws are in effect for every $x, y, z \in R$.[3]

Table: Axioms of Boolean Algebra.

Closure:	$(x+y) \in R$	$(x \cdot y) \in R$
Identity:	$x + 0 = x$	$x \cdot 1 = x$
Complementarity:	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
Commutativity:	$x + y = y + x$	$x \cdot y = y \cdot x$
Associativity:	$(x+y)+z = x+(y+z)$	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
Distributivity:	$x + (y \cdot z) = (x+y) \cdot (x+z)$	
	$x \cdot (y+z) = (x \cdot y) + (x \cdot z)$	

[3] By convention, the operation precedence is as follows: first $\bar{}$, then $\cdot$, then $+$.

Boolean algebra II

For every $x, y, z \in R$ we also have the following additional laws, which can be stated as theorems and proven using the axioms of Boolean algebra.

Table: Main Theorems of Boolean Algebra.

Involution:	$\overline{\overline{x}} = x$	
Idempotency:	$x + x = x$	$x \cdot x = x$
Dominance:	$x + 1 = 1$	$x \cdot 0 = 0$
Absorption:	$x + (x \cdot y) = x$	$x \cdot (x+y) = x$
Redundancy:	$x + (\overline{x} \cdot y) = x + y$	$x \cdot (\overline{x} + y) = x \cdot y$
DeMorgan's:	$\overline{(x+y)} = \overline{x} \cdot \overline{y}$	$\overline{(x \cdot y)} = \overline{x} + \overline{y}$
Consensus:	$(x+y) \cdot (\overline{x}+z) \cdot (y+z) = (x+y) \cdot (\overline{x}+z)$	
	$(x \cdot y) + (\overline{x} \cdot z) + (y \cdot z) = (x \cdot y) + (\overline{x} \cdot z)$	

Logic functions

Letting $R = \mathbb{B} = \{0, 1\}$, one can define a **logic function** $f : \mathbb{B}^n \to \mathbb{B}$ using a **truth table** that specifies output $f(x_1, x_2, ..., x_n) \in \mathbb{B}$ for every possible input $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$. There are 2^{2^n} such functions.

Any logic function $f(x_1, x_2, ..., x_n)$ can be written as a **logic expression**, involving input variables $x_1, x_2, ..., x_n$ and Boolean operations $\bar{}$, $\cdot$, $+$. There exist infinitely many logic expressions that can represent a given logic function.

Typically, we want to choose a logic expression with a minimum number of *literals*, because such an expression translates into the smallest *logic circuit* implementing a function $f(x_1, x_2, ..., x_n)$ under consideration.

A **literal** is any input variable x_k or its complement $\overline{x}_k$ appearing in a logic expression that represents $f(x_1, x_2, ..., x_n)$, where $k \in I_n = \{1, 2, ..., n\}$.

Logic function example and some of its logic expressions

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$f(x_1, x_2, x_3) = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot x_3 + x_1 \cdot \overline{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x}_3$ (12 literals)

$f(x_1, x_2, x_3) = \overline{x}_1 \cdot x_3 + \overline{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x}_3$ (7 literals)

$f(x_1, x_2, x_3) = (\overline{x}_1 + \overline{x}_2) \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x}_3$ (6 literals)

Logic circuits

Any logic expression can be represented by a **logic circuit** using basic *logic gates* NOT, AND, OR that implement Boolean operations $\bar{}$, $\cdot$, $+$. One can also use XOR logic gates, whose operation is denoted by $\oplus$.

NOT

x	$f(x) = \overline{x}$
0	1
1	0

AND

x	y	$f(x,y) = x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

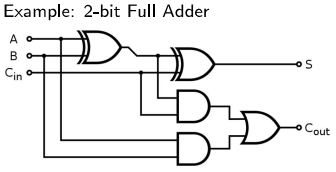
x	y	$f(x,y) = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	$f(x,y) = x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Note: $x \oplus y = x \cdot \overline{y} + \overline{x} \cdot y$.

Example: 2-bit Full Adder



$S = A \oplus B \oplus C_{in}$

$C_{out} = (A \oplus B) \cdot C_{in} + A \cdot B$

* Graphics from *Wikimedia Commons*.

Shannon expansion

Theorem (Shannon Expansion)

Any logic function $f(x_1, x_2, ..., x_k, ..., x_n)$ *can be expressed as follows:*

$$f(x_1, x_2, ..., x_k, ..., x_n) = x_k \cdot f(x_1, x_2, ..., 1, ..., x_n) + \overline{x}_k \cdot f(x_1, x_2, ..., 0, ..., x_n).$$

To prove it, we show that the equality holds for all possible values of x_k.

Case 1: $x_k = 0$. On the left-hand side, we have $f(x_1, x_2, ..., 0, ..., x_n)$, which is the same as the right-hand side evaluated at $x_k = 0$:

$$0 \cdot f(x_1, x_2, ..., 1, ..., x_n) + 1 \cdot f(x_1, x_2, ..., 0, ..., x_n) = f(x_1, x_2, ..., 0, ..., x_n).$$

Case 2: $x_k = 1$. On the left-hand side, we have $f(x_1, x_2, ..., 1, ..., x_n)$, which is the same as the right-hand side evaluated at $x_k = 1$:

$$1 \cdot f(x_1, x_2, ..., 1, ..., x_n) + 0 \cdot f(x_1, x_2, ..., 0, ..., x_n) = f(x_1, x_2, ..., 1, ..., x_n).$$

Minterms and maxterms

A **minterm** of a given n-tuple $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$ is a logic expression in the **product** form $\pi = y_1 \cdot y_2 \cdot ... \cdot y_n$, where each literal y_k is written as x_k if $x_k = 1$, or as $\overline{x}_k$ if $x_k = 0$, for every $k \in I_n$.

A **maxterm** of a given n-tuple $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$ is a logic expression in the **sum** form $\sigma = y_1 + y_2 + ... + y_n$, where each literal y_k is written as $\overline{x}_k$ if $x_k = 1$, or as x_k if $x_k = 0$, for every $k \in I_n$.

x_1	x_2	x_3	Minterm	Maxterm
0	0	0	$\overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$x_1 + x_2 + x_3$
0	0	1	$\overline{x}_1 \cdot \overline{x}_2 \cdot x_3$	$x_1 + x_2 + \overline{x}_3$
0	1	0	$\overline{x}_1 \cdot x_2 \cdot \overline{x}_3$	$x_1 + \overline{x}_2 + x_3$
0	1	1	$\overline{x}_1 \cdot x_2 \cdot x_3$	$x_1 + \overline{x}_2 + \overline{x}_3$
1	0	0	$x_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$\overline{x}_1 + x_2 + x_3$
1	0	1	$x_1 \cdot \overline{x}_2 \cdot x_3$	$\overline{x}_1 + x_2 + \overline{x}_3$
1	1	0	$x_1 \cdot x_2 \cdot \overline{x}_3$	$\overline{x}_1 + \overline{x}_2 + x_3$
1	1	1	$x_1 \cdot x_2 \cdot x_3$	$\overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Disjunctive and conjunctive normal forms I

We can write each n-tuple $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$ as an integer j computed as follows: $j = \sum_{k=1}^{n} x_k \cdot 2^{n-k} = x_1 \cdot 2^{n-1} + x_2 \cdot 2^{n-2} + ... + x_n$ (using integer arithmetic). Note that $0 \leq j \leq 2^n - 1$, and every j uniquely represents its associated n-tuple $(x_1, x_2, ..., x_n)$ in the familiar **decimal format**.[4]

Letting $j = \sum_{k=1}^{n} x_k \cdot 2^{n-k}$ represent an n-tuple $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$, we use π_j and σ_j to denote its associated minterm and maxterm.

j	x_1	x_2	x_3	Minterm π_j	Maxterm σ_j
0	0	0	0	$\pi_0 = \overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$\sigma_0 = x_1 + x_2 + x_3$
1	0	0	1	$\pi_1 = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3$	$\sigma_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$\pi_2 = \overline{x}_1 \cdot x_2 \cdot \overline{x}_3$	$\sigma_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$\pi_3 = \overline{x}_1 \cdot x_2 \cdot x_3$	$\sigma_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$\pi_4 = x_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$\sigma_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$\pi_5 = x_1 \cdot \overline{x}_2 \cdot x_3$	$\sigma_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$\pi_6 = x_1 \cdot x_2 \cdot \overline{x}_3$	$\sigma_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$\pi_7 = x_1 \cdot x_2 \cdot x_3$	$\sigma_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

[4] The bits of an n-tuple $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$ are read from right to left.

Disjunctive and conjunctive normal forms II

Using integer $j = \sum_{k=1}^{n} x_k \cdot 2^{n-k}$ to represent $(x_1, x_2, ..., x_n) \in \mathbb{B}^n$, we can write $f(x_1, x_2, ..., x_n)$ as $f_j \in \mathbb{B}$, which represents the value of a given logic function $f : \mathbb{B}^n \mapsto \mathbb{B}$ at a given n-bit input value $(x_1, x_2, ..., x_n)$.

Theorem

Every logic function $f : \mathbb{B}^n \mapsto \mathbb{B}$ has a **unique** *representation known as the* **disjunctive normal form (DNF)** *of f, which is*

$$f(x_1, x_2, ..., x_n) = \sum_{j=0}^{2^n-1} f_j \cdot \pi_j.$$

Theorem

Every logic function $f : \mathbb{B}^n \mapsto \mathbb{B}$ has a **unique** *representation known as the* **conjunctive normal form (CNF)** *of f, which is*

$$f(x_1, x_2, ..., x_n) = \prod_{j=0}^{2^n-1} (f_j + \sigma_j).$$

Disjunctive and conjunctive normal forms III

The normal forms of $f(x_1, x_2, ..., x_n)$ can also be expressed as follows:

- **DNF:** $f(x_1, x_2, ..., x_n) = \sum_{j \in P} \pi_j$, where $P = \{j \mid f_j = 1\}$,
- **CNF:** $f(x_1, x_2, ..., x_n) = \prod_{j \in S} \sigma_j$, where $S = \{j \mid f_j = 0\}$.

j	x_1	x_2	x_3	$f(x_1, x_2, x_3)$	Minterm π_j	Maxterm σ_j
0	0	0	0	$f_0 = 0$	$\pi_0 = \overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$\sigma_0 = x_1 + x_2 + x_3$
1	0	0	1	$f_1 = 1$	$\pi_1 = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3$	$\sigma_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$f_2 = 0$	$\pi_2 = \overline{x}_1 \cdot x_2 \cdot \overline{x}_3$	$\sigma_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$f_3 = 1$	$\pi_3 = \overline{x}_1 \cdot x_2 \cdot x_3$	$\sigma_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$f_4 = 0$	$\pi_4 = x_1 \cdot \overline{x}_2 \cdot \overline{x}_3$	$\sigma_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$f_5 = 1$	$\pi_5 = x_1 \cdot \overline{x}_2 \cdot x_3$	$\sigma_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$f_6 = 1$	$\pi_6 = x_1 \cdot x_2 \cdot \overline{x}_3$	$\sigma_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$f_7 = 0$	$\pi_7 = x_1 \cdot x_2 \cdot x_3$	$\sigma_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

DNF: $P = \{1, 3, 5, 6\} \longrightarrow f(x_1, x_2, x_3) = \pi_1 + \pi_3 + \pi_5 + \pi_6$.

CNF: $S = \{0, 2, 4, 7\} \longrightarrow f(x_1, x_2, x_3) = \sigma_0 \cdot \sigma_2 \cdot \sigma_4 \cdot \sigma_7$.

Ring interpretation

Let π_j and π_k be any pair of minterms representing some elements in $\mathbb{B}^n$. They are clearly **orthogonal** (i.e., $\pi_j \cdot \pi_k = 0$), since π_j is guaranteed to contain at least one variable which is complemented in π_k. Recall that orthogonality implies $\pi_j + \pi_k = \pi_j \oplus \pi_k$.

Consequently, we can rewrite any given DNF $f(x_1, x_2, ..., x_n) = \sum_{j \in P} \pi_j$, where $P = \{j \mid f_j = 1\}$, as $f(x_1, x_2, ..., x_n) = \bigoplus_{j \in P} \pi_j$, which expresses a summation of involved minterms using $\oplus$ instead of $+$. Since $\overline{x} = x \oplus 1$ for every $x \in \mathbb{B}$, any minterm π_j containing complemented variables can also be rewritten in a *complementation-free* form.

Thus, any logic function $f(x_1, x_2, ..., x_n)$ can be implemented using *only* XOR and AND gates (NOT and OR gates are not needed).

Example: AND-XOR implementation

1. Start with some sum-of-products (SOP) expression of a logic function:
$f(x_1, x_2, x_3) = x_1 \cdot x_2 + x_1 \cdot \overline{x}_2 \cdot x_3 + x_2 \cdot x_3$.

2. Apply Boolean laws to obtain a sum of *mutually orthogonal* products:
$x_2 \cdot (x_1 + x_3) + x_1 \cdot \overline{x}_2 \cdot x_3 =$ (distributivity law)
$x_2 \cdot (x_1 + \overline{x}_1 \cdot x_3) + x_1 \cdot \overline{x}_2 \cdot x_3 =$ (redundancy law)
$x_1 \cdot x_2 + x_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot x_3$. (all products are mutually orthogonal)

3. Replace $+$ with $\oplus$:
$x_1 \cdot x_2 + x_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot x_3 = x_1 \cdot x_2 \oplus x_1 \cdot \overline{x}_2 \cdot x_3 \oplus \overline{x}_1 \cdot x_2 \cdot x_3$.

4. Replace $\overline{x}$ with $x \oplus 1$ and apply distributivity:
$x_1 \cdot x_2 \oplus x_1 \cdot \overline{x}_2 \cdot x_3 \oplus \overline{x}_1 \cdot x_2 \cdot x_3 =$
$x_1 \cdot x_2 \oplus x_1 \cdot (x_2 \oplus 1) \cdot x_3 \oplus (x_1 \oplus 1) \cdot x_2 \cdot x_3 =$
$x_1 \cdot x_2 \oplus (x_1 \cdot x_2 \cdot x_3 \oplus x_1 \cdot x_3) \oplus (x_1 \cdot x_2 \cdot x_3 \oplus x_2 \cdot x_3)$.

5. Simplify (e.g., using $g \oplus g = 0$ and $h \oplus 0 = h$):
$f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_2 \cdot x_3$. (we have cancelled $x_1 \cdot x_2 \cdot x_3$)

Reed-Muller expansion

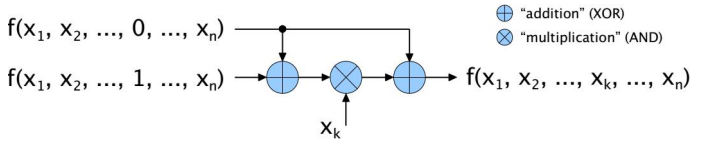
The **Boolean derivative** of a logic function $f(x_1, x_2, ..., x_k, ..., x_n)$ with respect to the variable x_k is defined as:

$$\frac{\partial f(x_1, x_2, ..., x_k, ..., x_n)}{\partial x_k} = f(x_1, x_2, ..., 0, ..., x_n) \oplus f(x_1, x_2, ..., 1, ..., x_n).$$

Theorem (Reed-Muller Expansion)

Any logic function $f(x_1, x_2, ..., x_k, ..., x_n)$ can be expressed as follows:

$$f(x_1, x_2, ..., x_k, ..., x_n) = f(x_1, x_2, ..., 0, ..., x_n) \oplus x_k \cdot \frac{\partial f(x_1, x_2, ..., x_k, ..., x_n)}{\partial x_k}.$$



Sequential logic circuits

Any sequential logic circuit with an m-bit input $\boldsymbol{x}(t)$, an n-bit output $\boldsymbol{y}(t)$, and a k-bit state vector $\boldsymbol{s}(t)$ can be viewed as a **linear MIMO machine**, where $t \in \mathbb{Z}_+$, "additions" are performed by XOR gates, "multiplications" are performed by AND gates, and $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are some suitable matrices:

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}\boldsymbol{x}(t). \end{aligned}$$

If $\boldsymbol{D} = \boldsymbol{0}$ (meaning that the output does not directly depend on the input), we refer to such a circuit as a **Moore-type finite-state machine (FSM)**; otherwise, we call it a **Mealy-type FSM.**

Digital Communications

Convolutional coding I

To enhance reliability of data communication over noisy wireless channels, an *original sequence* $X_M = (x_1, x_2, ..., x_M)$ of **data bits** is often converted into a *transmitted sequence* $W_M = (\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_M)$ of **codewords** that enter the channel — we let our codewords be n-bit vectors.

Next, a *received sequence* $Z_M = (\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_M)$ of codewords exiting the channel is decoded to generate a *decoded sequence* $\widehat{X}_M = (\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_M)$ of data bits.

Convolutional coding II

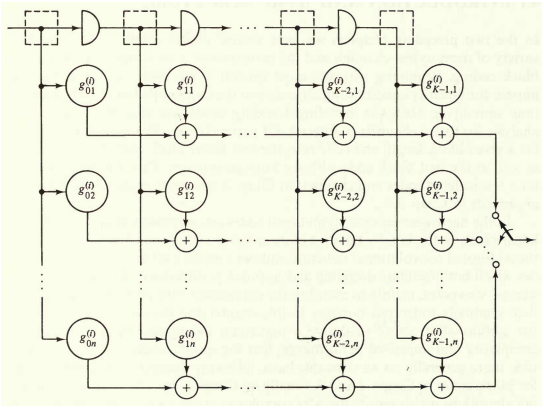
Suppose that we know the **conditional probability** $p(\boldsymbol{z} \mid \boldsymbol{w})$ of receiving a specified codeword $\boldsymbol{z}$ given a specified transmitted codeword $\boldsymbol{w}$. Assuming *independent errors*, we have $p(Z_M \mid W_M) = \prod_{m=1}^{M} p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)$.

A **maximum-likelihood (ML) decoder** receives a noisy input sequence $Z_M = (\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_M)$ of n-bit codewords and outputs the corresponding decoded sequence $\widehat{X}_M = (\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_M)$ of data bits derived from $\widehat{W}_M$, which is a *computed sequence* that satisfies the following equation:

$$p(Z_M \mid \widehat{W}_M) = \max\{\prod_{m=1}^{M} p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)\},$$

where the maximum is taken over all possible M-codeword sequences that can be generated by a **convolutional encoder** in use.

Convolutional encoder

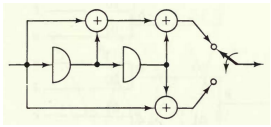


Half-circles are delay elements, boxes correspond to data bits of X_M, $\boldsymbol{g}_k^{(i)} \in \mathbb{B}^n$ are binary vectors specified for each delay tap $k \in \{0, 1, ..., K\}$ at time instance i, and big circles denote multiplications in $GF(2)$; K is called the **constraint length**.

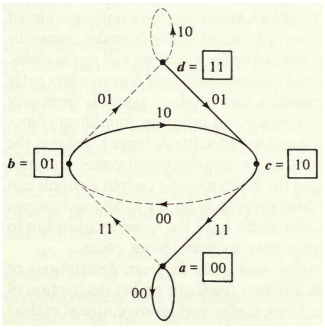
* Graphics from *Principles of Digital Communication and Coding* by Viterbi and Omura (web).

Encoder example: state diagram

Our example encoder has $K = 3$, $n = 2$, $\boldsymbol{g}_0 = (1,1)$, $\boldsymbol{g}_1 = (1,0)$, and $\boldsymbol{g}_2 = (1,1)$, which are *time-invariant*. When input x_m arrives, the encoder's current state $\boldsymbol{s}_m$ is (x_{m-1}, x_{m-2}).



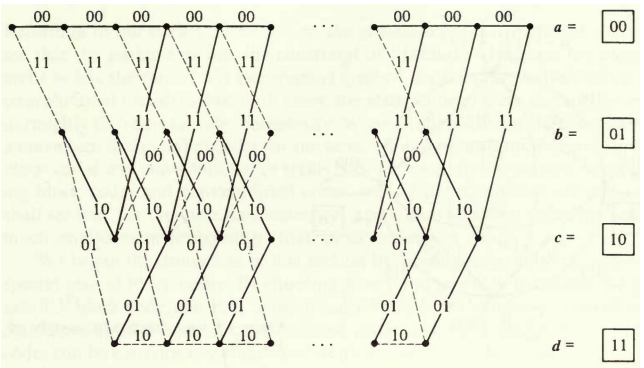
As shown in the encoder's state diagram, the possible values of $\boldsymbol{s}_m = (x_{m-1}, x_{m-2})$ are $\boldsymbol{a} = (0,0)$, $\boldsymbol{b} = (0,1)$, $\boldsymbol{c} = (1,0)$, and $\boldsymbol{d} = (1,1)$. Arc labels indicate codewords $\boldsymbol{w}_m$ produced by our encoder. A solid arc is taken, if the encoder's input is $x_m = 0$. If $x_m = 1$, a dashed arc is taken instead.



Note: This state diagram corresponds to a *Mealy-type finite-state machine*, as the output $\boldsymbol{w}_m$ depends on the current state $\boldsymbol{s}_m \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}$ and the input x_m.

* Graphics from *Principles of Digital Communication and Coding* by Viterbi and Omura (web).

Encoder example: trellis diagram



Solid branches correspond to $x_m = 0$, and dashed branches correspond to $x_m = 1$. For example, given $X_5 = (1,1,0,1,0)$ and starting from the state $\boldsymbol{a} = (0,0)$, our encoder produces $W_5 = ((1,1),(0,1),(0,1),(0,0),(1,0))$, finishing in the state $\boldsymbol{c} = (1,0)$. Note that the trellis width is 2^n, and the trellis length is $M > K$.

* Graphics from *Principles of Digital Communication and Coding* by Viterbi and Omura (web).

Decoding example I

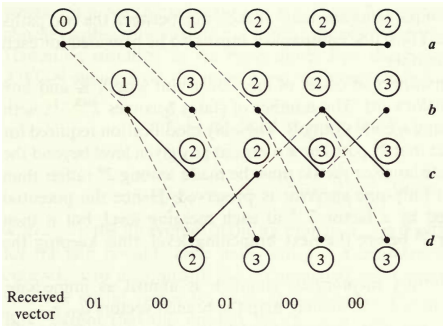
Let $Z_5 = (\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_4, \boldsymbol{z}_5) = ((0,1),(0,0),(0,1),(0,0),(0,0))$ be our received M-codeword sequence, where $M = 5$. We want to find $\widehat{X}_5 = (\widehat{x}_1, \widehat{x}_2, \widehat{x}_3, \widehat{x}_4, \widehat{x}_5)$.

We traverse our trellis starting from $\boldsymbol{s}_0 = \boldsymbol{a}$. Taking a solid ($\widehat{x}_m = 0$) or a dashed ($\widehat{x}_m = 1$) branch from $\boldsymbol{s}_{m-1}$ to $\boldsymbol{s}_m$ yields a certain *difference* between the output $\boldsymbol{w}_m$ of the encoder and the received codeword $\boldsymbol{z}_m$, where $m = 1,2,3,4,5$. It may be possible to arrive at $\boldsymbol{s}_m$ via different paths starting from $\boldsymbol{s}_0$. We always choose the path that has the *minimum distance* (i.e., the smallest total difference due to the traversal of the path's branches). For example, the path $\boldsymbol{s}_0$-$\boldsymbol{s}_1$-$\boldsymbol{s}_2 = \boldsymbol{a}$-$\boldsymbol{b}$-$\boldsymbol{d}$ has the distance of 2: the corresponding encoder's output subsequence $((1,1),(0,1))$ is different from the received codeword subsequence $((0,1),(0,0))$ in 2 bits. Note that this path represents the decoded bit subsequence $(\widehat{x}_1, \widehat{x}_2) = (1,1)$.

Our objective is to find the minimum-distance path from $\boldsymbol{s}_0$ to $\boldsymbol{s}_M$, where $M = 5$ for this example. Our constraints are represented by the trellis diagram, indicating the presence or absence of a branch between any pair of the encoder's states. Our optimal solution is $\widehat{X}_5 = (0,0,0,0,0)$, which corresponds to the path $\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$ of distance 2.

This example illustrates the decoding procedure known as the **Viterbi algorithm**.

Decoding example II



Circled numbers represent minimum distances from $\boldsymbol{s}_0 = \boldsymbol{a}$ to $\boldsymbol{s}_m \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}$, where $m = 1,2,3,4,5$. Shown trellis branches form the minimum-distance paths in question. The final state $\boldsymbol{s}_5 = \boldsymbol{a}$ has the smallest distance of 2 from the initial state $\boldsymbol{s}_0 = \boldsymbol{a}$. Tracing back from $\boldsymbol{s}_5 = \boldsymbol{a}$ to $\boldsymbol{s}_0 = \boldsymbol{a}$ yields the path $\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$-$\boldsymbol{a}$ that corresponds to the decoded sequence $(0,0,0,0,0)$.

* Graphics from *Principles of Digital Communication and Coding* by Viterbi and Omura (web).

Viterbi algorithm and dynamic programming (DP) I

The Viterbi algorithm, illustrated in the previous example, is essentially a **dynamic programming** procedure relying on *memoization* and *traceback*.

The DP stage sequence $(\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_M)$ represents our received codeword sequence $Z_M = (\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_M)$.

Each state variable $\boldsymbol{s}_m$ $(m = 1,2,...,M)$ represents an encoder's state that can be reached from the initial state $\boldsymbol{s}_0$ by traversing exactly m arcs of the encoder's state diagram.

Viterbi algorithm and dynamic programming (DP) II

Let $d_m(\boldsymbol{s}_m)$ denote the *distance* from $\boldsymbol{s}_0$ to $\boldsymbol{s}_m$ — it equals the number of bits in the received sequence $(\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_m)$ that are different from those in the output sequence of the encoder after traversing m arcs of its state diagram.

We can treat $d_m(\boldsymbol{s}_m)$ as our DP objective function to be minimized:

$$\boxed{d_m(\boldsymbol{s}_m) = \min\{d_{m-1}(\boldsymbol{s}_{m-1}) + \delta_m(\boldsymbol{s}_{m-1}, \boldsymbol{s}_m)\}},$$

where $\delta_m(\boldsymbol{s}_{m-1}, \boldsymbol{s}_m)$ represents the difference between received $\boldsymbol{z}_m$ and an encoder's output due to an arc from specified $\boldsymbol{s}_{m-1}$ to specified $\boldsymbol{s}_m$ in the encoder's state diagram (if such an arc does not exist, the corresponding δ_m is set to infinity). The final solution is given by $d_M(\boldsymbol{s}_M)$.

Viterbi algorithm and maximum-likelihood decoding I

Recall that a maximum-likelihood decoder computes $\widehat{W}_M$ that satisfies

$$p(Z_M \mid \widehat{W}_M) = \max\{\textstyle\prod_{m=1}^{M} p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)\},$$

where the maximum is taken over all possible M-codeword sequences that can be generated by an encoder in use. If data communication occurs over a **binary symmetric channel (BSC)**, then

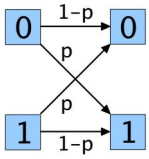
$$p(\boldsymbol{z}_m \mid \boldsymbol{w}_m) = p^{\delta_m} \cdot (1-p)^{n-\delta_m},$$

where n is the codeword size in bits. We assume that $p < 1 - p$.

Binary Symmetric Channel:

$p(0 \mid 1) = p(1 \mid 0) = p,$

$p(0 \mid 0) = p(1 \mid 1) = 1 - p.$



Viterbi algorithm and maximum-likelihood decoding II

Substituting $\ln p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)$ for $p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)$, we obtain:

$$\textstyle\prod_{m=1}^{M} p(\boldsymbol{z}_m \mid \boldsymbol{w}_m) \;\longrightarrow\; \sum_{m=1}^{M} \ln\left(p^{\delta_m} \cdot (1-p)^{n-\delta_m}\right)$$

$$= \sum_{m=1}^{M} -\delta_m \cdot \underbrace{\ln \frac{1-p}{p}}_{\alpha>0} + \underbrace{\sum_{m=1}^{M} n \cdot \ln(1-p)}_{\beta<0}.$$

Hence, we have:

$$\boxed{\max\{\textstyle\prod_{m=1}^{M} p(\boldsymbol{z}_m \mid \boldsymbol{w}_m)\} \;\longrightarrow\; \max\{-\alpha \cdot \sum_{m=1}^{M} \delta_m + \beta\}},$$

which is equivalent to finding $\min\{\sum_{m=1}^{M} \delta_m\}$.

Discrete Fourier Transform

DFT: Discrete Fourier Transform

Let $\omega^k = \exp(\mathrm{i}\frac{2\pi k}{n}) \in \mathbb{C}$, where $n \in \mathbb{N}$, $\mathrm{i} = \sqrt{-1}$, and $k \in \{0, 1, ..., n-1\}$. Each ω^k is a root of the equation $z^n - 1 = 0$ in $\mathbb{C}[z]$. Next, we define the following $n \times n$ matrix:

$$\boldsymbol{F} = [f_{jk} \in \mathbb{C}]_{n\times n} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-2} & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{n-4} & \omega^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \omega^{n-2} & \omega^{n-4} & \cdots & \omega^4 & \omega^2 \\ 1 & \omega^{n-1} & \omega^{n-2} & \cdots & \omega^2 & \omega \end{bmatrix}.$$

Observe that $f_{jk} = \omega^{j\cdot k}$ for every $j, k \in \{0, 1, ..., n-1\}$, with $\omega^0 = \omega^n = 1$ and $\omega^{-m} = \omega^{n-m}$. For example, $f_{2(n-2)} = \omega^{2\cdot(n-2)} = \omega^{2\cdot n}\omega^{-4} = \omega^{n-4}$.

We shall refer to $\boldsymbol{F}$ as the **Discrete Fourier Tranform** (**DFT**) matrix.

IDFT: Inverse Discrete Fourier Transform

The DFT matrix $\boldsymbol{F}$ has the following inverse:

$$\boldsymbol{F}^{-1} = \frac{1}{n} \cdot [g_{jk} \in \mathbb{C}]_{n\times n} = \frac{1}{n} \cdot \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & \omega^{n-1} & \omega^{n-2} & \cdots & \omega^2 & \omega \\ 1 & \omega^{n-2} & \omega^{n-4} & \cdots & \omega^4 & \omega^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{n-4} & \omega^{n-2} \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-2} & \omega^{n-1} \end{bmatrix},$$

Since $g_{jk} = \omega^{n-j\cdot k} = f_{jk}^{-1}$ for every $j, k \in \{0, 1, ..., n-1\}$, we can express the relationship between $\boldsymbol{F}^{-1}$ and $\boldsymbol{F}$ as follows: $\boldsymbol{F}^{-1}(\omega) = \frac{1}{n} \cdot \boldsymbol{F}(\omega^{-1})$.

We shall refer to $\boldsymbol{F}^{-1}$ as the **Inverse Discrete Fourier Tranform** (**IDFT**) matrix. It can be shown that $\boldsymbol{F}\boldsymbol{F}^{-1} = \boldsymbol{F}^{-1}\boldsymbol{F} = \boldsymbol{I}$.

Polynomial evaluation and interpolation

Let $\lambda_0, \lambda_1, ..., \lambda_{n-1}$ denote n distinct elements of some field F.

Given a polynomial $a(x) = \sum_{k=0}^{n-1} a_k x^k$ over a field F, we refer to finding data points $b_0 = a(\lambda_0), b_1 = a(\lambda_1), , ..., b_{n-1} = a(\lambda_{n-1})$ as **polynomial evaluation** at $x = \lambda_0, \lambda_1, ..., \lambda_{n-1} \in F$.

Given data points $b_0 = a(\lambda_0), b_1 = a(\lambda_1), ..., b_{n-1} = a(\lambda_{n-1})$, we refer to finding coefficients $a_0, a_1, ..., a_{n-1}$ of $a(x)$ as **polynomial interpolation**.

Let's express our evaluated and interpolated quantities in the vector form: $\boldsymbol{a} = [a_0\ a_1\ ...\ a_{n-1}]^\top$ and $\boldsymbol{b} = [b_0\ b_1\ ...\ b_{n-1}]^\top$, where $b_j = \sum_{k=0}^{n-1} a_k \lambda_j^k$ for every $j = 0, 1, ..., n-1$.

Polynomial evaluation is the process of deriving $\boldsymbol{b}$ from $\boldsymbol{a}$.

Polynomial interpolation is the process of deriving $\boldsymbol{a}$ from $\boldsymbol{b}$.

DFT/IDFT as polynomial evaluation/interpolation

Having $\mathbb{C}$ as our field and letting $\lambda_0 = \omega^0, \lambda_1 = \omega^1, ..., \lambda_{n-1} = \omega^{n-1} \in \mathbb{C}$, we can interpret computing $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{a}$ as *polynomial evaluation*, i.e., finding n values $b_j = \sum_{k=0}^{n-1} a_k \cdot \omega^{j \cdot k}$, where $j = 0, 1, ..., n-1$.

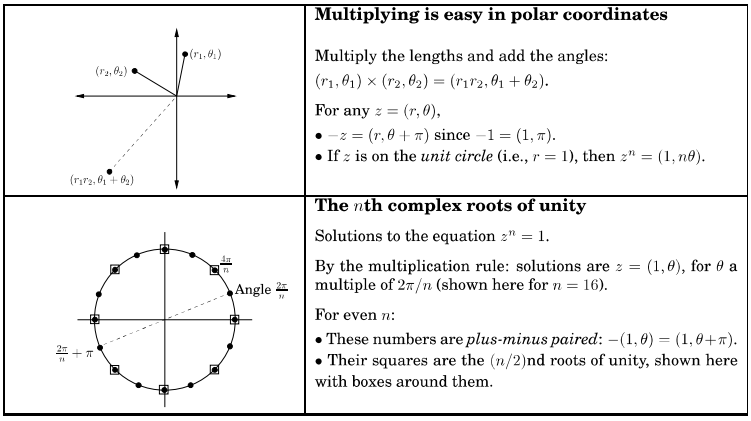
We shall refer to $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{a}$ as the DFT of $\boldsymbol{a}$.

We can interpret computing $\boldsymbol{a} = \boldsymbol{F}^{-1}\boldsymbol{b}$ as *polynomial interpolation*, i.e., finding n coefficients $a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} b_j \cdot \omega^{-j \cdot k}$, where $k = 0, 1, ..., n-1$.

We shall refer to $\boldsymbol{a} = \boldsymbol{F}^{-1}\boldsymbol{b}$ as the IDFT of $\boldsymbol{b}$.

Our next task is to figure out how to perform such polynomial evaluations and interpolations efficiently.

Complex roots of unity



Multiplying is easy in polar coordinates

Multiply the lengths and add the angles:
$(r_1, \theta_1) \times (r_2, \theta_2) = (r_1 r_2, \theta_1 + \theta_2)$.

For any $z = (r, \theta)$,

- $-z = (r, \theta + \pi)$ since $-1 = (1, \pi)$.
- If z is on the *unit circle* (i.e., $r = 1$), then $z^n = (1, n\theta)$.

The nth complex roots of unity

Solutions to the equation $z^n = 1$.

By the multiplication rule: solutions are $z = (1, \theta)$, for θ a multiple of $2\pi/n$ (shown here for $n = 16$).

For even n:

- These numbers are *plus-minus paired*: $-(1, \theta) = (1, \theta + \pi)$.
- Their squares are the $(n/2)$nd roots of unity, shown here with boxes around them.

Observe that if n is even, we have $\omega^{n/2} = \exp(\mathrm{i}\pi) = -1$, $\omega^{n/2+m} = -\omega^m$, and $(\omega^{n/2+m})^2 = (-\omega^m)^2 = \omega^{2m}$ for each $m \in \{0, 1, ..., \frac{n}{2} - 1\}$.

* Graphics from *Algorithms* by Dasgupta *et al* (web).

Polynomial evaluation using divide-and-conquer

Important: If n is even, we can represent a polynomial $a(x) = \sum_{k=0}^{n-1} a_k \cdot x^k$ as $a(x) = a_{even}(x^2) + x \cdot a_{odd}(x^2) = \sum_{k=0}^{n/2-1} a_{2k} \cdot (x^2)^k + x \cdot \sum_{k=0}^{n/2-1} a_{2k+1} \cdot (x^2)^k$. Therefore, we can find the pair of values $a(\lambda)$ and $a(-\lambda)$ based on the pair of values $a_{even}(\lambda^2)$ and $a_{odd}(\lambda^2)$:

$$\boxed{a(\lambda) = a_{even}(\lambda^2) + \lambda \cdot a_{odd}(\lambda^2)}, \qquad \boxed{a(-\lambda) = a_{even}(\lambda^2) - \lambda \cdot a_{odd}(\lambda^2)}.$$

Letting y denote x^2, we can see that $\deg a_{even}(y) = \deg a_{odd}(y) = \frac{n}{2} - 1$, whereas $\deg a(x) = n - 1$. Consequently, obtaining $a(\lambda)$ and $a(-\lambda)$ based on the values of $a_{even}(y)$ and $a_{odd}(y)$ at $y = \lambda^2$ is *less expensive* than evaluating $a(x)$ directly at $x = \lambda$ and $x = -\lambda$. This is the result of tackling our problem in three steps:

1. **Divide:** Partition our original problem of evaluating $a(x)$ at $x = \lambda$ and at $x = -\lambda$ into two *independent subproblems* of the similar type: evaluating $a_{even}(y) = \sum_{k=0}^{n/2-1} a_{2k} \cdot y^k$ and $a_{odd}(y) = \sum_{k=0}^{n/2-1} a_{2k+1} \cdot y^k$ at $y = \lambda^2$.
2. **Conquer:** Solve each subproblem independently.
3. Combine individual subproblem solutions to obtain the final solution of our original problem.

Divide-and-conquer applied to DFT I

Letting $\lambda = \omega^j$, we can now compute the pair of values $a(\omega^j)$ and $a(-\omega^j)$ for each $j = 0, 1, ..., \frac{n}{2} - 1$ as follows: $a(\omega^j) = a_{even}(\omega^{2 \cdot j}) + \omega^j \cdot a_{odd}(\omega^{2 \cdot j})$ and $a(-\omega^j) = a_{even}(\omega^{2 \cdot j}) - \omega^j \cdot a_{odd}(\omega^{2 \cdot j})$, where $a(-\omega^j) = a(\omega^{n/2+j})$.

Hence, each evaluation of the pair $a(\omega^j)$ and $a(-\omega^j)$ for $j = 0, 1, ..., \frac{n}{2} - 1$ yields the pair of values b_j and $b_{n/2+j}$ of $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{a}$.

Calculating 8-point DFT $\boldsymbol{b} = \boldsymbol{F}\boldsymbol{a}$

For $n = 8$, we have $\omega = \exp(\mathrm{i}\frac{\pi}{4})$, $\omega^2 = \exp(\mathrm{i}\frac{\pi}{2})$, and for each $j = 0, 1, 2, 3$:

$$b_j = a(\omega^j) = a_{even}(\omega^{2 \cdot j}) + \omega^j \cdot a_{odd}(\omega^{2 \cdot j}),$$
$$b_{j+4} = a(-\omega^j) = a_{even}(\omega^{2 \cdot j}) - \omega^j \cdot a_{odd}(\omega^{2 \cdot j}).$$

Note that $a_{even}(y) = a_0 + a_2 \cdot y + a_4 \cdot y^2 + a_6 \cdot y^3$ is to be evaluated at $y = \omega^{2 \cdot j}$, and $a_{odd}(y) = a_1 + a_3 \cdot y + a_5 \cdot y^2 + a_7 \cdot y^3$ is to be evaluated at $y = \omega^{2 \cdot j}$. Thus,

$$\begin{aligned}
j = 0 &\longrightarrow b_{0,4} = a(\pm 1) = a_{even}(1) \pm a_{odd}(1) \\
j = 1 &\longrightarrow b_{1,5} = a(\pm\omega) = a_{even}(\omega^2) \pm \omega \cdot a_{odd}(\omega^2) \\
j = 2 &\longrightarrow b_{2,6} = a(\pm\omega^2) = a_{even}(\omega^4) \pm \omega^2 \cdot a_{odd}(\omega^4) \\
j = 3 &\longrightarrow b_{3,7} = a(\pm\omega^3) = a_{even}(\omega^6) \pm \omega^3 \cdot a_{odd}(\omega^6)
\end{aligned}$$

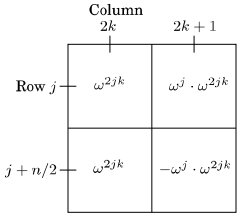
Divide-and-conquer applied to DFT II

Next, we explore how $a_{even}(\omega^{2 \cdot j})$ and $\omega^j \cdot a_{odd}(\omega^{2 \cdot j})$ are to be evaluated for each $j = 0, 1, ..., \frac{n}{2} - 1$. Letting $\boldsymbol{F}_n(\omega)$ denote our original $n \times n$ matrix $\boldsymbol{F}$, we construct a new $\frac{n}{2} \times \frac{n}{2}$ matrix $\boldsymbol{F}_{n/2}(\omega^2) = [h_{jk}]_{\frac{n}{2} \times \frac{n}{2}}$, where $h_{jk} = \omega^{2 \cdot j \cdot k}$ for every $j, k \in \{0, 1, ..., \frac{n}{2} - 1\}$:

$$\boldsymbol{F}_{n/2}(\omega) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n/2-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n/2-1} & \cdots & \omega \end{bmatrix} \longrightarrow \boldsymbol{F}_{n/2}(\omega^2) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^2 & \cdots & \omega^{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-2} & \cdots & \omega^2 \end{bmatrix}.$$

We also construct a new $n \times n$ matrix $\widehat{\boldsymbol{F}}_n(\omega)$ simply by rearranging the columns of $\boldsymbol{F}_n(\omega)$, see the figure ($j, k = 0, 1, ..., \frac{n}{2} - 1$).

To match its column ordering, we convert $\boldsymbol{a}$ to $\widehat{\boldsymbol{a}} = [a_0\ a_2\ ...\ a_{n-2} \mid a_1\ a_3\ ...\ a_{n-1}]^\top$, i.e., $\widehat{\boldsymbol{a}} = [\boldsymbol{a_{even}} | \boldsymbol{a_{odd}}]^\top$, so that $\boldsymbol{F}_n\boldsymbol{a} = \widehat{\boldsymbol{F}}_n\widehat{\boldsymbol{a}}$.



* Graphics from *Algorithms* by Dasgupta *et al* (web).

Divide-and-conquer applied to DFT III

Divide-and-conquer method for calculating $\boldsymbol{b} = \boldsymbol{F}_n(\omega)\boldsymbol{a}$

Evaluating $a_{even}(\omega^{2 \cdot j})$ over all $j = 0, 1, ..., \frac{n}{2} - 1$ is equivalent to calculating the DFT $\boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{even}}$. Evaluating $a_{odd}(\omega^{2 \cdot j})$ over all $j = 0, 1, ..., \frac{n}{2} - 1$ is equivalent to calculating the DFT $\boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{odd}}$. Note that in both cases we are using $\boldsymbol{F}_{n/2}(\omega^2)$ — the *top-left quarter submatrix* of $\widehat{\boldsymbol{F}}_n(\omega)$. Since $\boldsymbol{b} = \boldsymbol{F}_n(\omega)\boldsymbol{a}$ can be calculated as $\boldsymbol{b} = \widehat{\boldsymbol{F}}_n(\omega)\widehat{\boldsymbol{a}}$, we can determine $\boldsymbol{b}$ using the following **divide-and-conquer** method:

1. For a given even $n \in \mathbb{N}$ and $\omega = \exp(\mathrm{i}\frac{2\pi}{n}) \in \mathbb{C}$, calculate $\boldsymbol{F}_{n/2}(\omega^2)$.
2. Given an n-element vector $\widehat{\boldsymbol{a}} = [\boldsymbol{a_{even}} | \boldsymbol{a_{odd}}]^\top$, calculate intermediate $\frac{n}{2}$-element vectors $\boldsymbol{u} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{even}}$ and $\boldsymbol{v} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{odd}}$.
3. For each $j = 0, 1, ..., \frac{n}{2} - 1$, calculate $\boldsymbol{b}$'s elements $b_j = u_j + \omega^j \cdot v_j$ and $b_{n/2+j} = u_j - \omega^j \cdot v_j$.

FFT: Fast Fourier Transform

Our divide-and-conquer technique for calculating $\boldsymbol{b} = \boldsymbol{F}_n(\omega)\boldsymbol{a}$ relies on the following three ideas. First, we convert $\boldsymbol{F}_n(\omega)$ to $\widehat{\boldsymbol{F}}_n(\omega)$ and convert $\boldsymbol{a}$ to $\widehat{\boldsymbol{a}} = [\boldsymbol{a_{even}}|\boldsymbol{a_{odd}}]^\top$. Expressing the top-left quarter submatrix of $\widehat{\boldsymbol{F}}_n(\omega)$ as $\boldsymbol{F}_{n/2}(\omega^2)$, we calculate $\boldsymbol{u} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{even}}$ and $\boldsymbol{v} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{odd}}$ next. Then, we use $\boldsymbol{u}$ and $\boldsymbol{v}$ to calculate $\boldsymbol{b}$.

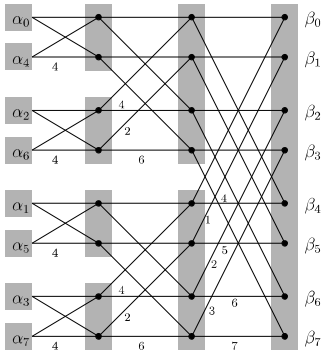
If $\frac{n}{2}$ is also even, we can apply the same divide-and-conquer steps to find $\boldsymbol{u} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{even}}$ and $\boldsymbol{v} = \boldsymbol{F}_{n/2}(\omega^2)\boldsymbol{a_{odd}}$. We convert $\boldsymbol{u}$ and $\boldsymbol{F}_{n/2}(\omega^2)$ to $\widehat{\boldsymbol{F}}_{n/2}(\omega^2)$ and $\widehat{\boldsymbol{u}} = [\boldsymbol{u_{even}}|\boldsymbol{u_{odd}}]^\top$, respectively. Next, we use $\boldsymbol{F}_{n/4}(\omega^4)$, the top-left quarter submatrix of $\widehat{\boldsymbol{F}}_{n/2}(\omega^2)$, to obtain $\frac{n}{4}$-element vectors $\boldsymbol{s} = \boldsymbol{F}_{n/4}(\omega^4)\boldsymbol{u_{even}}$ and $\boldsymbol{t} = \boldsymbol{F}_{n/4}(\omega^4)\boldsymbol{u_{odd}}$. Finally, we obtain $\boldsymbol{u}$ from $\boldsymbol{s}$ and $\boldsymbol{t}$. We can calculate $\boldsymbol{v}$ in a similar fashion.

If n can be expressed as 2^m for some $m \in \mathbb{N}$, we can find $\boldsymbol{b} = \boldsymbol{F}_n(\omega)\boldsymbol{a}$ by applying the divide-and-conquer technique recursively:

$$\boldsymbol{F}_n(\omega) \leftarrow \boldsymbol{F}_{n/2}(\omega^2) \leftarrow \boldsymbol{F}_{n/4}(\omega^4) \leftarrow ... \leftarrow \boldsymbol{F}_4(\omega^{n/4}) \leftarrow \boldsymbol{F}_2(\omega^{n/2}).$$

This is known as the **Fast Fourier Transform (FFT) algorithm.**

Example: 8-Point FFT network



We have applied our divide-and-conquer recursion three times to build the 8-point FFT network shown above. Edge labels k denote multiplications by ω^k (no labels are shown for $k = 0$). Dots represent summations of the edge data arriving from the left, while $[\alpha_0\ \alpha_1\ ...\ \alpha_7]^\top$ and $[\beta_0\ \beta_1\ ...\ \beta_7]^\top$ represent $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively.

* Graphics from *Algorithms* by Dasgupta *et al* (web).

Computational complexity of FFT

Let $T(n)$ denote the computational time required to perform n evaluations $a(\omega^j)$ and n evaluations of $a(-\omega^j)$, where $j = 0, 1, ..., n-1$. Utilizing our divide-and-conquer method, we perform $\frac{n}{2}$ evaluations of $a_{even}(\omega^{2\cdot j})$ and $\frac{n}{2}$ evaluations of $a_{odd}(\omega^{2\cdot j})$ for $j = 0, 1, ..., \frac{n}{2} - 1$, which takes $2 \cdot T(\frac{n}{2})$ time. Combining these $(\frac{n}{2} + \frac{n}{2})$ found values to obtain our final solution can be accomplished in $O(n)$ time, as we never process more than $\frac{n}{2}$ pairs of our subproblem solutions, and each pair is processed in *constant time* $O(1)$.[5]

It takes $O(n)$ time to partition our original problem of $(n + n)$ evaluations into two independent subproblems, each involving $\frac{n}{2}$ evaluations, and then combining subproblem solutions. Hence, we have $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$.

Applying the **master theorem** to our case with $a = 2$, $b = 2$, and $d = 1$, we obtain: $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d) \longrightarrow T(n) = O(n \cdot \log n)$.

[5] The amount of time needed to calculate $a(\pm\omega^j) = a_{even}(\omega^{2\cdot j}) \pm \omega^j \cdot a_{odd}(\omega^{2\cdot j})$, i.e., the time complexity of processing a pair of found values $a_{even}(\omega^{2\cdot j})$ and $a_{odd}(\omega^{2\cdot j})$, is *independent* of the problem size n.

Traveling Salesman Problem

Two examples

Problem: Task sequencing with setup costs.

Consider N independent tasks $\{T_1, T_2, ..., T_N\}$ to be executed sequentially on some machine. If task T_k immediately follows task T_j in the execution sequence, the corresponding *setup cost* is $c_{jk} > 0$. Tasks may be executed in any order, but each task must run exactly once. We need to identify a task execution sequence (to be replicated over time), such that the total setup cost is minimized.

Problem: Hole drilling in printed circuit boards (PCB).

Consider a PCB with N identical holes to be drilled as fast as possible at specified locations (to maximize the PCB production rate). Equivalently, our objective is to minimize the total distance travelled by a moving drill, which must visit every hole location exactly once and then return to its starting point (for processing the next board of the same PCB design).

TSP: Traveling salesman problem

Our last two examples are instances of the **traveling salesman problem (TSP)**, where a ficticious salesman seeks a minimum-cost **Hamiltonian circuit** in a given undirected graph of cities (nodes), to be visited in any order via available intercity routes (edges with travel costs).[6]

TSP is a well-known $\mathcal{NP}$-complete problem, which can be stated formally as follows. Let $V = \{1, 2, ..., N\}$ denote a set of cities to be visited exactly once in any order. Let c_{jk} denote the cost of traveling from city j to city k directly.[7] Let $\pi : V \mapsto V$ denote a *permutation* of V. For every $x, y \in V$, $\pi(x) = y$ means that city y is number x in the travel sequence.[8] We need to determine $\pi : V \mapsto V$, such that the sum $c_{\pi(N)\pi(1)} + \sum_{i=1}^{N-1} c_{\pi(i)\pi(i+1)}$ is minimized.

Note: There are $N!$ possible permutations of the set $V = \{1, 2, ..., N\}$.

[6] A circuit is called *Hamitonian* if it visits every node of a given graph exactly once.
[7] If there no direct route from city j to city k, we simply let $c_{jk} = \infty$.
[8] For example, $\pi(2) = 3$ means that city 3 is visited second.

Solving TSP: Branch-and-bound method I

An exhaustive search for an optimal permutation of N cities is impractical: such a brute-force method would take $O(N!)$ time, which is asymptotically worse than any exponential-complexity algorithm.

Alternatively, the following branch-and-bound method can find an optimal solution in $O(N^2 2^N)$ time. Let $X_{j \rightsquigarrow k}$ denote a *partial solution* representing a subset of cities $X_{j \rightsquigarrow k} \subseteq V$ on a current path from city j to city k, where j is fixed and k varies. To complete such a partial solution, we seek a path from k back to j, visiting all the cities from the subset $X_{k \rightsquigarrow j} = V \setminus X_{j \rightsquigarrow k}$. Finding such a path (of a minimum cost) is a *subproblem* associated with $X_{j \rightsquigarrow k}$ under consideration.

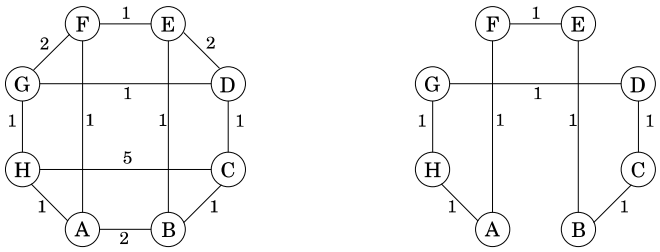
We can expand (i.e., branch out of) $X_{j \rightsquigarrow k}$ along at most $|X_{k \rightsquigarrow j}|$ directions, each corresponding to a direct route from k to one of the cities in $|X_{k \rightsquigarrow j}|$. The breadth of $X_{j \rightsquigarrow k}$ (i.e., the number of branches originating from $X_{j \rightsquigarrow k}$) in our branch-and-bound search tree is determined by $|X_{k \rightsquigarrow j}|$; whereas, the depth of $X_{j \rightsquigarrow k}$ (i.e., the number of branches leading to $X_{j \rightsquigarrow k}$) in our search tree is determined by $|X_{j \rightsquigarrow k}|$.

Solving TSP: Branch-and-bound method II

To avoid unnecessary expansion of $X_{j \rightsquigarrow k}$, we need to know the associated *lower bound*. One possible lower bound is the *sum* of the following four terms:

1. The total cost of all edges forming the path from j to k, represented by $X_{j \rightsquigarrow k}$;
2. The total cost of all edges forming a minimum-cost *spanning tree* $\mathcal{T}$ that must include all and only the cities in $X_{k \rightsquigarrow j} = V \setminus X_{j \rightsquigarrow k}$;
   - It is possible to encounter $X_{k \rightsquigarrow j}$ whose cities cannot be spanned by a tree (some cities are not connected via a direct route). In such cases, we declare the subproblem associated with $X_{j \rightsquigarrow k}$ as *infeasible*.
3. The cost of a minimum-cost edge connecting city k to $\mathcal{T}$;
4. The cost of a minimum-cost edge connecting $\mathcal{T}$ to city j.

Solving TSP: Example I



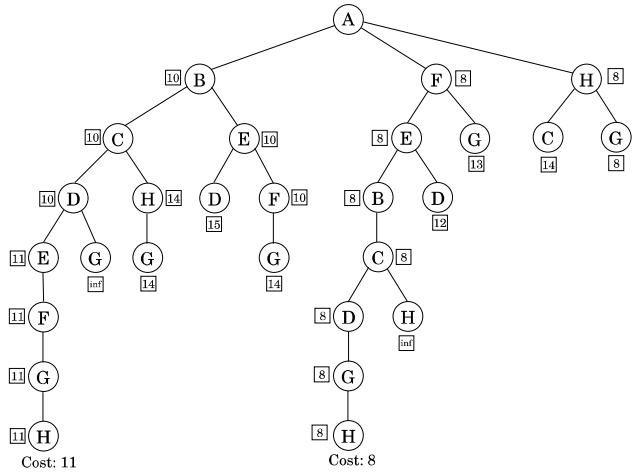
The left graph is a TSP instance with $V = \{A, B, C, D, E, F, G, H\}$.

The right graph is the minimum-cost Hamiltonian circuit, identified by our branch-and-bound method, whose starting point is city $j = A$.

In the branch-and-bound tree shown on the next slide, boxed numbers are the lower bounds.

* Graphics from *Algorithms* by Dasgupta *et al* (web).

Solving TSP: Example II



* Graphics from *Algorithms* by Dasgupta *et al* (web).

Solving TSP: Example III

Our starting city A is directly connected to B, F, and H; therefore, we can expand it into the following three subproblems.

- Given $X_{A \rightsquigarrow B} = \{A, B\}$, find the path from B back to A visiting all the cities from the subset $X_{B \rightsquigarrow A} = V \setminus X_{A \rightsquigarrow B} = \{C, D, E, F, G, H\}$. The minimum-cost spanning tree for $\{C, D, E, F, G, H\}$ is the set of edges $\{\{C, D\}, \{D, G\}, \{D, E\}, \{E, F\}, \{G, H\}\}$, whose total cost is 6. Our partial solution $X_{A \rightsquigarrow B}$ itself has the cost of 2, due to the edge $\{A, B\}$. We also need to connect the endpoints of $X_{A \rightsquigarrow B}$ (i.e., cities A and B) to our spanning tree via the cheapest edges (i.e., $\{A, F\}$ and $\{B, E\}$), which adds 2 to the total cost. The lower bound is $6 + 2 + 2 = 10$.
- Given $X_{A \rightsquigarrow F} = \{A, F\}$, find the path from F back to A visiting all the cities from the subset $X_{F \rightsquigarrow A} = V \setminus X_{A \rightsquigarrow F} = \{B, C, D, E, G, H\}$. The lower bound is $5 + 1 + 2 = 8$.
- Given $X_{A \rightsquigarrow H} = \{A, H\}$, find the path from H back to A visiting all the cities from the subset $X_{H \rightsquigarrow A} = V \setminus X_{A \rightsquigarrow H} = \{B, C, D, E, F, G\}$. The lower bound is $5 + 1 + 2 = 8$.

Solving TSP: Example IV

Now consider expanding $X_{A \rightsquigarrow H} = \{A, H\}$, which can be done in two ways, along the edges $\{H, C\}$ and $\{H, G\}$. Thus, we obtain the partial solutions $X_{A \rightsquigarrow C} = \{A, H, C\}$ and $X_{A \rightsquigarrow G} = \{A, H, G\}$. Their corresponding costs are 6 and 2, and their corresponding lower bounds are 14 and 8, in accordance with the minimum-cost spanning trees that span $X_{C \rightsquigarrow A} = \{B, D, E, F, G\}$ and $X_{G \rightsquigarrow A} = \{B, C, D, E, F\}$, respectively.

As another example, consider the partial solution $X_{A \rightsquigarrow G} = \{A, B, C, D, G\}$. To complete it, we require a path from G back to A visiting all the cities from the subset $X_{G \rightsquigarrow A} = \{E, F, H\}$. We cannot construct the associated spanning tree, and the corresponding subproblem as marked as infeasible. The partial solution $X_{A \rightsquigarrow H} = \{A, F, E, B, C, H\}$ has a similar outcome.

The first complete solution, found by the branch-and-bound search shown on slide 31, was the Hamiltonian circuit *A-B-C-D-E-F-G-H-A* of cost 11. The optimal solution is *A-F-E-B-C-D-G-H-A* of cost 8.