Problem Solving Methods: from EE to CS

Sao-Jie Chen

First Ed.: May 1st, 2002

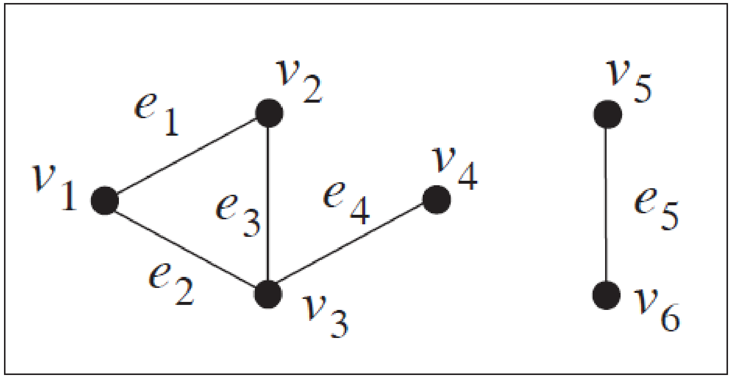
Updated Sep. 14, 2010

Background: GRAPH THEORY

Graph: A mathematical object representing a set of "points" and "interconnections" between them.

Notation: $G(V, E)$, where:

* V is the vertex set: $\{v_1, v_2, v_3, v_4, v_5, v_6\}$
* E is the edge set: $\{e_1, e_2, e_3, e_4, e_5\}$
* An edge has two endpoints, *e.g.*, $e_1 = (v_1, v_2)$



DEPTH-FIRST SEARCH

```
/* Given is the graph G(V,E) */

struct vertex {
        . . .
        int mark;
};

dfs(struct vertex v)
{
        v.mark ← 0;
        "process v";
        for each (v, u) ∈ E {
                "process (v, u)";
                if (u.mark) dfs(u);
        }
}
```

```
main ()
{
                for each v∈ V
                        v.mark ← 1;
                for each v∈ V
                        if (v.mark)
dfs(v);
}
```

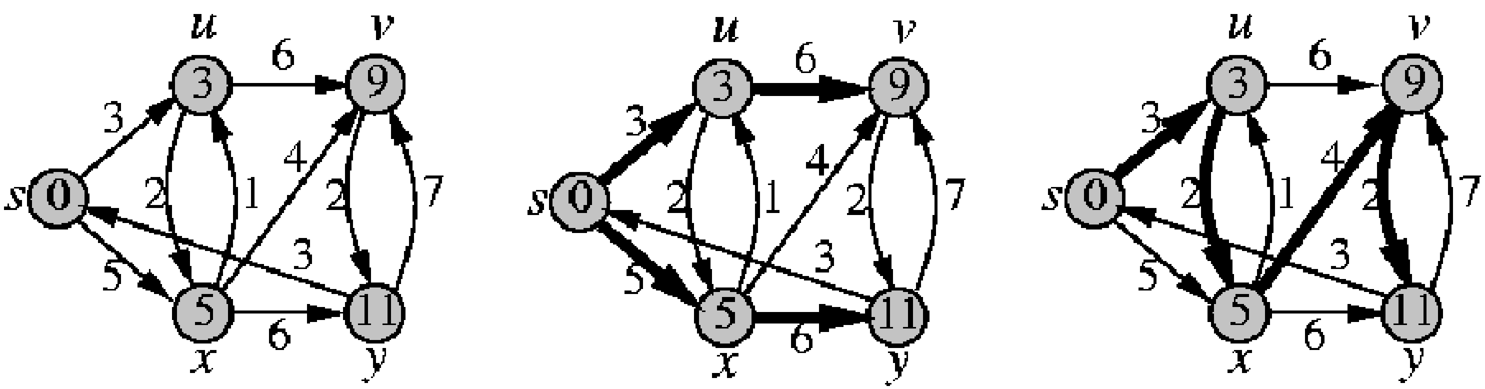
BREADTH-FIRST SEARCH

```
bfs(struct vertex v)
{
  struct fifo *Q
  struct vertex u, w;
  Q ← ();
  shif_in(Q, v);
  do {w ← shift_out(Q);
     "process w";
    for each (w, u) ∈ E {
      "process (w, u)";
      if (u.mark) {
        u.mark ← 0;
        shift_in(Q, u);;
      }
    }
  } while(Q ≠())
}
```

```
main ()
{
  for each v ∈ V
    v.mark ← 1;
  for each v ∈ V
    if (v.mark) {
      v.mark ← 1;
      bfs(v);
    }
}
```

SHORTEST-PATH Problem

- **The Shortest Path (SP) Problem**
  - **Given:** A **directed** graph *G=(V, E)* with edge weights, and a specific **source node** *s*.
  - **Goal:** Find a minimum weight path (or cost) from *s* to every other node in *V*.
- Applications: weights can be distances, times, wiring cost, delay. etc.
- **Special case:** BFS finds shortest paths for the case when all edge weights are 1.



DIJKSTRA'S SHORTEST-PATH

dijkstra(**set of struct** vertex V, **struct** vertex v_s, **struct** vertex v_t)
{
    set of struct vertex T;
    struct vertex u, v;
    $V \leftarrow V \setminus \{ v_s \}$;
    $T \leftarrow \{ v_s \}$;
    v_s.distance $\leftarrow$ 0;
    for each $u \in V$
        if $((v_s, u) \in E)$
            u.distance $\leftarrow w((v_s, u))$
        else u.distance $\leftarrow +\infty$;
    while $(v_t \notin T)$ {
        $u \leftarrow$ "$u \in$ V, such that $\forall\, v \in V$: u.distance $\leq$ v.distance";
        $T \leftarrow T \cup \{u\}$;
        $V \leftarrow V \setminus \{u\}$;
        for each v "such that $(u, v) \in E$"
            if (v.distance $> w((u, v)) +$ u.distance)
                v.distance $\leftarrow w((u, v)) +$ u.distance;
        }
}

How Does Computer Scientist (CS) Solve a Problem?

- Typically, they use a *graph* to model the problem, then apply DFS (*Depth-First Search*) or BFS (*Breadth-First Search*) to enumerate all the possible solution paths.

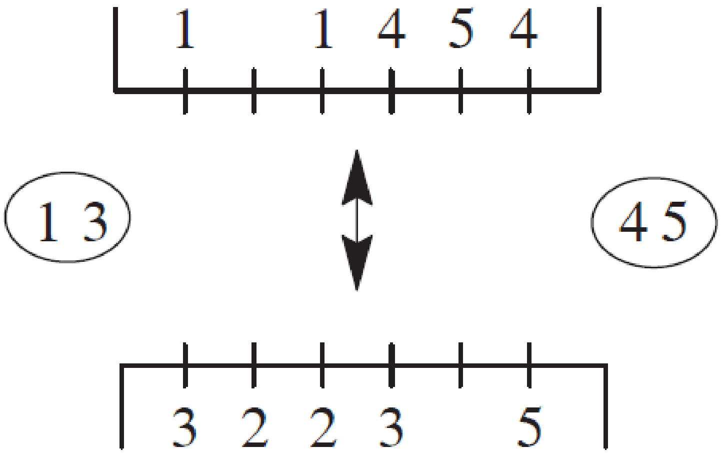
- In such a way, *Max. Flow* (or *min. Cut*) of the graph can be determined and the original problem is solved.

- Let us take a *Channel Routing* problem as an example.

CHANNEL ROUTING

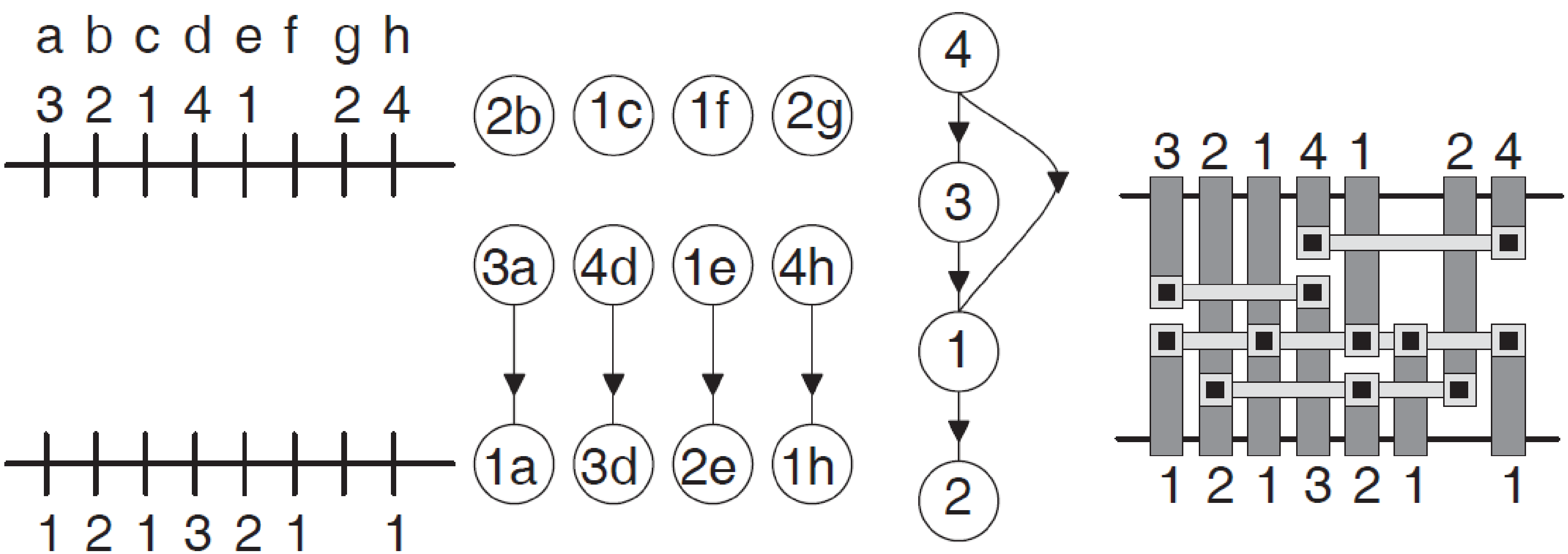
Channel routing is characterized by:

- a rectangular routing area;
- the top and bottom rectangle boundaries contain terminals with fixed positions;
- the left and right boundaries of the channel have terminals with floating positions;
- the goal is to minimize the height of the routing area.



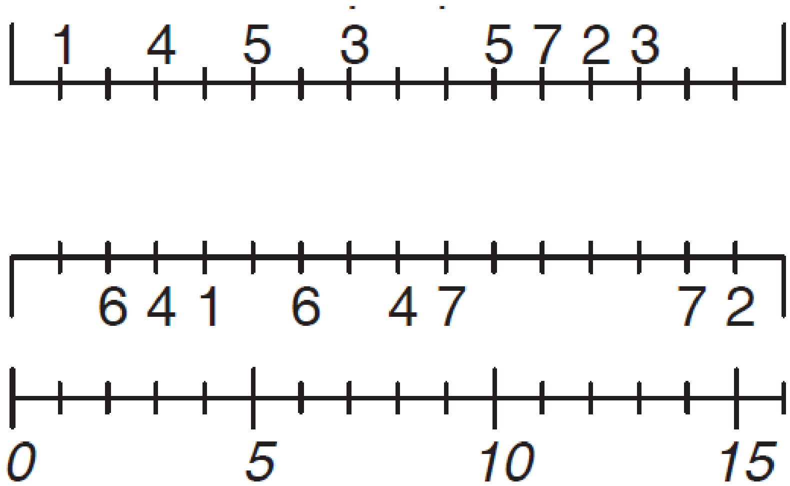
VERTICAL CONSTRAINTS

Vertical constraints can be combined into a **vertical constraint graph** under the assumption that each net will use one horizontal segment.

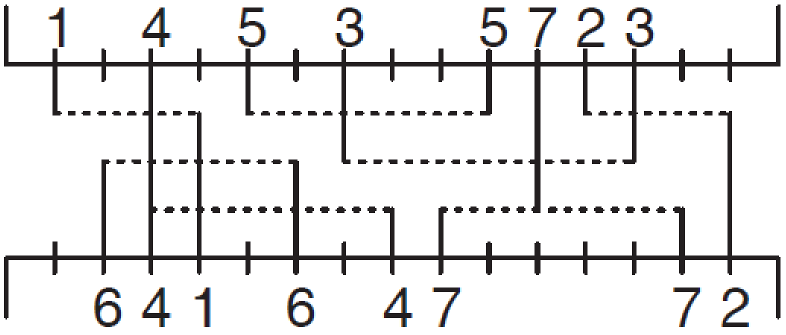


HORIZONTAL CONSTRAINTS

An example problem:



Problem solution:

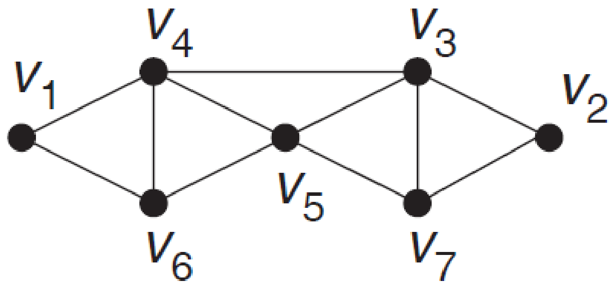


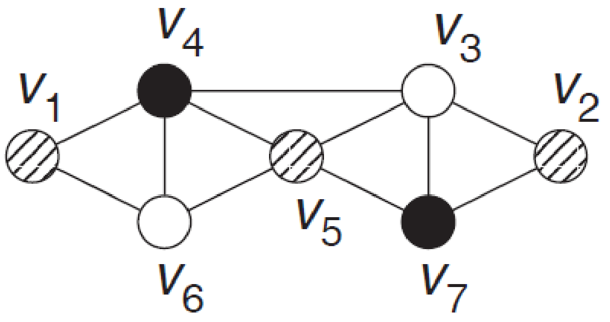
Intervals: i_1 =[1, 4], i_2 =[12, 15], i_3 =[7, 13], i_4 =[3, 8],
i_5 =[5, 10], i_6 = [2, 6], i_7 = [9, 14].

INTERVAL GRAPHS

Consider the intervals: $i_1 = [1, 4]$, $i_2 = [12, 15]$, $i_3 = [7, 13]$, $i_4 = [3, 8]$, $i_5 = [5, 10]$, $i_6 = [2, 6]$, $i_7 = [9, 14]$.

- There is a vertex for each interval.
- Vertices corresponding to overlapping intervals are connected by an edge.
- Solving the track assignment problem is equivalent to finding a minimal vertex coloring of the graph.



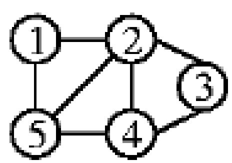


Graph Representations: Adjacency Matrix

- **Adjacency matrix:** A $|V| \times |V|$ matrix $A = (a_{ij})$ such that

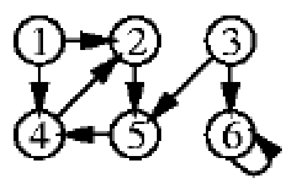
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Advantage: $O(1)$ time to find an edge.
- Drawback: $O(V^2)$ storage, more suitable for **dense** graph.
- How to save space if the graph is undirected?



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

undirected graph



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

directed graph

LINEAR PROGRAMMING (LP)

Given: matrix $\mathbf{A}$, vectors $\mathbf{b}$, $\mathbf{c}$ (constants), and the vector $\mathbf{x}$ (unknowns).

Canonical form:

* Minimize or maximize:

  $\mathbf{c}^T\mathbf{x}$

* Subject to:

  $\mathbf{Ax} \leq \mathbf{b}$

  $\mathbf{x} \geq \mathbf{0}$

Standard form:

* Minimize or maximize:

  $\mathbf{c}^T\mathbf{x}$

* Subject to:

  $\mathbf{Ax} = \mathbf{b}$

  $\mathbf{x} \geq \mathbf{0}$

- The two forms can be converted into each other.
- Solvable in polynomial time by ellipsoid algorithm; in practice better performance with simplex algorithm (exponential time complexity).

Network Flow Theorem from Operation Research

Max-Flow = min Cut (a polynomial time problem).

⇒ i.e., Maximum Clique = minimun Independent Set in Graph Theory (which is an NP-complete problem), also it is equal to the *Bi-Partite Matching* problem.

- All these good properties are from the Primal-dual characteristics of an LP, we can thus transform the problem of finding a solution in LP by looking for the possible number of paths (or loops) in a graph.

- The above results have been obtained from the field of Combinatorial Optimization or Discrete Optimization
  ⇒ *Matroid* Theory [Lawler’76 Ch. 7&8], [Papa’82, Ch. 12].

Matching: A subset M of edges with the property that no two edges of M share the same node.

Matching Problem:

Find the maximum *matching* M of a graph G.

Definition of Matroid:

Given a finite set $|S| < \infty$, with element $e \subseteq S$ and

I = a non-empty collection of (independent) subsets of S satisfying

(i) $\emptyset \in I$

(ii) If $X \in I$ and $Y \subseteq X$, then $Y \in I$, (inclusion closure)

i.e., all proper subsets of a set X in I are in I (i.e., $Y \in I$).

Then, H=(S, I) is called an *Independent System* (or *Subset System*).

If we have furthermore

(iii) $\forall$ X, Y $\in$ I, $|X| = |Y| + 1 \Rightarrow \exists\, e \in X \setminus Y$ such that $(Y + e) \in I$

Then, we call such M=(S, I) a *Matroid.*

Semi-Matching Problem:

Given an $n \times m$ matrix $\mathbf{W} = (w_{ij}) \geq 0$, choose a maximum weight subset of elements subject to no two elements in this subset X are from the same row of the matrix.

$$\mathbf{W} = \begin{pmatrix} 4 & \underline{\mathbf{6}} & 4 & 5 \\ 3 & \underline{\mathbf{8}} & 1 & 6 \\ 2 & 9 & 2 & \underline{\mathbf{10}} \\ 1 & 2 & 3 & \underline{\mathbf{18}} \end{pmatrix}$$

Let S = {elements of **W**}

I = $\{X \subseteq S \mid$ no two elements in the same row$\}$

Then, M = (S, I) is a *matroid*, solvable by a *greedy algorithm.*

(Sol.) Pick the element with the largest weight first, reject if it is in a row where an element has been selected.

Matroid Intersection Problem:

Given $M_1 = (S, I_1)$, $M_2 = (S, I_2)$, weight $W: S \longrightarrow R^+$

Find a set $X \subseteq S$ independent in both M_1 and M_2 with maximum weight W(X), such that $X \in I_1 \cap I_2$.

Such M = (S, I) where $I = \{X \subseteq S\}$ is an *Intersection Matroid*.

Matching Matroid Problem:

Given a graph $G=(V, E)$, $S \subseteq V$, and
$I = \{X \subseteq S \mid \exists$ a matching M covering all nodes in $X\}$.
Then, $M = (S, I)$ is called a *Matching Matroid*.

(Non-bipartite) Matching (NPC)

Matching Matroid Problem (3-MI)

Bi-partite Matching (Polynomial)

Matroid Intersection Problem (2-MI)

Semi-Matching (Linear Time)

(Greedy) Matroid Problem

[Theorem] A matching matroid M=(S, I) is a matroid.

Pf: (i) $\varnothing$ covered by any matching.

(ii) If $X \in I$ and $Y \subseteq X$, then clearly Y is also covered by the same matching covering X => $Y \in I$.

(iii) Suppose $X, Y \in I$, $|X| = |Y| + 1$, and matchings M_X, M_Y cover X, Y respectively.

(a) If $\exists$ $e \in X \setminus Y$, e is also covered by M_Y , then Y+e is also covered by M_Y, => $Y+e \in I$ (Cond. iii in matroid definition).

(b) Suppose $\forall$ $e \in X \setminus Y$, e is not covered by M_Y, now consider $M_X \oplus M_Y = (M_X \cup M_Y) \setminus (M_X \cap M_Y)$, there must exist one alternating path P between a node not in Y and a node $e \in X \setminus Y$ such that $M_Y \oplus P$ is a bigger matching covering Y+e, => $Y+e \in I$. Thus, M is a matroid.

Matching Matroid Example

S={1,2,3,5,6,7}

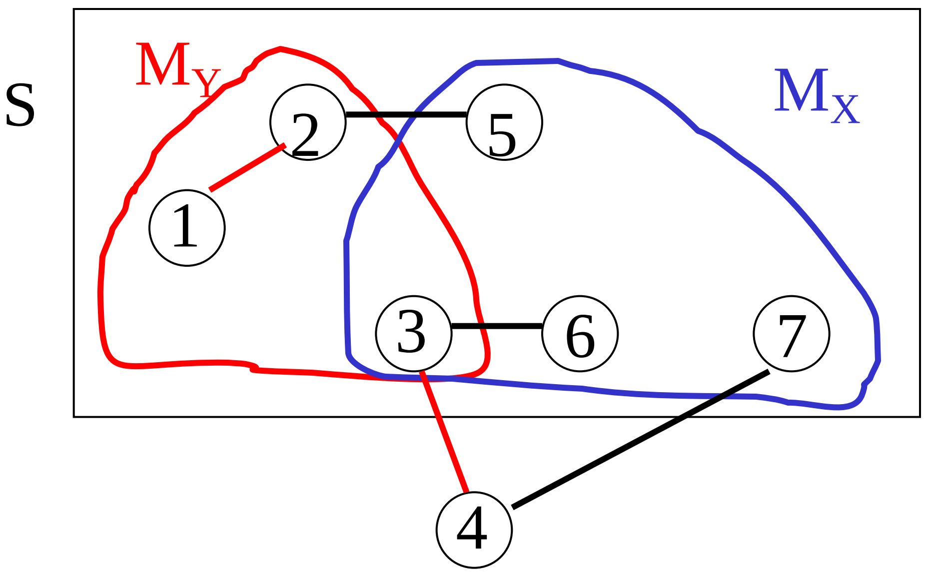
X={3,5,6,7}   Y={1,2,3}

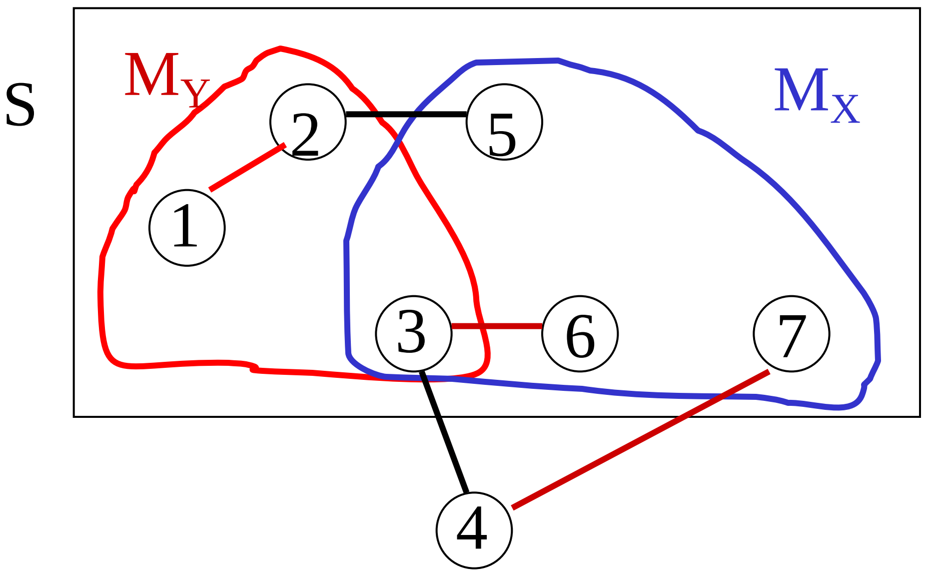
M_X={(2,5) (3,6) (4,7)}
M_Y={(1,2) (3,4)}

P={(3,6) (3,4) (4,7)}

$M_Y \oplus P$ ={(1,2) (3,6) (4,7)}
covers 1,2,3,6, and 7,
where e = 6 or 7 (or both).

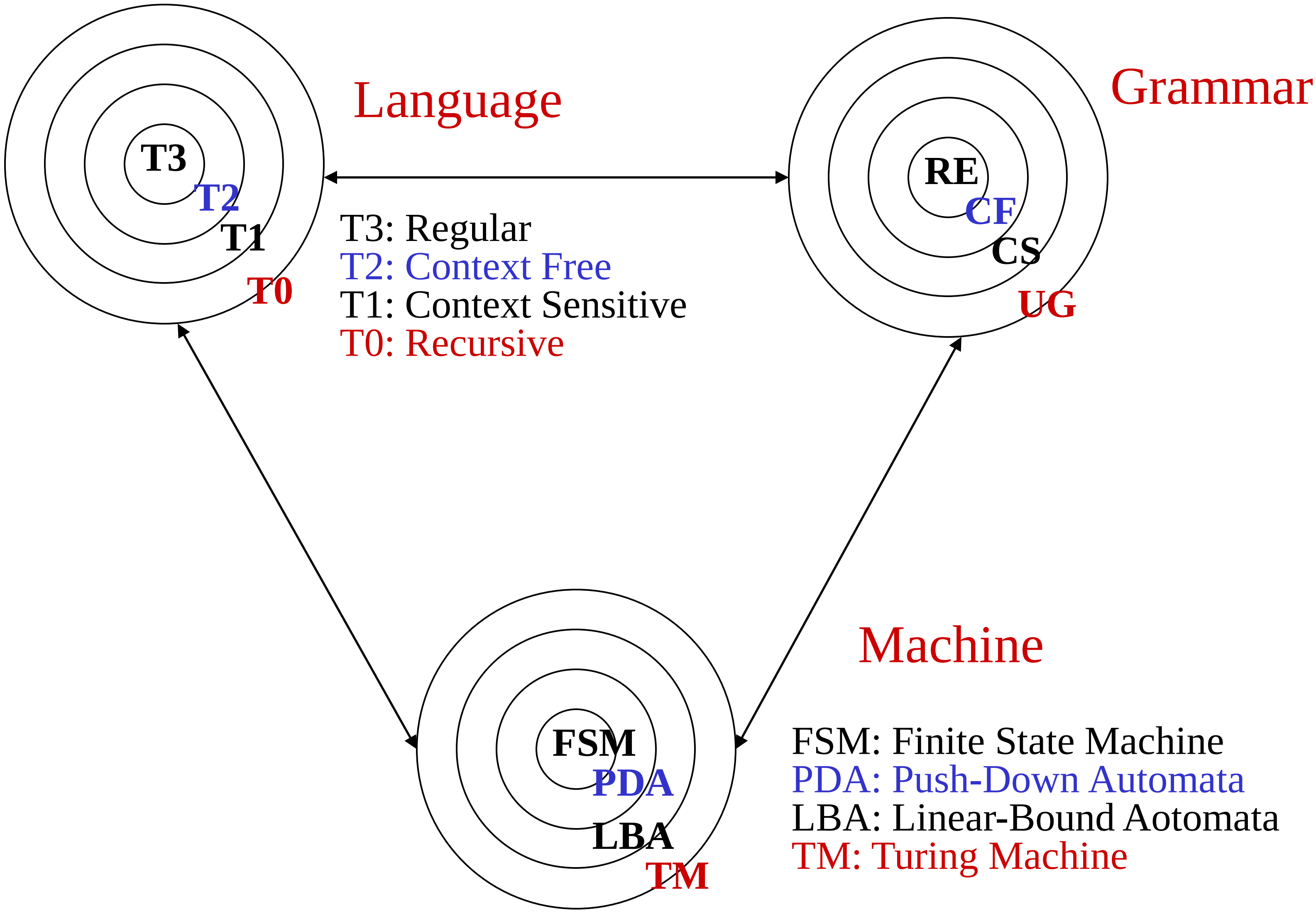
Then, we have Y+e $\in$ I.



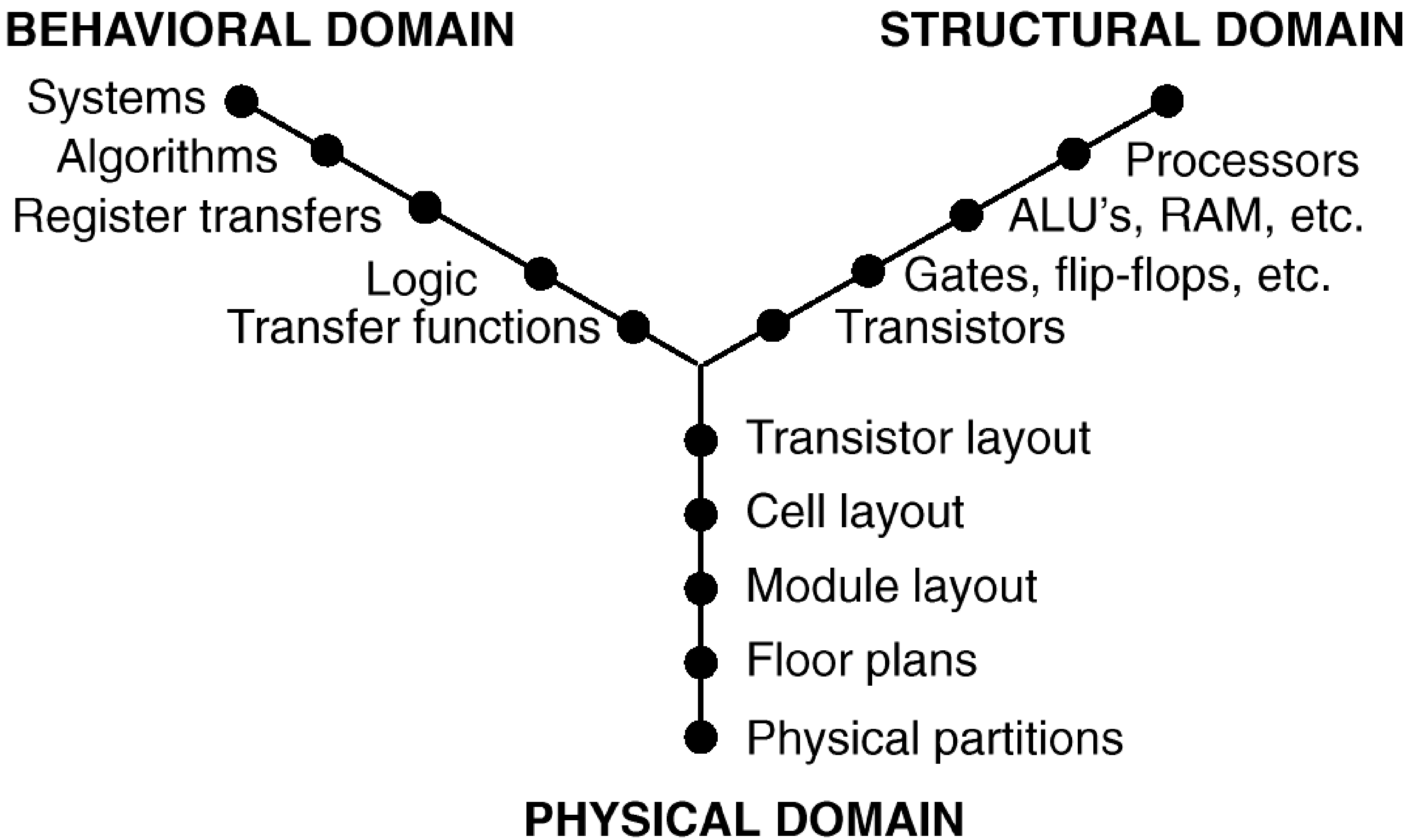


Chomsky's Classification

RE: Regular Expression
CF: Context Free
CS: Context Sensitive
UG: Unrestricted Grammar



GAJSKI'S Y- CHART



Chomsky's Classification vs Y-Chart

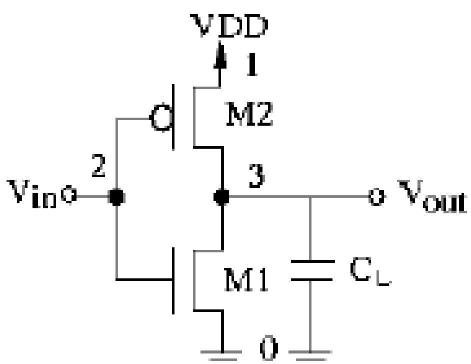
***Language*:** a S/W program, an Algorithm, or a Problem formulation
(Semantics, Behavioral description of a circuit).

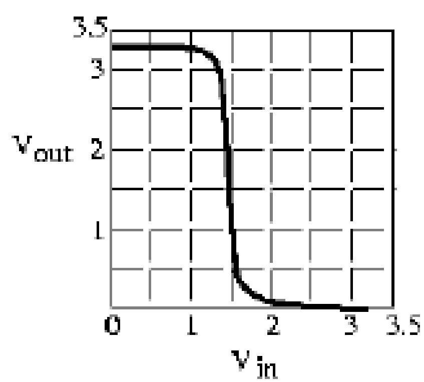
***Grammar*:** Production Rules, State-Tx diagram, Net-list
(Syntax, Structural description of a circuit).

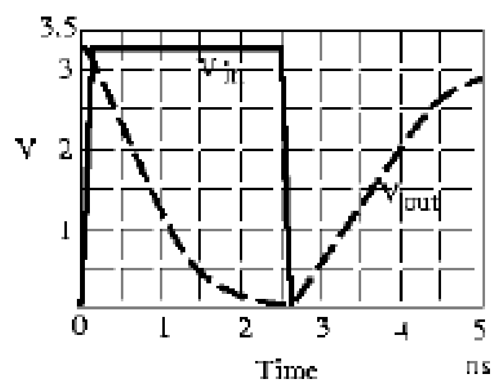
***Machine*:** a H/W solution, an Algebra, a Problem solver,
(Physical, Detailed implementation of a circuit).

Circuit Simulation of a CMOS Inverter (0.6 μm)

```
M1 3 2 0 0 nch W=1.2u L=0.6u AS=2.16p PS=4.8u AD=2.16p PD=4.8u
M2 3 2 1 1 pch W=1.8u L=0.6u AS=3.24p PS=5.4u AD=3.24p PD=5.4u
CL 3 0 0.2pF

VDD 1 0 3.3
VIN 2 0 DC 0 PULSE (0 3.3 0ns 100ps 100ps 2.4ns 5ns)

.LIB '../mod_06' typical

.OPTION NOMOD POST INGOLD=2 NUMDGT=6 BRIEF
.DC VIN 0V 3.3V 0.001V
.PRINT DC V(3)
.TRAN 0.001N 5N
.PRINT TRAN V(2) V(3)
.END
```

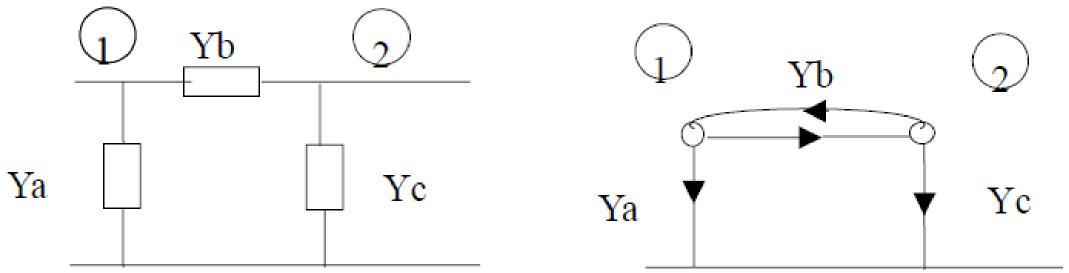






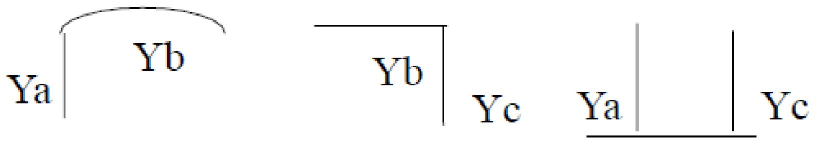
Tree (Digraph) Representation of an RLC-gm Network

Use I = YV as example, where Y is a node-admittance matrix



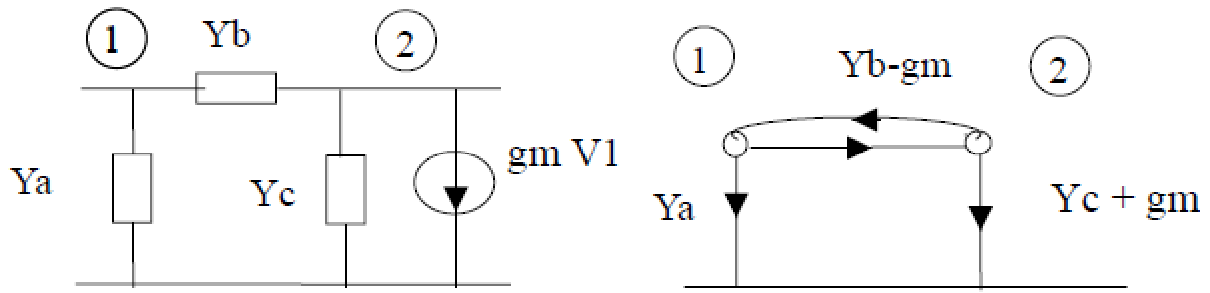
* Matrix representation:

$$Y = \begin{bmatrix} Ya + Yb & -Yb \\ -Yb & Yb + Yc \end{bmatrix}$$



$$\Delta = YaYb + YaYc + YbYb + YbYc - YbYb$$
$$= YaYb + YaYc + YbYc$$

* Tree representation: $\delta = YaYb + YbYc + YaYc$

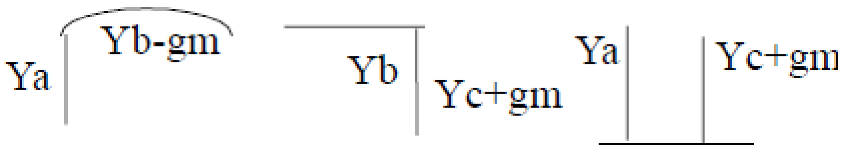


*** Matrix representation:**

$$Y = \begin{bmatrix} Ya + Yb & -Yb \\ -Yb + gm & Yb + Yc \end{bmatrix} \qquad \begin{aligned} \Delta &= YaYb + YaYc + YbYb + YbYc - YbYb + Ybgm \\ &= YaYb + YaYc + YbYc + Ybgm \end{aligned}$$

*** Tree representation:**

$$\delta = YaYb + YbYc + YaYc + Yb\,gm$$

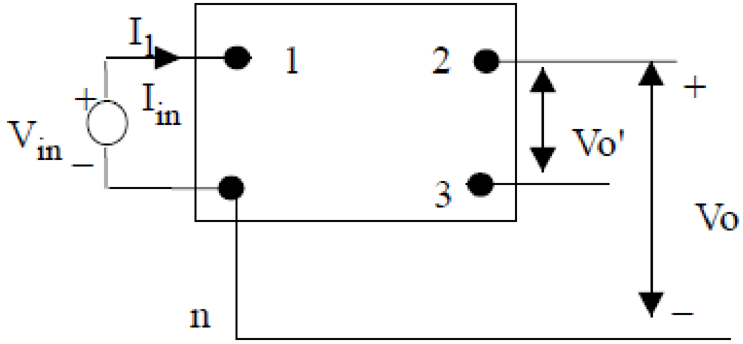


*** Relationship between Δ and δ:** $\Delta_{jj} = \delta_{jj}$

$\Delta_{jk} = \delta_{jj} - \delta_{jk}$

Network with Voltage (Current) Source Input

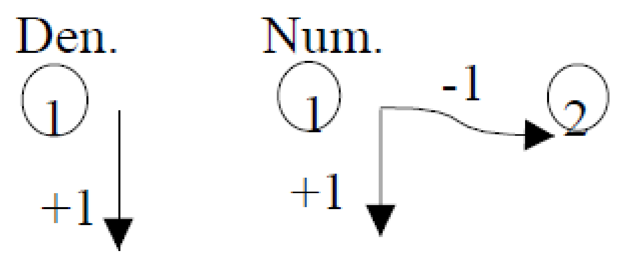
A. General Case (n: reference point)



(a) Voltage Source cases:

$$\frac{V_o}{V_{in}} = \frac{V_2}{V_1} = \frac{\Delta_{12}}{\Delta_{11}}$$

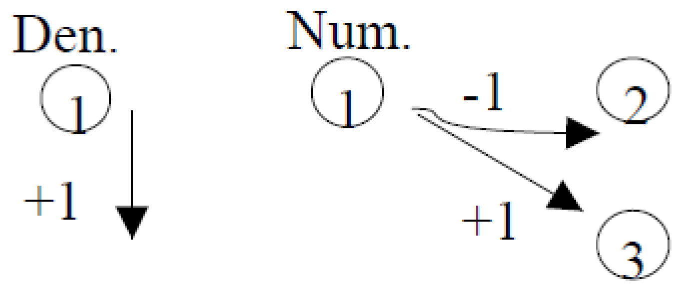
$$= \frac{\delta_{11} - \delta_{12}}{\delta_{11}} = \frac{\text{Numerator}}{\text{Denominator}}$$



(a) Voltage Source cases (Cont.):

$$\frac{V_{o'}}{V_{in}} = \frac{V_2 - V_3}{V_1} = \frac{\Delta_{12} - \Delta_{13}}{\Delta_{11}}$$

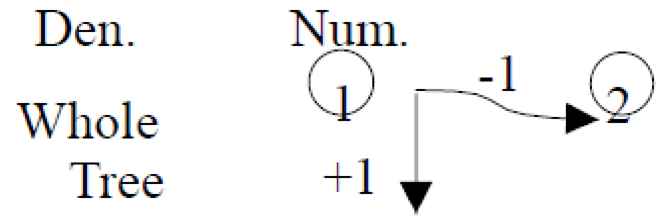
$$= \frac{\delta_{13} - \delta_{12}}{\delta_{11}} = \frac{\text{Num.}}{\text{Den.}}$$



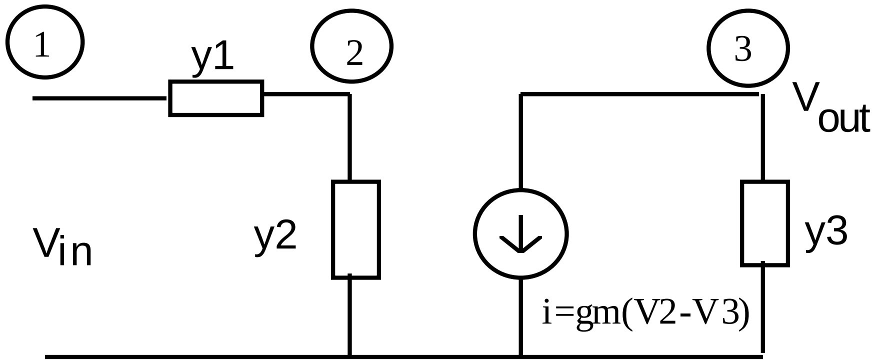
(b) Current Source Cases:

$$\frac{V_o}{I_{in}} = \frac{V_2}{I_1} = \frac{\Delta_{12}}{\Delta}$$

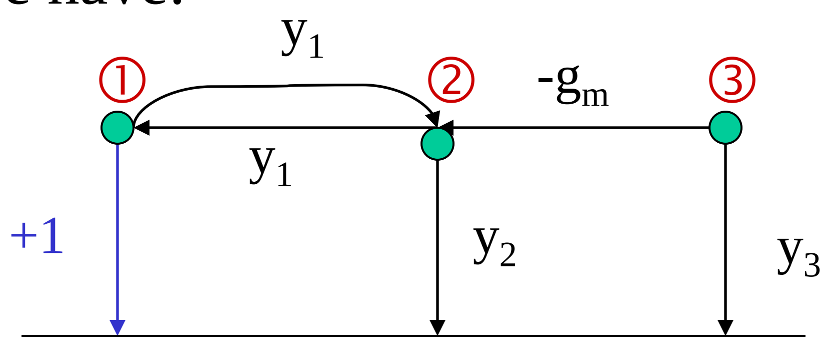
$$= \frac{\delta_{11} - \delta_{12}}{\delta} = \frac{\text{Num.}}{\text{Den.}}$$



Use digraph to find V_{out} / V_{in}

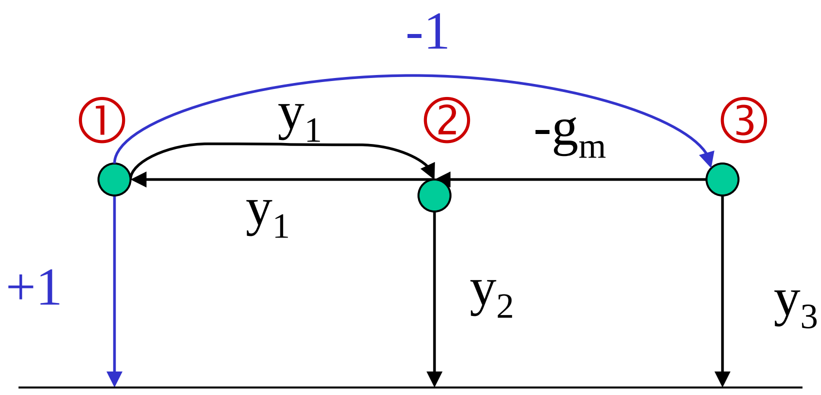


Denominator: voltage source ⇒ add a +1 branch to node ①-0, and we have:



$$V_{in} = \Delta_{11} = \delta_{11} = -y_1 g_m + y_1 y_3 + y_2 y_3 - y_2 g_m \quad \text{(Eq. a)}$$

Numerator: add +1, -1 branches to nodes ① -0, ① - ③, and we have:



$$\delta_{11} = -y_1 g_m + y_1 y_3 + y_2 y_3 - y_2 g_m \qquad \text{(Eq. a)}$$

$$\delta_{13} = y_2 g_m - y_2 y_3 - y_1 y_3 \qquad \text{(Eq. b)}$$

$$V_{out} = \Delta_{13} = \delta_{11} - \delta_{13} = \text{(Eq. a)} - \text{(Eq. b)} = -y_1 g_m$$

$$\Rightarrow \mathbf{V_{out} / V_{in} = -y_1 g_m / (-y_1 g_m + y_1 y_3 + y_2 y_3 - y_2 g_m)}$$

How Does Electrical Engineer (EE) Solve a Problem?

*** Transfer Function Finding:**

1. For a circuit network, use KVL to find $YV = I$
2. Then, use determinant and cofactor of Y to find the transfer function (e.g., $V_{out} / V_{in} = \Delta_{13} / \Delta_{11}$), $O(n!)$.

Step 1 is difficult for a computer to understand.

Step 2 Solving $\boldsymbol{V = Y^{-1} I}$ is very time consuming, even we use *Gausian Elimination* or *LU Decomposition* $O(n^3)$.

$\Rightarrow$ **Tree Enumeration** solves both Steps 1 and 2 simultaneously and gives the transfer function we need.

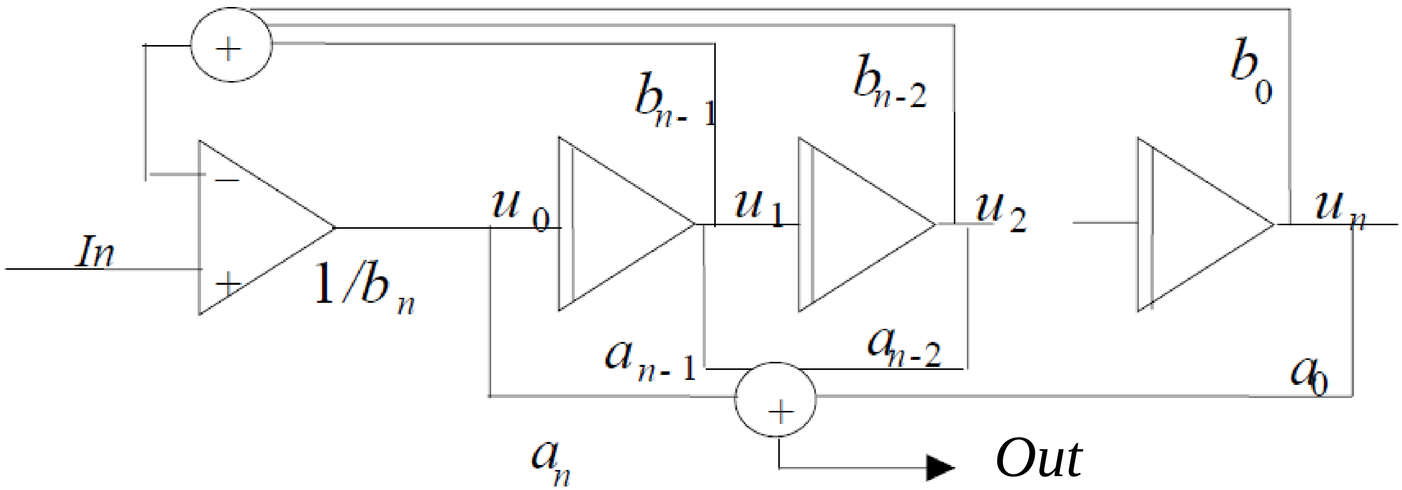
*** Time (or Frequency) Response Finding**

How to Find the Time Response?

For a general system, by defining $u_k = u_0/s^k$, we have:

$$\frac{Out}{In} = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_n s^n}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n}$$

$$= \frac{a_0 u_n + a_1 u_{n-1} + a_2 u_{n-2} + \ldots + a_n u_0}{b_0 u_n + b_1 u_{n-1} + b_2 u_{n-2} + \ldots + b_n u_0}$$

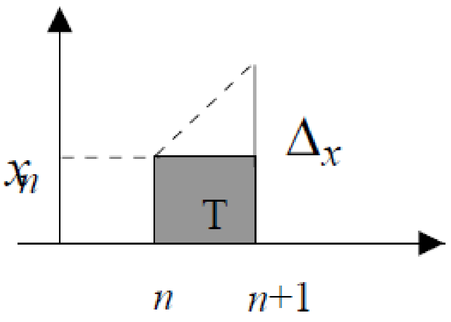


Numerical Integration

(1) Forward Euler (FE): slope

$$\dot{x}_n = \frac{\Delta x}{T}$$

$$x_{n+1} = x_n + T\dot{x}_n$$

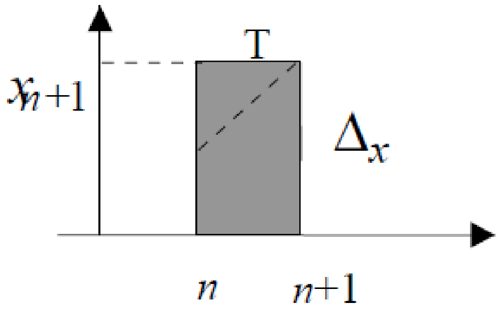


*** ST-plane of FE:** $ST = e^{j\theta} - 1$

(2) Backward Euler (BE): slope

$$\dot{x}_{n+1} = \frac{\Delta x}{T}$$

$$x_{n+1} = x_n + T\dot{x}_{n+1}$$



*** ST plane of BE:** $ST = 1 - e^{j\theta}$

*** Note that we have now z-tx from the numerical integration formulation**

$$x_n = e^{At_n} = x(t_n)$$

$$x_{n+1} = x(t_n + T) = e^{A(t_n + T)} = e^{AT} x_n = z x_n$$

$$x_{n-1} = x(t_n - T) = e^{A(t_n - T)} = e^{-AT} x_n = z^{-1} x_n$$

$$\dot{x}_n = A x_n = A e^{At_n}$$

$$\dot{x}_{n+1} = A x_{n+1}$$

$\Rightarrow$ **An analog problem can be discretized and solved digitally**

From the General Form of *Out / In*

We have also:

$$In = b_n u_0 + \sum_{k=1}^{n} b_{n-k} u_k , \quad u_0 = \left(In - \sum_{k=1}^{n} b_{n-k} u_k \right) \Big/ b_n$$

If we use BE:

$$\begin{aligned} u_{k,n+1} &= u_{k,n} + T \cdot u_{k-1,n+1} \\ &= u_{k,n} + \sum_{j=1}^{k-1} \left(T\right)^j \cdot u_{k-j,n} + \left(T\right)^k \cdot u_{0,n+1} \end{aligned}$$

we can then solve for BE the following (Eq. c):

$$u_{0,n+1} = \frac{In - \sum_{k=1}^{n} \left\{ b_{n-k} \left[u_{k,n} + \sum_{j=1}^{k-1} \left(T\right)^j \cdot u_{k-j,n} \right] \right\}}{\sum_{k=0}^{n} \left(T\right)^k \cdot b_{n-k}}$$

Linear Multi-Step (LMS) Algorithm

Step 0. Set up initial values: $u_{0,n} = In(0)/b_n$,

$u_{k,n} = 0$, **for** $k = 1, \ldots, n$

Step 1. Find new $u_{0,\, n+1}$ **using (Eq. c).**

Step 2. Find all new u_k **by using old** u_k **and** u_{k-1}

$$u_{k,n+1} = u_{k,n} + T \cdot u_{k-1,n+1} \quad \text{or}$$

$$u_{k,n+1} = u_{k,n} + \frac{T}{2} \cdot [u_{k-1,n} + u_{k-1,n+1}]$$

Step 3. Find output time function value by

$$Out = \sum_{k=0}^{n} a_{n-k} u_{k,n+1}$$

Step 4. Set $u_{k,n} = u_{k,n+1}$, **for** $k = 0, \ldots,\ n$ **goto Step 1.**

Frequency Response

$$F(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_n s^n}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n}$$

Substitute $s = jw$ **into it, plot Frequency in Hz, Amplitude in DB, and Phase in Deg.**

$$DB = 10\log\left|\frac{V_{in}}{V_{out}}\right|^2 = 20\log\left|\frac{V_{in}}{V_{out}}\right|$$

$$A + jB = \sqrt{A^2 + B^2}\angle\theta$$

CONCLUSION

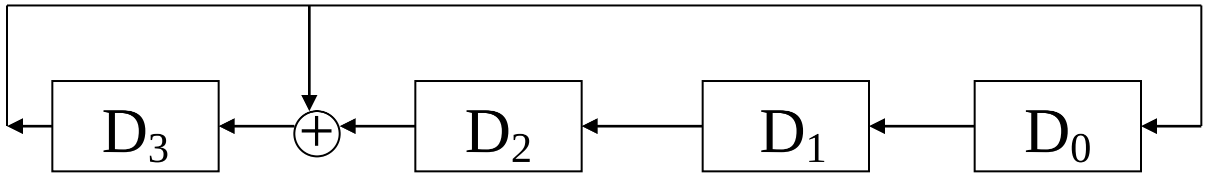
EE uses **a Tree Enumeration** method to find the transfer function and then **LMS** (which does integration *digitally* by + and $\times$) to find the time response of an *analog* circuit.

Q: Why the problem solving methods used in EE and CS look so alike?

A: We use the same machine (a general purpose *Computer*) to solve the problem in hand. Thus, we have to think *digitally* or discretely for telling the Computer how to solve the problem (see the LFSR example in next page).

Linear Feedback Shift Register

- A counter part of the general analog circuit is digital LFSR.
- A LFSR is frequently used as a modular divider to find the remainder (or called a *signature* in some other application).
- By analyzing the function of the following LFSR, we find that it can be used as a (pseudo) random number generator.



- For example: Given $D_3D_2D_1D_0$ = 0110, the above LFSR will behave as:
  0110→1100→0001→0010→0100→1000→1001→1011→1111→0111→1110→0101→1010→1100→0011→0110→ …

References:

S. H. Gerez, *Algorithms for VLSI Design Automation*, ©1999, John Wiley & Sons Ltd

E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, ©1976, Holt, Rinehart and Winston (Ch. 7 & 8 on Matroids)

C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithm and Complexity*, © 1982 Prentice-Hall (Ch. 12)

L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms & Computational Techniques*, ©1975, Prentice-Hall (Ch. 14, Tree Enumeration Method)

W. K. Chen, *Linear Networks and Systems*, 2nd Ed., ©1990, World Scientific