Chapter - 8

PATHS, PATH PRODUCTS AND REGULAR EXPRESSIONS

Motivation

- Flow graphs are being an abstract representation of programs.
- Any question about a program can be cast into an equivalent question about an appropriate flowgraph.
- Most software development, testing tools and debugging use flow graphs analysis techniques.

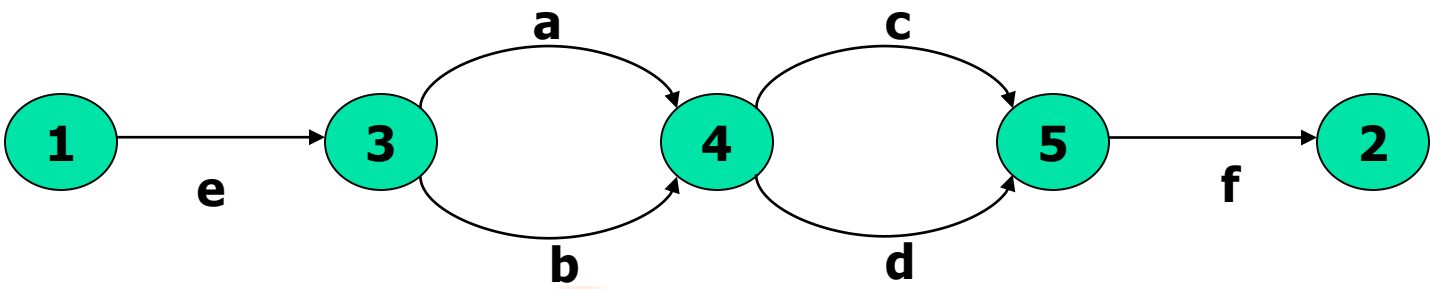
Path products

- Normally flow graphs used to denote only control flow connectivity.
- The simplest weight we can give to a link is a name.
- Using link names as weights, we then convert the graphical flow graph into an equivalent algebraic like expressions which denotes the set of all possible paths from entry to exit for the flow graph.

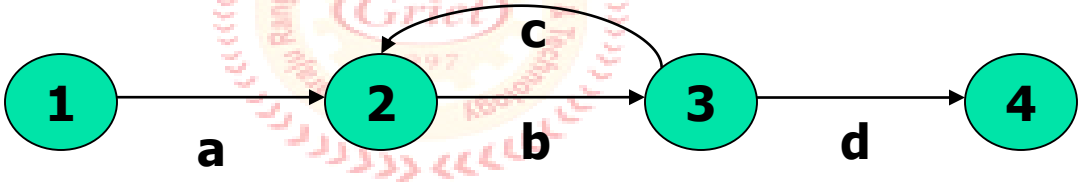
Path products -continued.,

- Every link of a graph can be given a name.
- The link name will be denoted by lower case italic letters.
- In tracing a path or path segment through a flow graph, you traverse a succession of link names.
- The name of the path or path segment that corresponds to those links is expressed naturally by concatenating those link names.
- For example, if you traverse links a,b,c and d along some path, the name for that path segment is abcd.
- This path name is also called a path product.

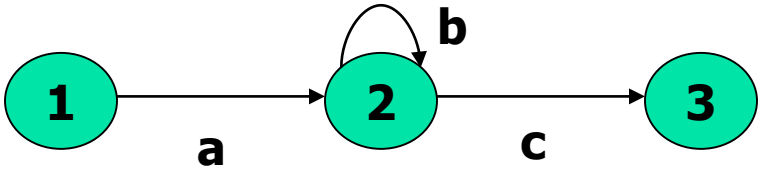
Examples of Paths



(eacf, eadf, ebcf, ebdf)



(abd, abcbd, abcbcbd, abcbcbcbd)



(ac, abc, abbc, abbbc,abbbbc)

Path Expression

- Consider a pair of nodes in a graph and the set of paths between those node. Denote that set of paths by Upper case letter such as X,Y. the members of the path set can be listed as follows:

  ac, abc, abbc, abbbc, abbbbc……

Alternatively, the same set of paths can be denoted by : ac+abc+abbc+abbbc+abbbbc+……

- The + sign is understood to mean "or" between the two nodes of interest, paths ac, or abc, or abbc, and so on can be taken.
- Any expression that consists of path names and "OR"s and which denotes a set of paths between two nodes is called a "PATH EXPRESSION"

Path Products

- The name of a path that consists of two successive path segments is conveniently expressed by the concatenation or PATH PRODUCT of the segment names.
- For example, if X and Y are defined as X=abcde,Y=fghij,then the path corresponding to X followed by Y is denoted by

  XY=abcdefghij

Similarly, YX=fghijabcde

aX=aabcde

Xa=abcdea

XaX=abcdeaabcde

Path Products-continued.,

- If X and Y represents sets of paths or path expressions, their product represents the set of paths that can be obtained by following every element of X by any element of Y in all possible ways.
- For example, X=abc+def+ghi

  Y =uvw+z
- then XY=abcuvw+defuvw+ghiuvw+abcz+defz+ghiz
- If a link or segment name is repeated, that fact is denoted by an exponent. The exponent value denotes the number of repetitions: $a^1=a;a^2=aa;a^3=aaa;....a^n=aaaaaaa.....$(n times)

Path Products-continued.,

- Similarly, if X=abcde then

  x^1=abcde

  x^2=abcdeabcde=$(abcde)^2$

  x^3=abcdeabcdeabcde=$(abcde)^3$
- The path product is not commutative (that is XY!=YX).
- The path product is Associative.
- RULE 1: A(BC)=(AB)C=ABC

  where A,B,C are path names, set of path names or path expressions.
- The zeroth power of a link name, path product, or path expression is also needed for completeness. It is denoted by the numeral "1" and denotes the path whose length is zero- that is the path does not have any links.

  $a^0=1$

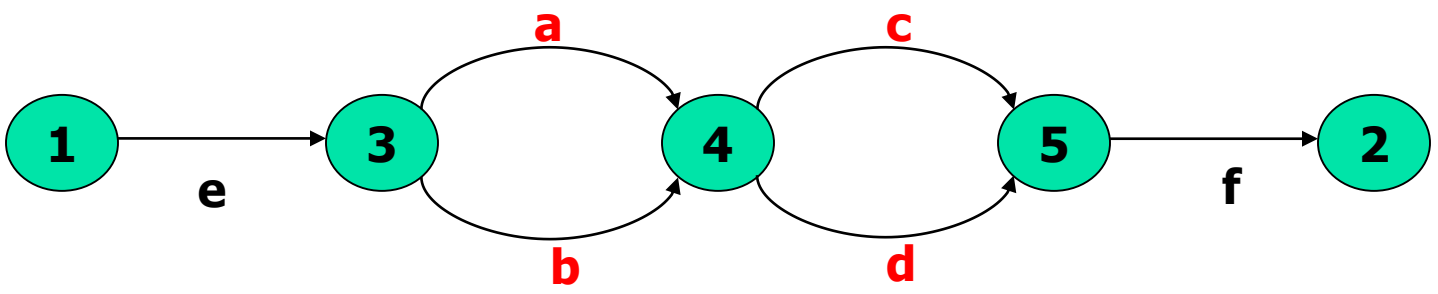
  $X^0=1$

Path Sums

- The "+" sign was used to denote the fact that path names were part of the same set of paths.
- The "PATH SUM" denotes paths in parallel between nodes.

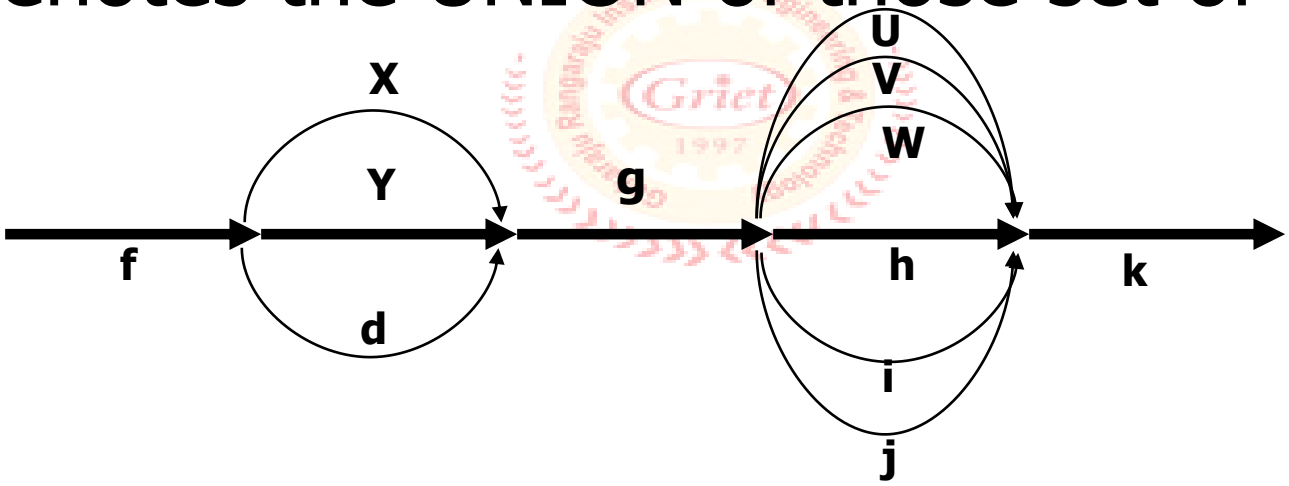
Path Sums- continued.,

- For example, in the below graph links a and b are parallel and are denoted by a+b. similarly c and d are parallel and dented by c+d.
- The set of all paths between nodes 1 and 2 can be thought of as a set of parallel paths and denoted by **eacf+eadf+ebcf+ebdf**

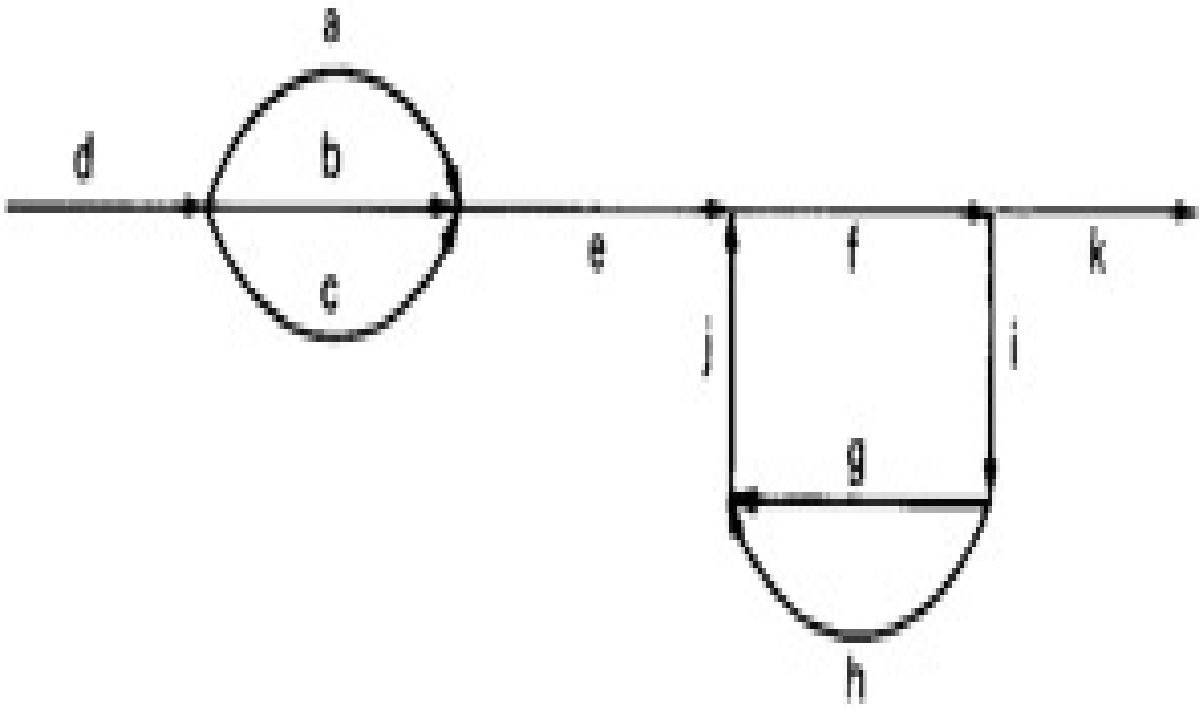


Path Sums- continued.,

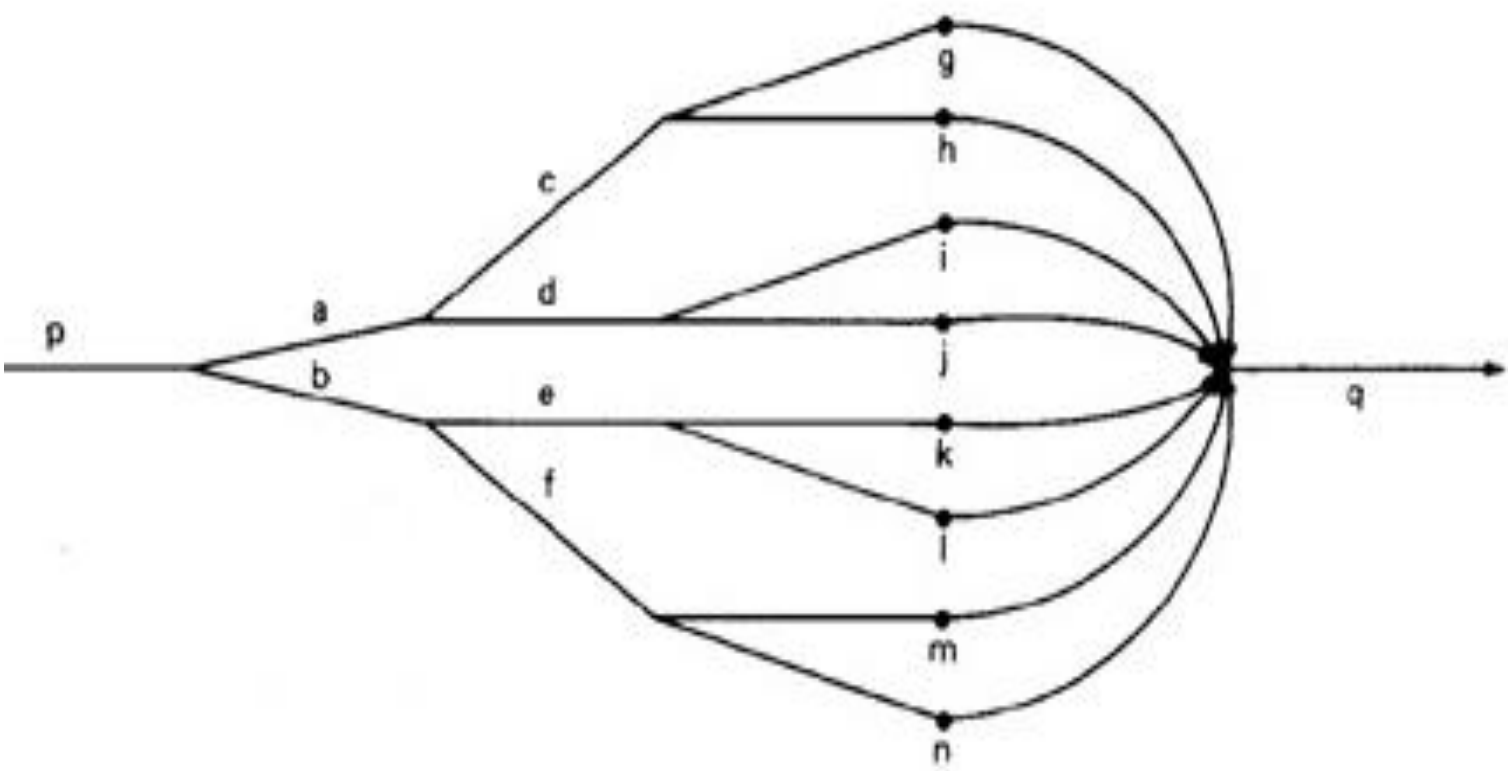
- If X and Y are sets of paths that lie between the same pair of nodes, then X+Y denotes the UNION of those set of paths.

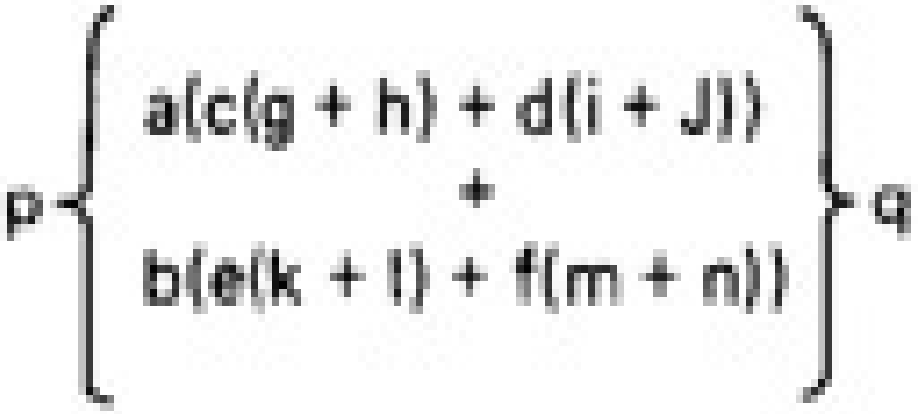


The first set of parallel paths is denoted by X+Y+d and the second set by U+V+W+h+i+j. The set of all paths in the flow graph is f(X+Y+d)g(U+V+W+h+i+j)k


a
d
b
c
e
f
k
j
i
g
h


d(a+b+c)ef(i(g+h)jf)*k


p
a
b
c
d
e
f
g
h
i
j
k
l
m
n
q


p
a(c(g + h) + d(i + J))
+
b(e(k + l) + f(m + n))
q

Path Sums- continued.,

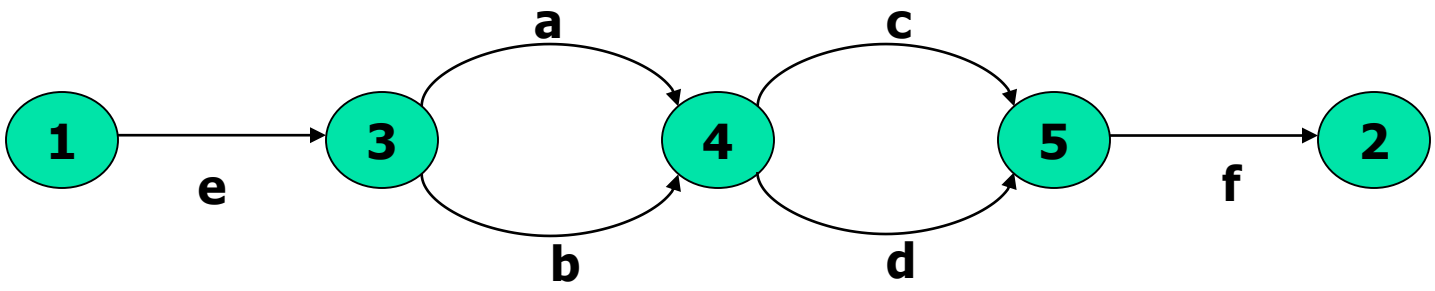
- The path sum is a set union operation, it is clearly Commutative and Associative.
- RULE 2: X+Y=Y+X
- RULE 3: (X+Y)+Z=X+(Y+Z)=X+Y+Z

Distributive Laws

- The product and sum operations are distributive, and the ordinary rules of multiplication apply; that is
- RULE 4: A(B+C)=AB+AC and (B+C)D=BD+CD
- Applying these rules to the below figure yields

e(a+b)(c+d)f=e(ac+ad+bc+bd)f

=eacf+eadf+ebcf+ebdf



Absorption Rule

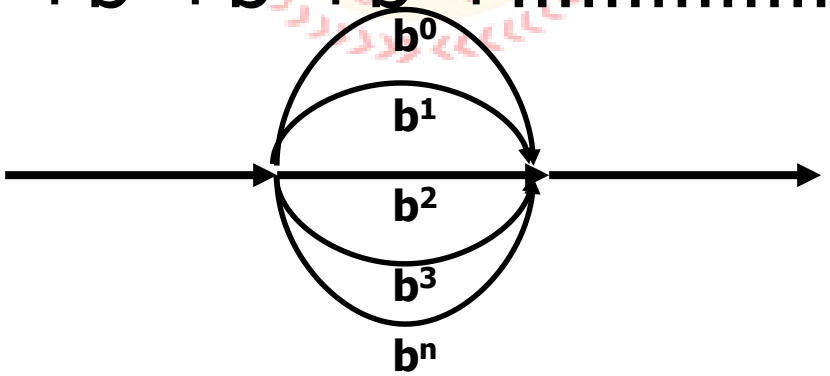
- If X and Y denote the same set of paths, then the union of these sets is unchanged; consequently,
- RULE 5: X+X=X (Absorption Rule)
- If a set consists of paths names and a member of that set is added to it, the "new" name, which is already in that set of names, contributes nothing and can be ignored.
- For example, if X=a+aa+abc+abcd+def then

  X+a= X+aa= X+abc+ X+abcd= X+def= X

  It follows that any arbitrary sum of identical path expressions reduces to the same path expression.

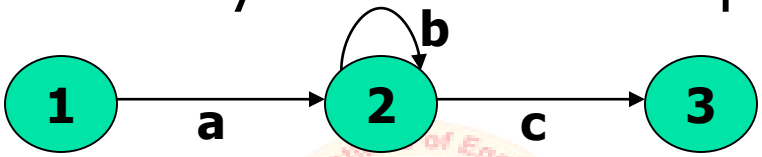
Loop

- Loops can be understood as an infinite set of parallel paths. Say that the loop consists of a single link b. then the set of all paths through that loop point is $b^0+b^1+b^2+b^3+b^4+b^5+\ldots\ldots\ldots\ldots$



Loops – continued.,

- This potentially infinite sum is denoted by b* for an individual link and by X* when X is a path expression.



- The path expression for the above figure is denoted by the notation: ab*c=ac+abc+abbc+abbbc+.....
- Evidently $aa^*=a^*a=a^+$ and $XX^*=X^*X=X^+$
- It is more convenient to denote the fat that a loop cannot be taken more than a certain, say n, number of times. A bar is used under the exponent to denote the fact as follows:

  $X^{\underline{n}}= X^0+X^1+X^2+X^3+X^4+X^5+.....+X^n$

- The following rules can be derived from the previous rules:
- RULE 6: $X^n + X^m = X^n$ if n>m

  X^m if m>n
- RULE 7: $X^n X^m = X^{n+m}$
- RULE 8: $X^n X^* = X^* X^n = X^*$
- RULE 9: $X^n X^+ = X^+ X^n = X^+$
- RULE 10: $X^* X^+ = X^+ X^* = X^+$

Identity Elements

- RULE 11: 1+1=1
- RULE 12: 1X=X1=X following or preceding a set of paths by a path of zero length does not change the set.
- RULE 13: $1^n=1^{\underline{n}}=1^*= 1^+=1$ no matter how often you traverse a path of zero length. It is a path of zero length.
- RULE 14: $1^++1=1^*=1$
- The null set of paths is denoted by the numeral 0. it obeys the following rules:
- RULE 15: X+0=0+X=X
- RULE 16: 0X=X0=0 If you block the paths of a graph for or aft by a graph that has no paths , there wont be any paths.

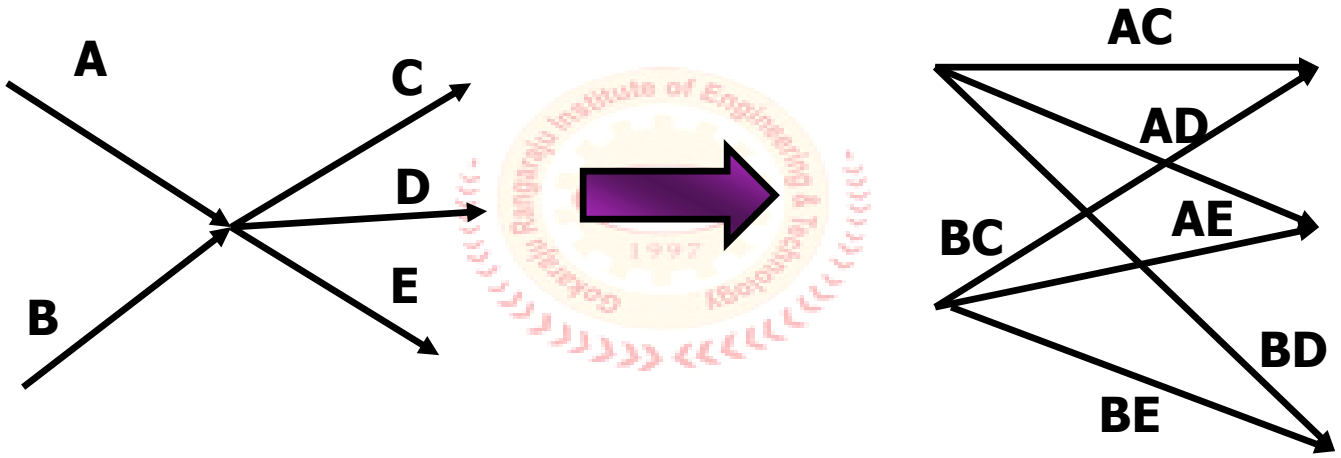
A Reduction Procedure

- This procedure is a node by node removal algorithm which is used to convert a flowgraph whose links are labeled with names into a path expression that denotes the set of all entry/exit paths in that flow graph.
- Steps to initialize the process:
  1. Combine all serial links by multiplying their path expressions.
  2. Combine all parallel links by adding their path expressions.
  3. Move all self-loops by replacing them with a link of the form X*, where X is the path expression of the link in that loop.
- Steps in the algorithm's loop:
  4. Select any node for removal other than the initial or final node. Replace it with a set of equivalent links whose path expressions corresponds to all the ways you can form a product of the set of inlinks with the set of outlinks of that node.
  5. Combine any remaining serial links by multiplying their path expressions.
  6. combine all parallel links by adding their path expressions.
  7. Remove all self-loops as in step-3.
  8. Does the graph consists of a single link between the entry node and the exit node? If yes then the expression for that link is a path expression for the original flow graph; otherwise return to step-4.

Cross – Term Step (Step 4)

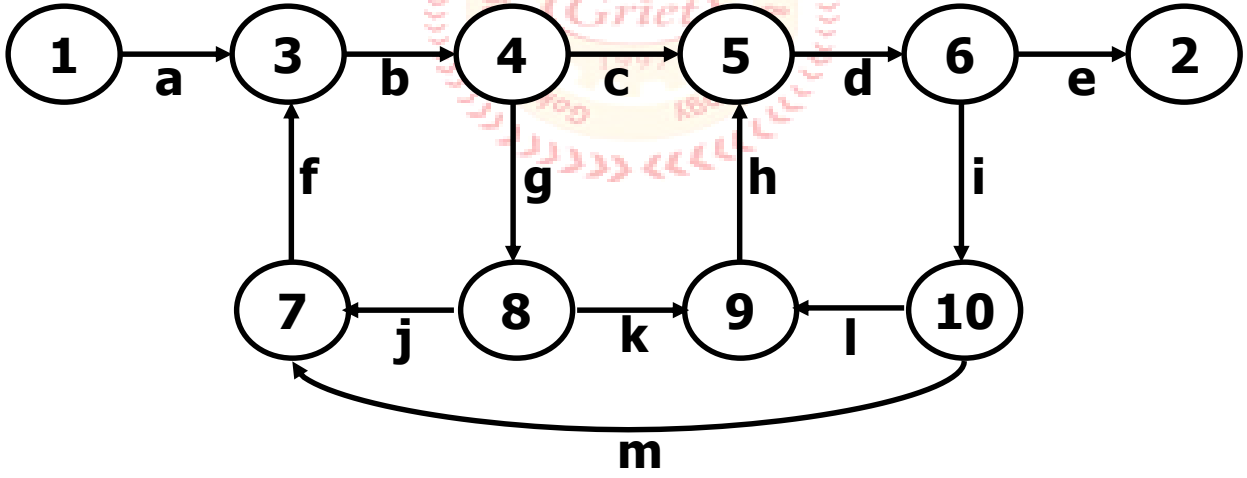
- The cross term step is the fundamental step of the reduction algorithm.
- It removes a node, there by reducing the number of nodes by one.
- Successive application of this step eventually get you down to one entry and one exit node.

Cross – Term Step (Step 4)- Example

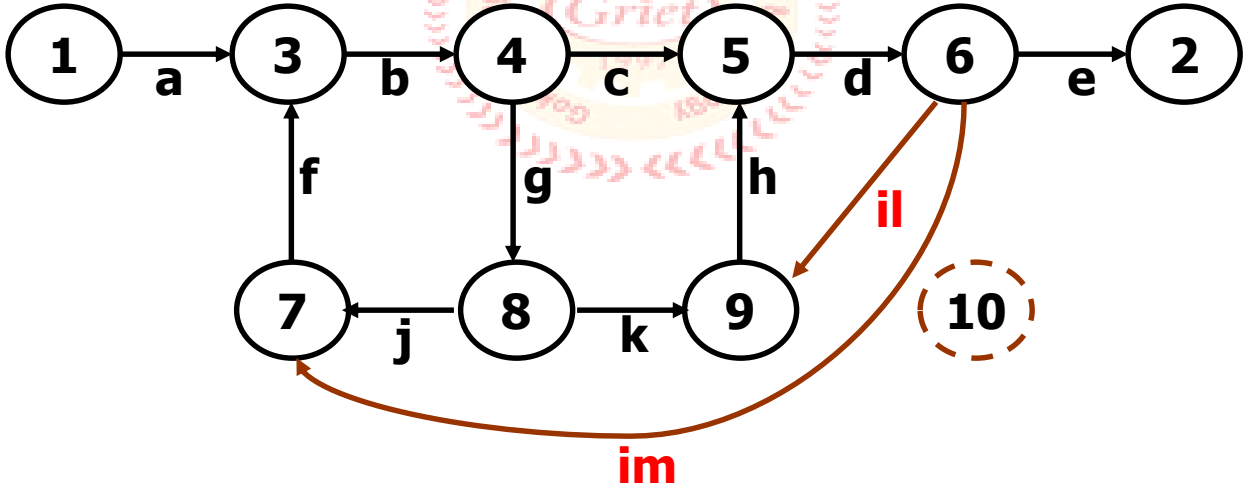


A Reduction Procedure-Example

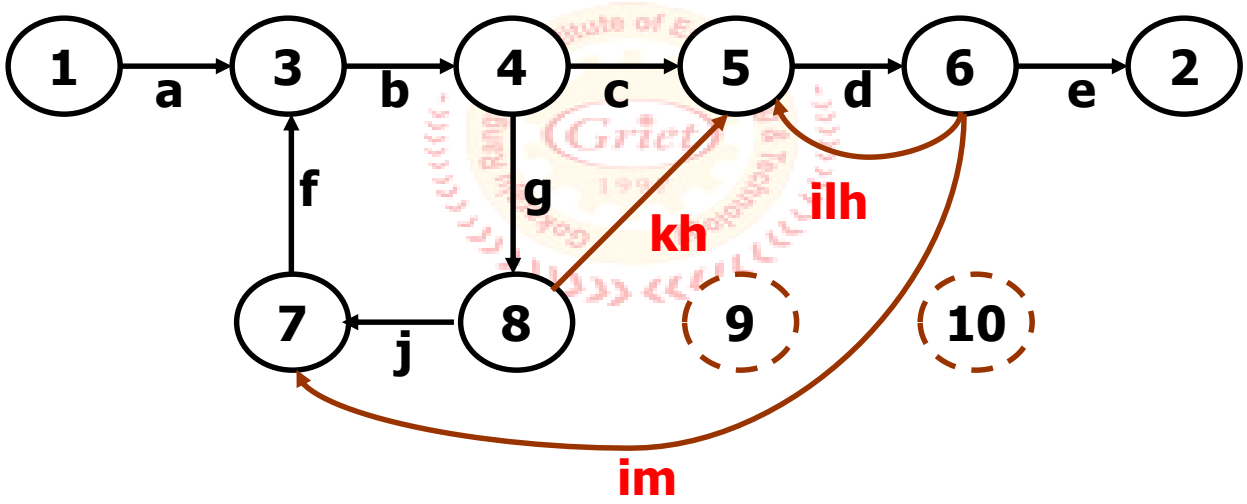
- Applying this algorithm to the following graph, we remove several nodes in order; that is



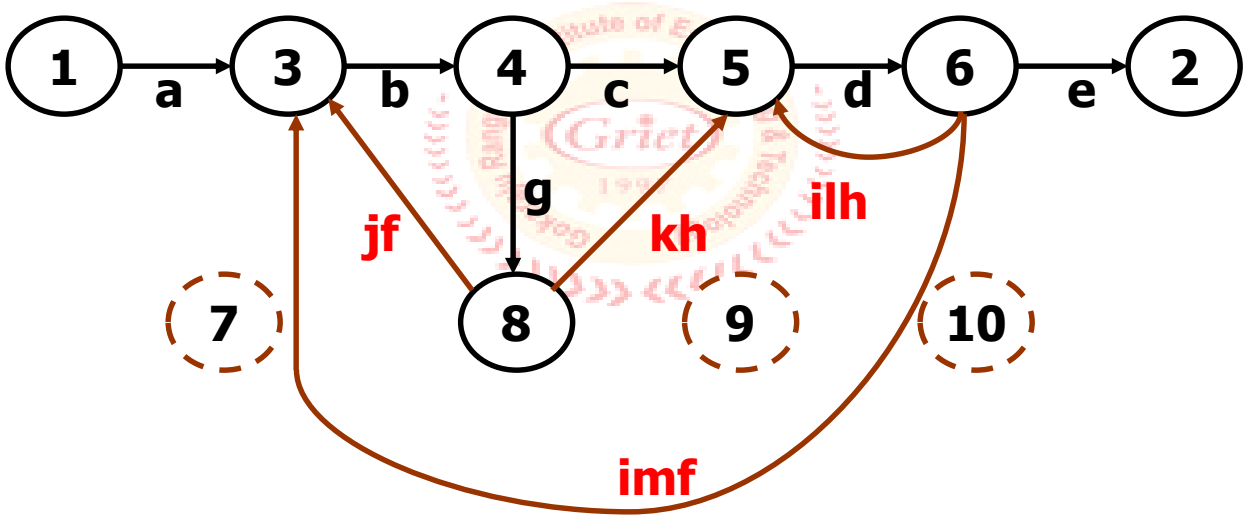
- Remove node 10 by applying step 4 and combine by step 5 to yield



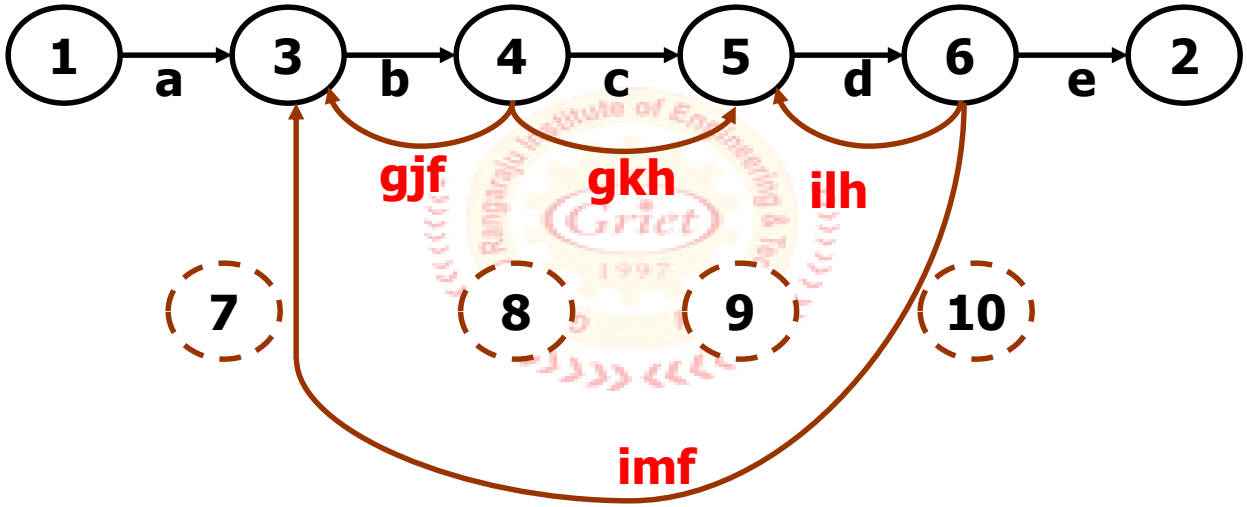
- Remove node 9 by applying step4 and 5 to yield



- Remove node 7 by steps 4 and 5, as follows:

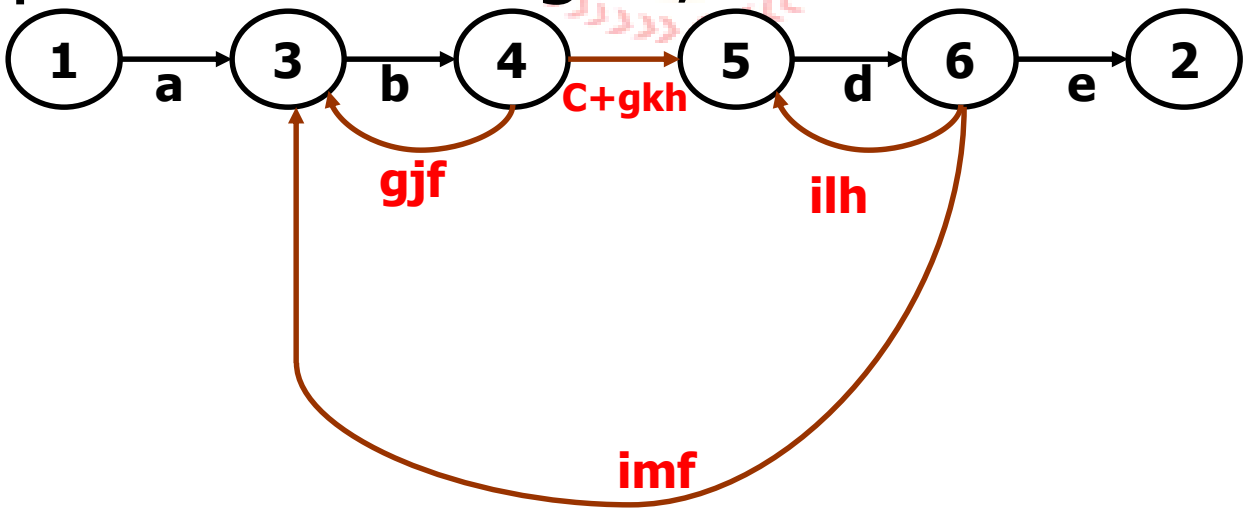


- Remove node 8 by steps 4 and 5, to obtain



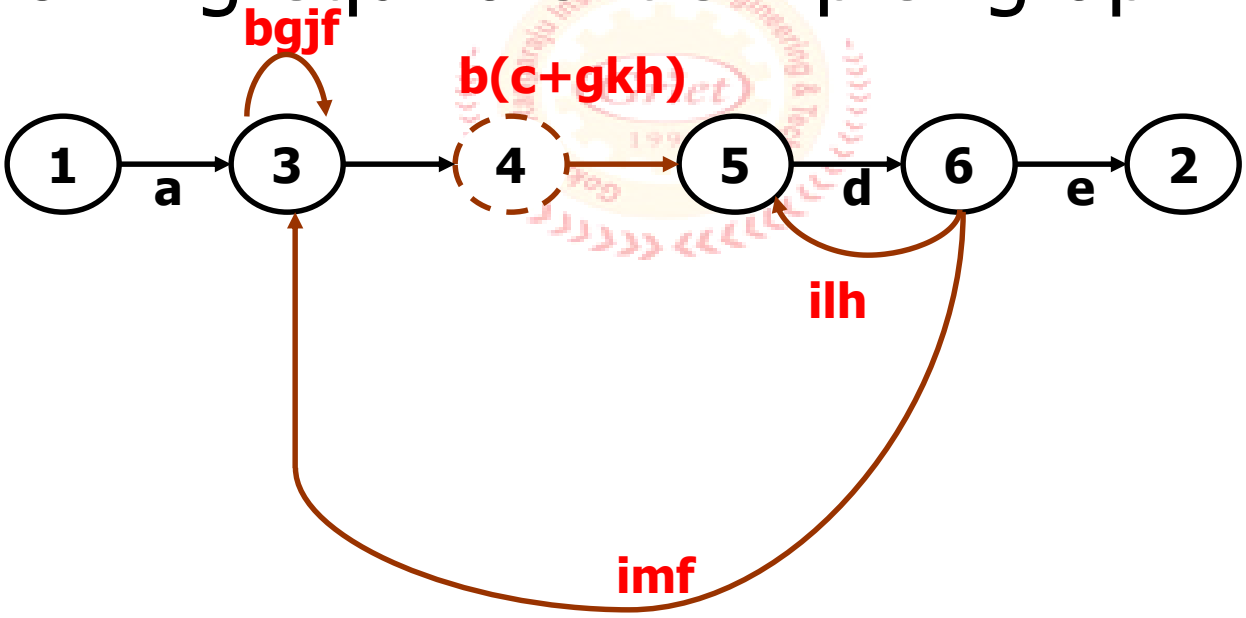
Parallel Term (step 6)

- Removal of node 8 above led to a pair of parallel links between nodes 4 and 5. combine them to create a path expression for an equivalent link whose path expression is c+gkh; that is

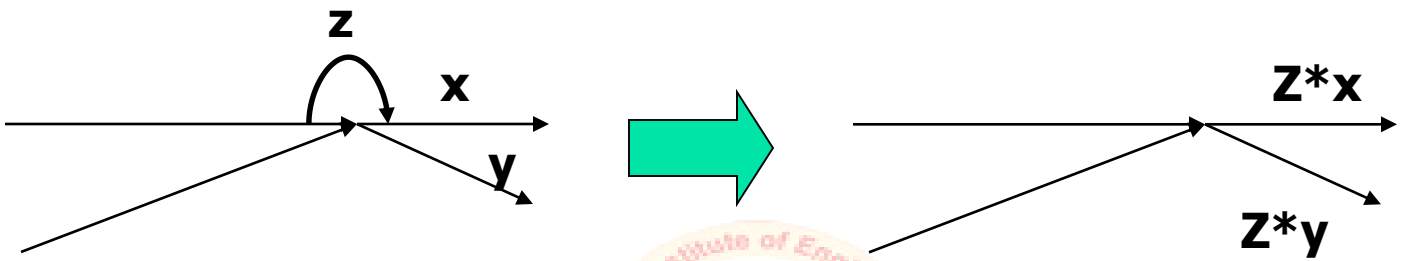


Loop Term (step 7)

- Removing node 4 leads to a loop term. The graph has now been replaced with the following equivalent simpler graph:

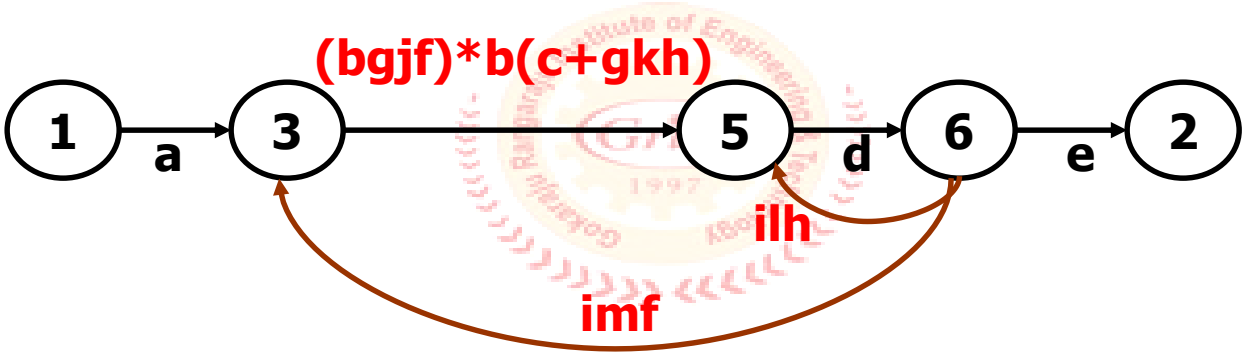


Loop-removal operations

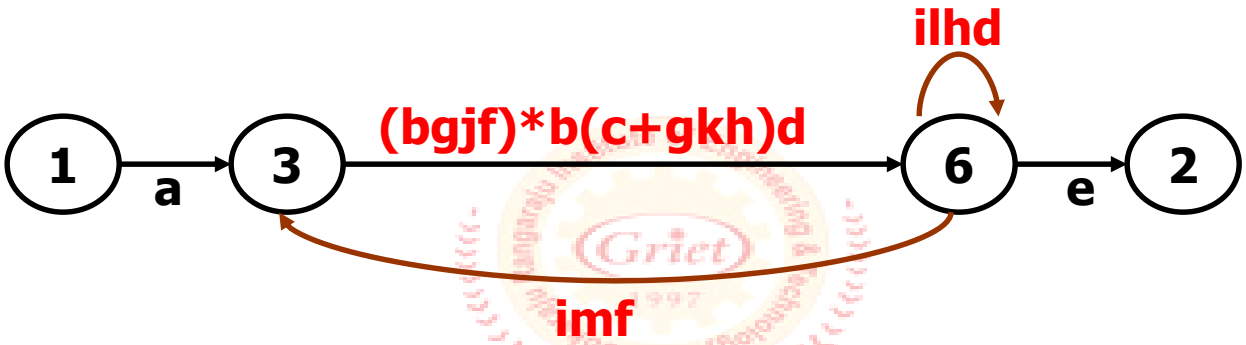


In this way, we remove the self-loop and then multiply all outgoing links by z*

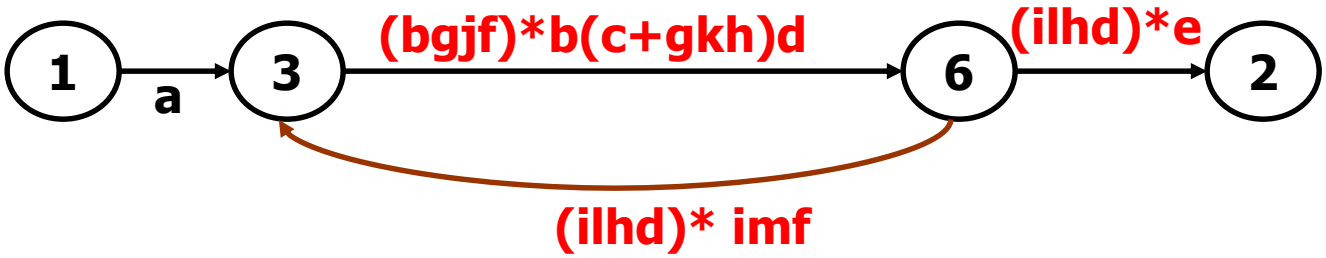
- Continue the process by applying the loop-removal step as follows:



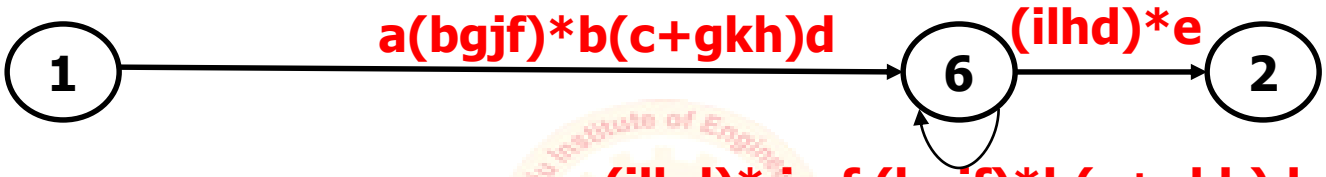
- Removing node 5 produces



Remove the loop at node 6 to yield



- Remove node 3 to yield



Removing the loop and then node 6 result in the following expression:

a(bgjf)*b(c+gkh)d((ilhd)*imf(bjgf)*b(c+gkh)d)*(ilhd)*e

Applications

- The purpose of the node removal algorithm is to present one very generalized concept- the path expression and way of getting it.
- Every application follows this common pattern:
- Convert the program or graph into a path expression.
- Identify a property of interest and derive an appropriate set of arithmetic rules that characterizes the property.
- Replace the link names by link weights for the property of interest. Now the path expression been converted into some algebra like ordinary algebra or regular expression.
- Simplify or evaluate the resulting algebraic expression to answer the question you asked.

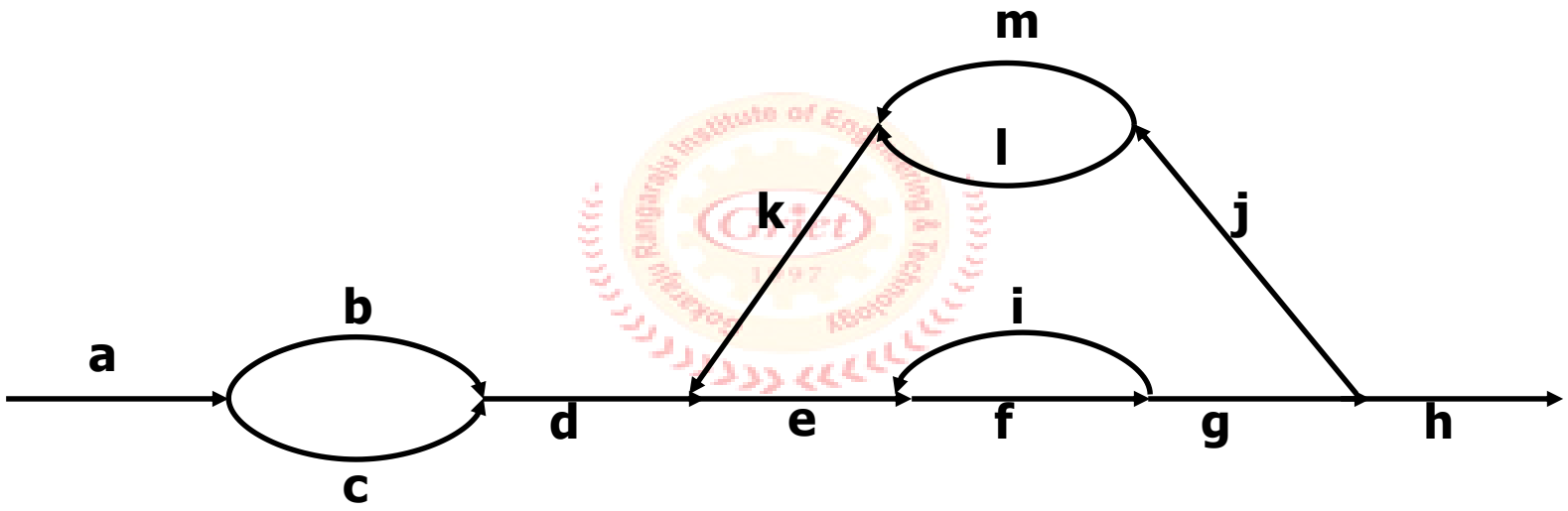
Maximum Path Count Arithmetic

- Label each link with a link weight that corresponds to the number of paths that link represents.
- Also mark each loop with the maximum number of times that loop can be taken. If the answer is infinite, you might as well stop the analysis because it is clear that the maximum number of paths will be infinite.
- There are three cases of interest: parallel links, serial links, and loops

Case	Path expression	Weight expression
Parallels	A+B	W_A+W_B
Series	AB	$W_A W_B$
Loop	A^n	$\sum_{J=0}^{n} W_A^j$

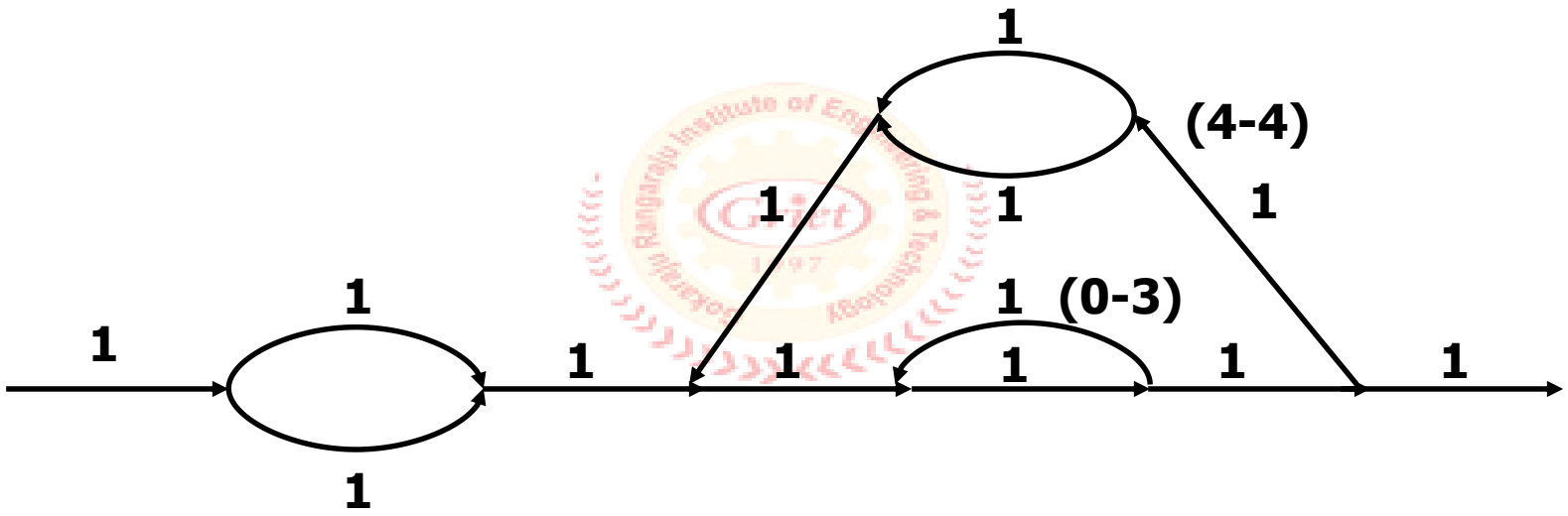
This arithmetic is an ordinary algebra. The weight is the number of paths in each set.

Maximum Path Count Arithmetic-example

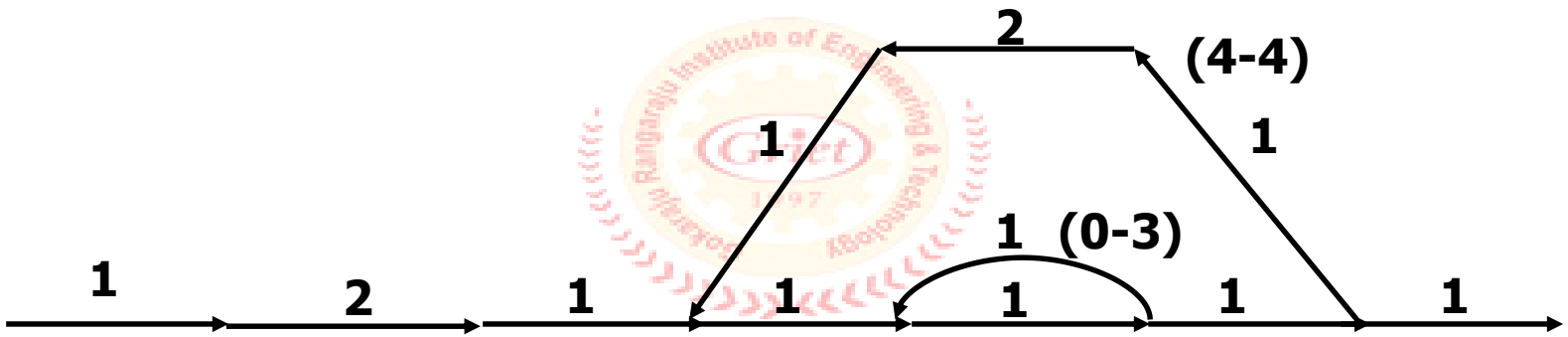


Each link represents a single link and consequently is given a weight of "1"To start. Lets say the outer loop will be taken exactly four times and inner Loop Can be taken zero or three times

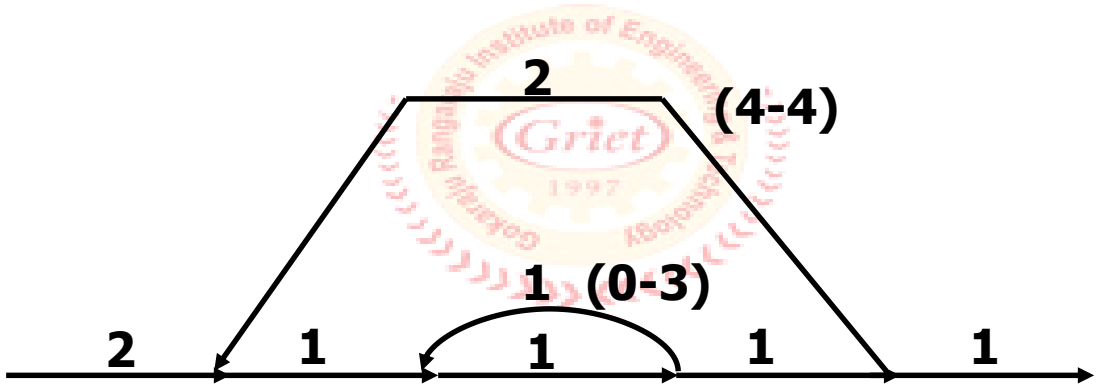
- Path expression: a(b+c)d{e(fi)*fgj(m+l)k}*e(fi)*fgh



Annotated the flow graph by replacing the link name with the maximum of paths through that link(1) also noted the number of times for looping.

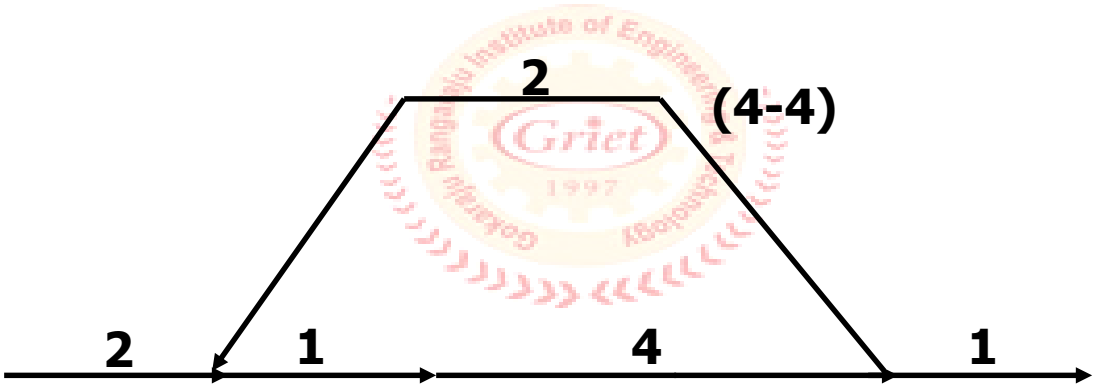


Combined the first pair of parallel loops outside the loop and also the pair in the outer loop

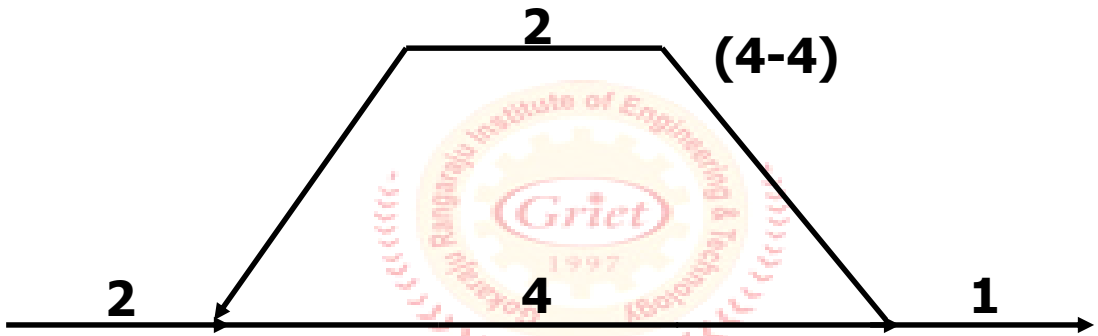


Multiplied the things out and removed nodes to clear the clutter

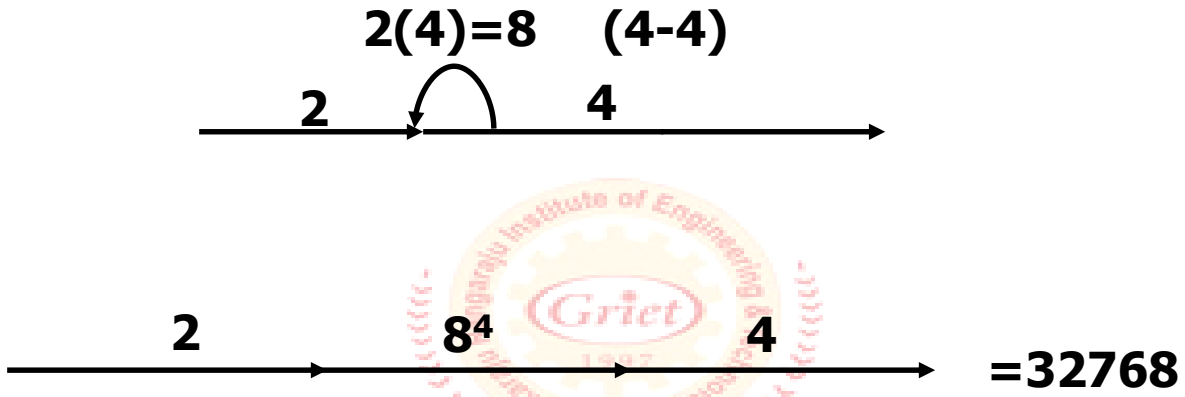
- For the inner loop,



Take care of the inner loop, there are four possibilities leading to four values. Then multiplied by the following link weight.



- Using cross term to create the self loop with a weight.



Alternatively, you could have substituted a "1" for each link in the path expression and then simplified, as follows:

$$1(1+1)1(1(1*1)^{\underline{3}}1*1*1(1+1)1)^{\underline{4}}1(1*1)^{\underline{3}}1*1*1$$

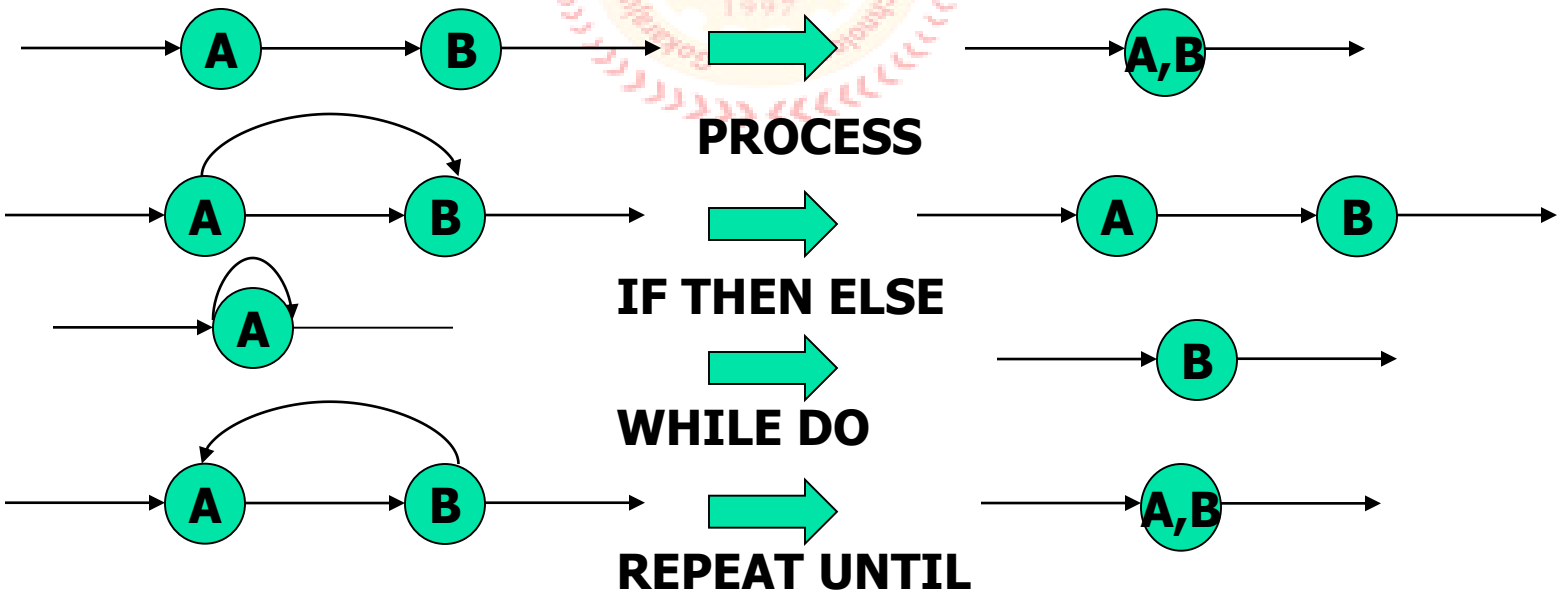
$$=2(1^{\underline{3}}1*(2))^{\underline{4}}1^{\underline{3}}$$

but $1^{\underline{3}}=1+1^1+1^2+1^3=4$

$$=2(4*2^4)*4=2*8^4*4=32768$$

Structured Flowgraph

- A structured flowgraph is one that can be reduced to a single link by successive application of the transformation of the following:

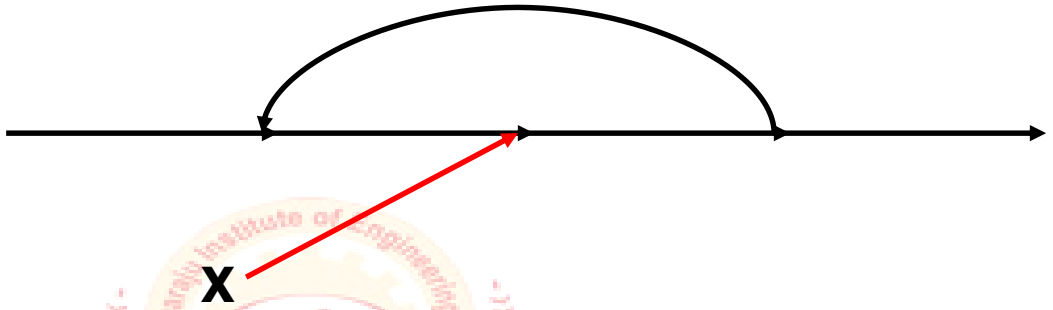


Structured Flowgraph- continued.,

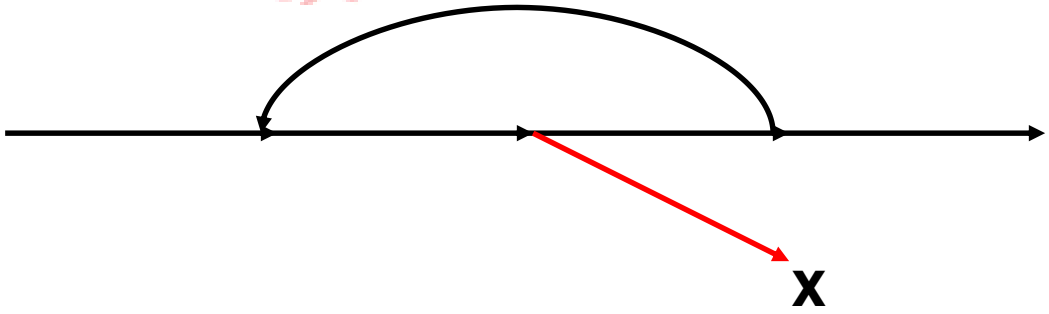
- Flow graphs that do not contain one or more of the graphs shown below as sub graphs are structured.
  - Jumping into loops
  - Jumping out of loops
  - Branching into decisions
  - Branching out of decisions

Unstructured Sub Graphs

Jumping into loops

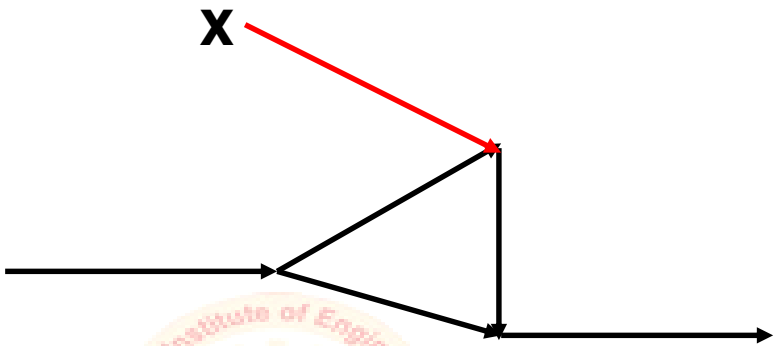


Jumping out of loops

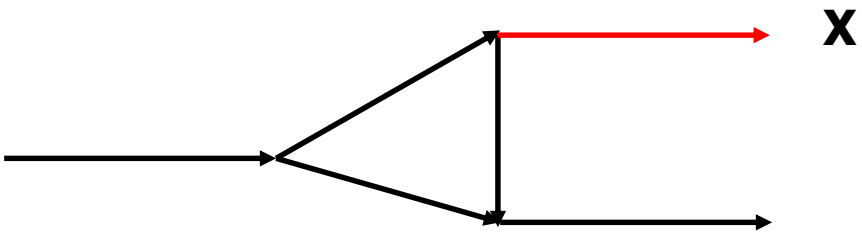


Unstructured Sub Graphs

Branching into decisions



Branching out of decisions

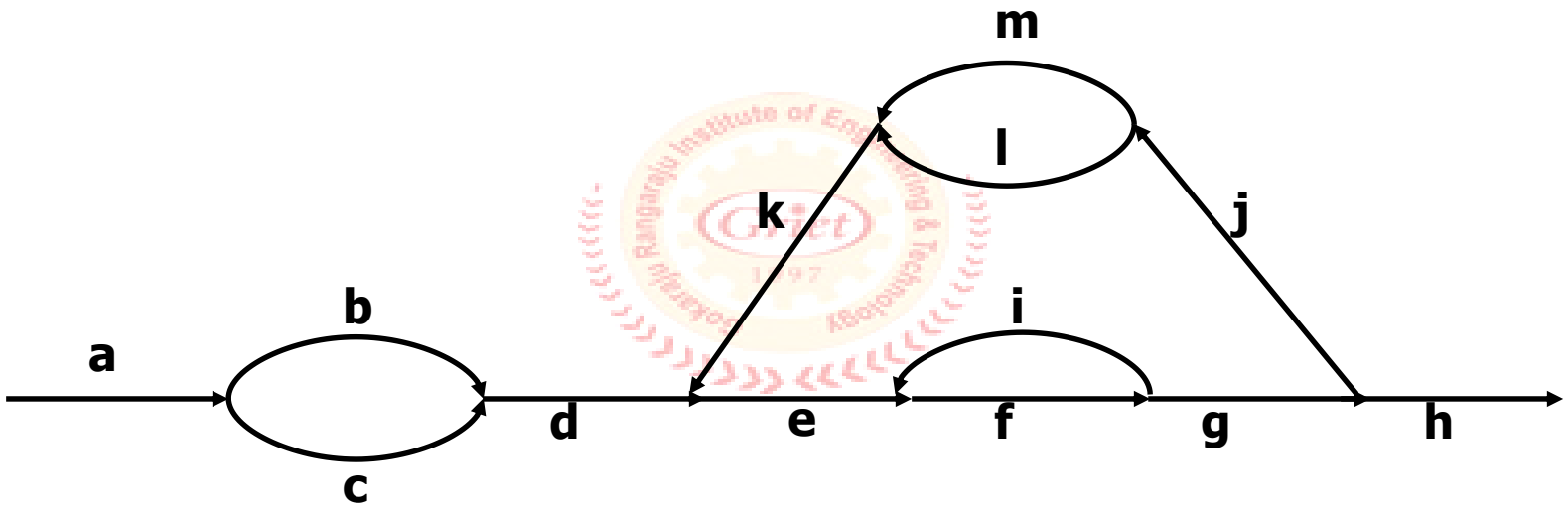


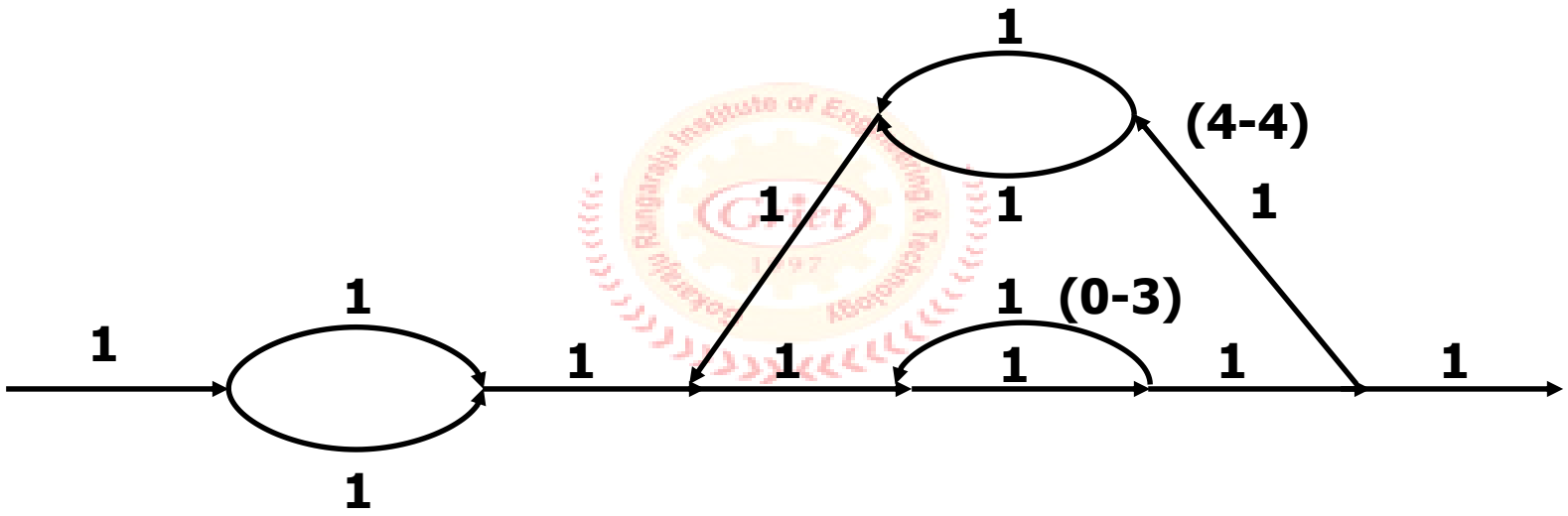
Lower Path Count Arithmetic

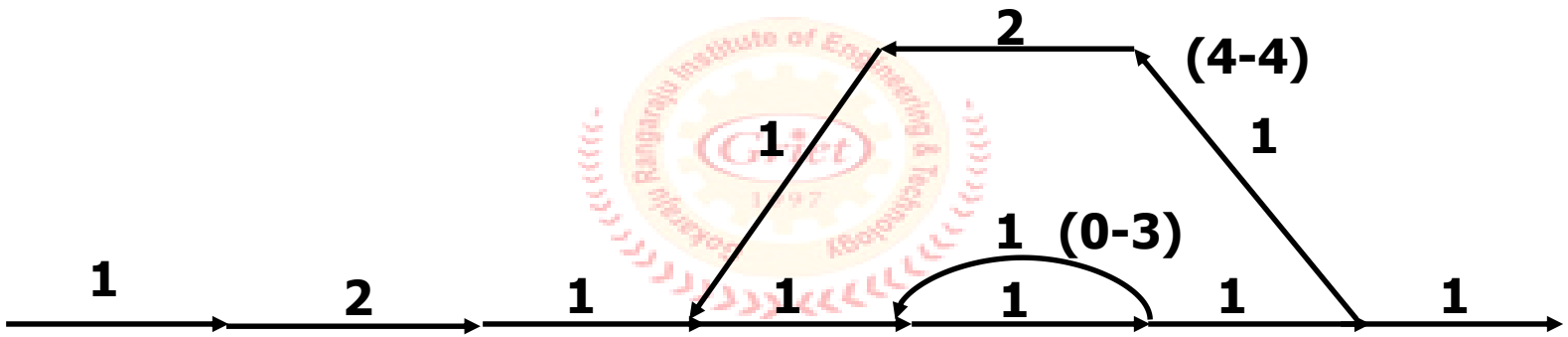
- A lower bound on the number of paths in a routine can be approximated for structured flow graphs.
- The arithmetic is as follows:
- The values of the weights are the number of members in a set of paths.

Case	Path expression	Weight expression
Parallels	A+B	W_A+W_B
Series	AB	max(W_A W_B)
Loop	$A^{\underline{n}}$	1, W_1

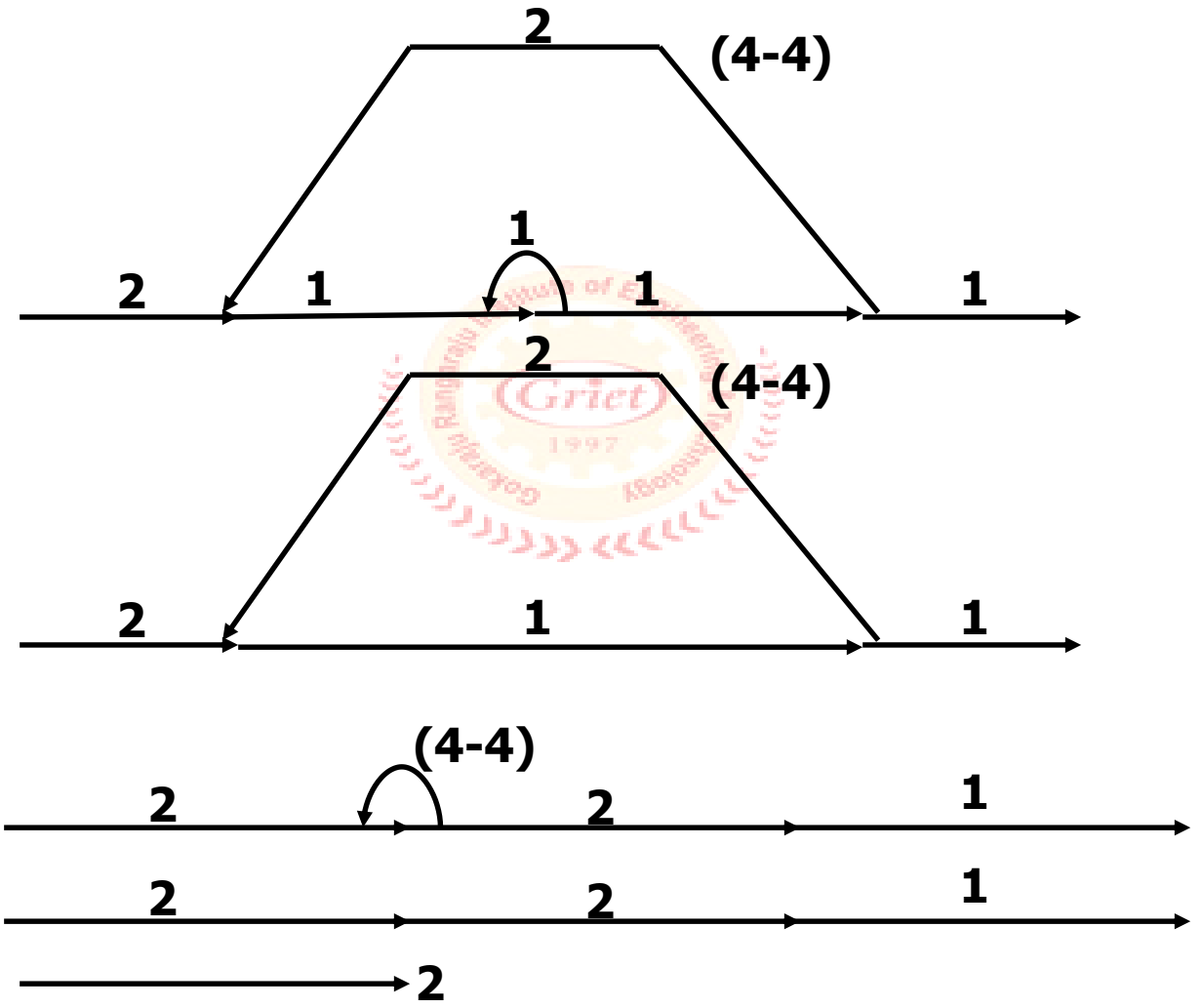
Minimum Path Count-example.,




1
(4-4)
1
1
1
1
1
(0-3)
1
1
1
1
1
1
1
1
1
1



Combined the first pair of parallel loops outside the loop and also the pair in the outer loop


2
(4-4)
1
2
1
1
1
2
(4-4)
2
1
1
(4-4)
2
2
1
2
2
1
2

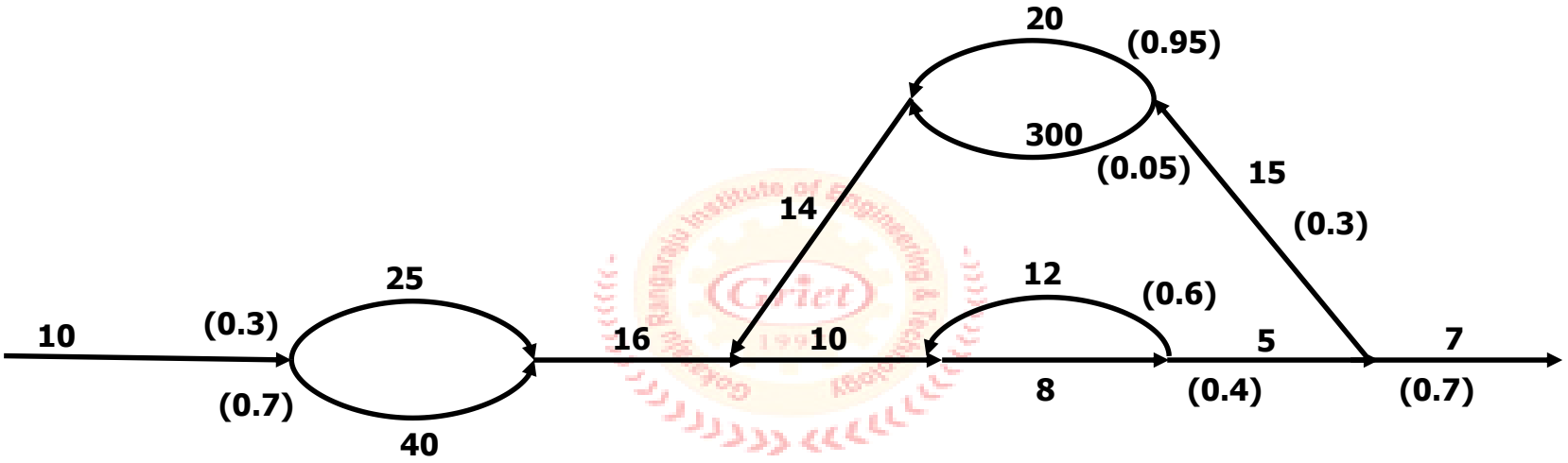
Mean Processing times of Routines

- Given the execution time of all statements or instructions for every link in a flowgraph and the probability for each direction for all decisions are to find the mean processing time for the routine as a whole.
- The model has two weights associated with every link: the processing time for that link, denoted by T, and the probability of that link P.

The rules for mean processing times are:

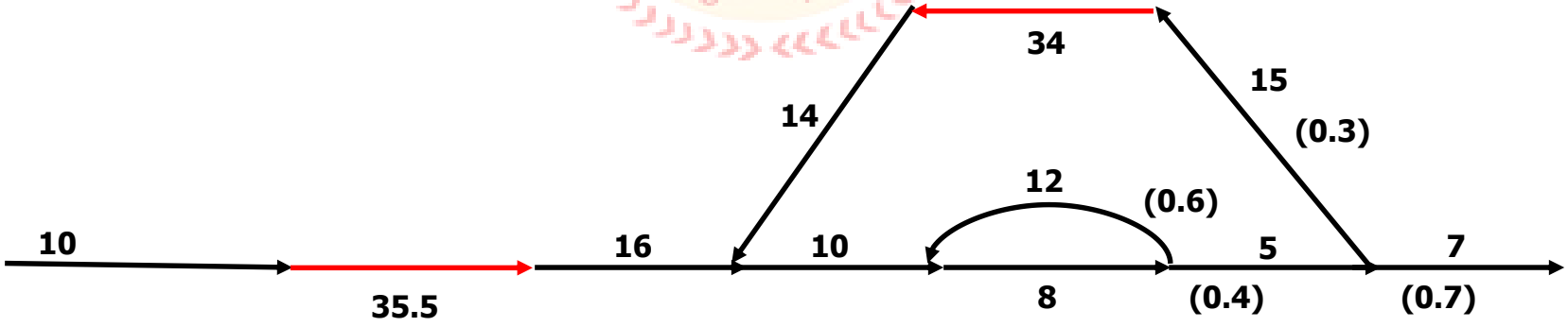
Case	Path expression	Weight expression
Parallels	A+B	$T_{A+B}=(P_AT_A+P_BT_B)/(P_A+P_B)$ $P_{A+B}= P_A+P_B$
Series	AB	$T_{AB}=T_A+T_B$ $P_{AB} =P_AP_B$
Loop	$A^{\underline{n}}$	$T_A=T_LP_L/(1-P_L)$ $P_A =P_A/(1-P_L)$

Example

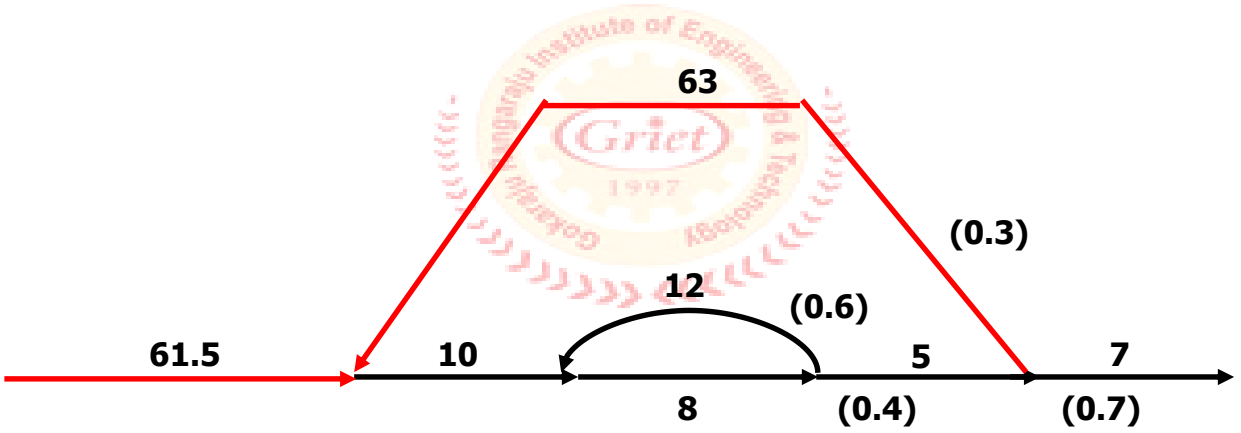


- Start with the original flow graph annotated with probabilities and processing time.

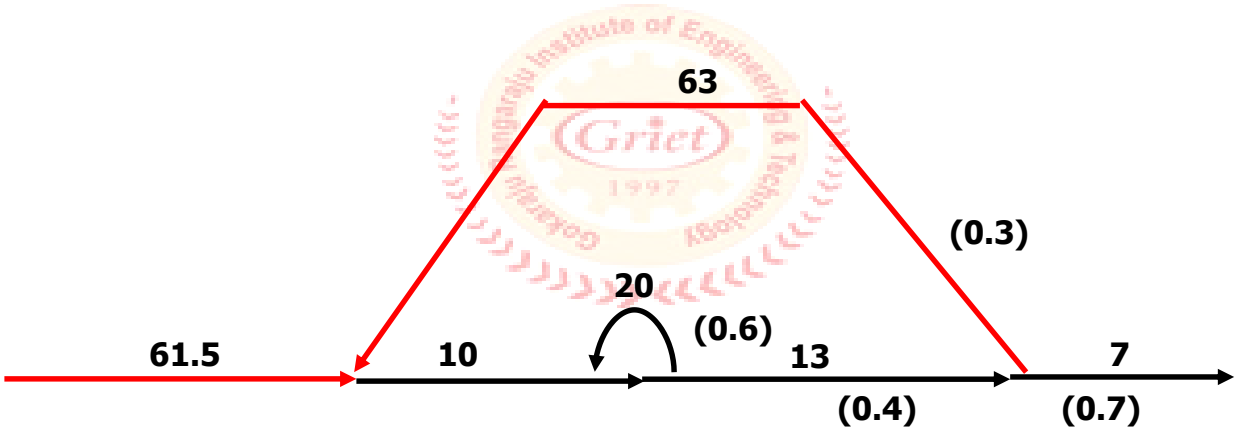
- Combine the parallel links of the outer loop. The result is just the mean of the processing times for the links because there aren't any other links leaving the first node. Also combine the pair of links at the beginning of the flow graph.

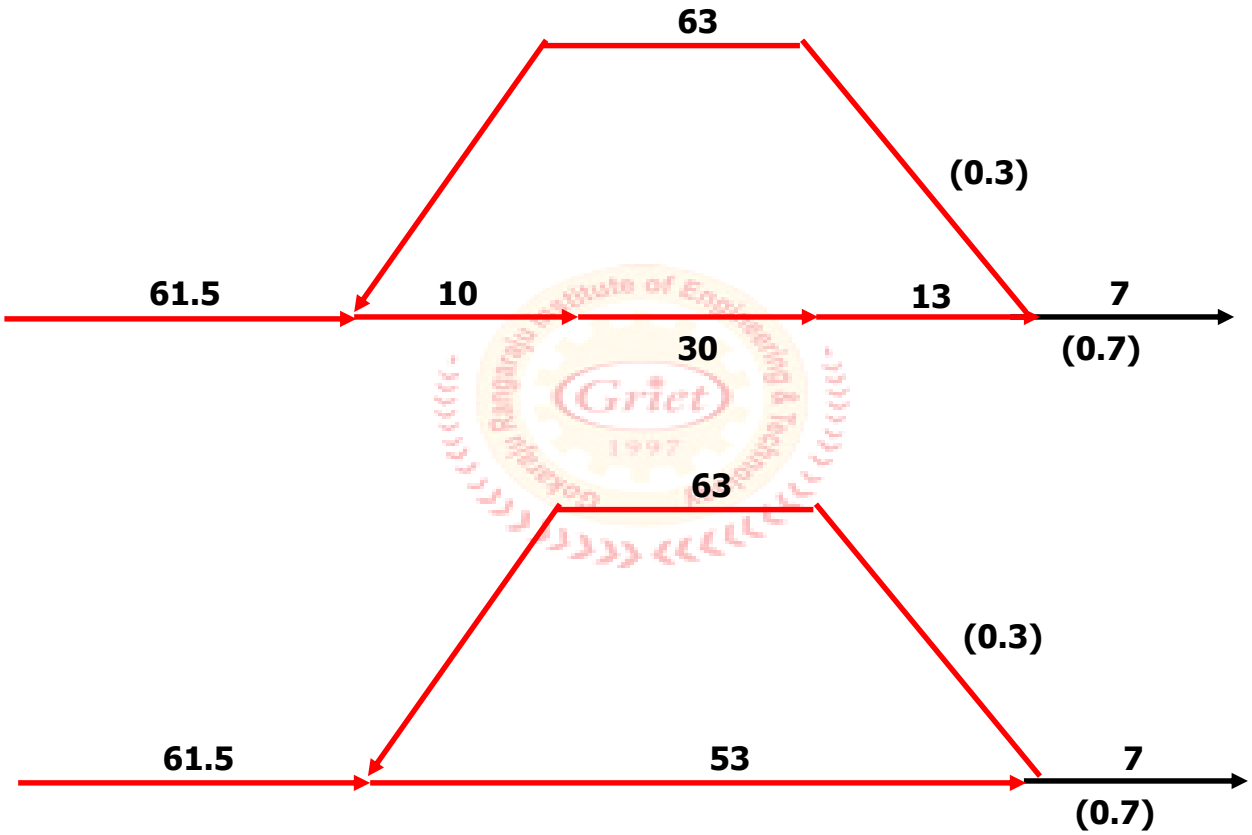


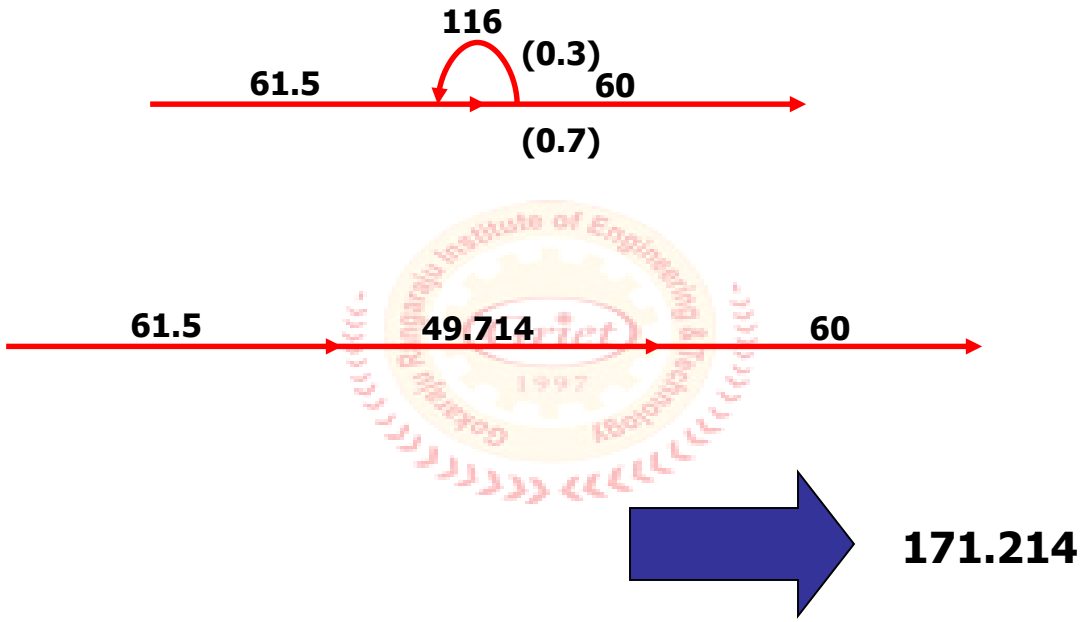
- Combine as many as serial links as you can



- Use the cross term step to eliminate a node and to create the inner self loop.




63
(0.3)
61.5
10
13
7
30
(0.7)
63
(0.3)
61.5
53
7
(0.7)


116
(0.3)
61.5
60
(0.7)
61.5
49.714
60
171.214

Regular Expressions and Flow Anomaly Detection

- The flow anomaly detection problem is that of looking for a specific sequence of operations considering all possible paths through a routine.
- Here we are interested in knowing whether a specific sequence occurred but not what the net effect of the routine is.
- The method of anomaly detection:
  - Annotate each link in the graph with the appropriate operator or the null operator (1).
  - Simplify things to the extent possible.
  - After performing the above two steps you obtain a Regular Expression that denotes the possible sequences of operators in that graph.
  - You can now examine that regular expression for the sequence of interest.

Chapter - 10

LOGIC-BASED TESTING

Programmers and Logic

- "Logic" is one of the most often used words In programmers' vocabularies but one of their least used techniques.
- Boolean Algebra is being the simplest form of logic.
- Boolean Algebra is to logic as arithmetic is to mathematics.

Hardware Logic Testing

- Logic has been the primary tool of hardware logic designers.
- Many test methods developed for hardware logic testing are adapted to software logic testing. Because hardware testing automation is 10 to 15 years ahead of software testing automation.
- So hardware testing methods and its associated theory is a fertile ground for software testing methods.
- We can expect the both hardware and software designers meet at a middle ground where Boolean algebra will be a basic to their common language of disclosure.

Specification Systems and Languages

- The trouble with specifications is that they're hard to express.
- Boolean Algebra (Sentential Calculus) is the most basic of all logic systems.
- Higher order logic systems are needed and used for formal specifications.
- These tools incorporate methods to simplify, transform, and check specifications, and the methods are to a large extent based on Boolean algebra.

Knowledge Based Systems

- The knowledge based system(also expert system, or artificial intelligence system) has become the programming construct of choice for many applications that were once considered very difficult.
- Knowledge based systems incorporate knowledge from a knowledge domain such as medicine, law or civil engineering into a data base. The data can be queried and interacted to provide solutions to problems in that domain.
- Other implementation of knowledge based systems is to incorporate the expert's knowledge into a set of rules. The user can then provide data and ask questions based on that data.
- The processing is done by a program called Inference Engine.

Decision Tables

- A decision table is a table that consists of four areas called the condition stub, the condition entry, the action stub and the action entry.
- Each column of the table is a rule that specifies the conditions under which the actions named in the action stub will take place.

An example of a Decision Table

CONDITION ENTRY

	RULE 1	RULE 2	RULE 3	RULE 4
CONDITION 1	YES	YES	NO	NO
CONDITION 2	YES	I	NO	I
CONDITION 3	NO	YES	NO	I
CONDITION 4	NO	YES	NO	YES
ACTION 1	YES	YES	NO	NO
ACTION 2	NO	NO	YES	NO
ACTION 3	NO	NO	NO	YES

ACTION ENTRY

CONDITION ENTRY (OR-OF) ACTION ENTRY (OR-OF)

- The condition stub is a list of names of conditions.
- A rule specifies whether a condition should or should not be met for the rule to be satisfied. "yes" means that the conditions must be met, and "No" means that conditions must not be met, and "I" means that the conditions plays no part in the rule, or it is immaterial to the rule.
- The action stub names the actions the routine will take or initiate if the rule is satisfied.
- If the action entry is "YES" , the action will take place; if "NO" , the action will not take place.
- Some of the rules are not specified by the decision table for which a default action to be taken are called Default Rules.

Decision Table Processors

- Decision tables can be automatically translated into code and as such are a higher order language.
- The decision table translator checks the source decision table for consistency and completeness and fills in any required default rules.
- Decision tables as a source language have the virtue of clarity, direct correspondence to specifications, and maintainability.

Decision tables as a Basis for Test Case Design

- If a specification is given as a decision table, it follows that decision tables should be used for test case design.
- If a program's logic is implemented as a decision table, decision table should also be used as a basis for test design.
- It is not always possible or desirable to implement the program as a decision table because the program's logical behavior is only part of its behavior. The program interfaces with other programs, there are restrictions or the decision table language may not have needed features.
- The use of a decision table model to design tests is warranted when:
- The specification is given as a decision table or can be easily converted into one.
- The order in which the predicates are to be evaluated does not affect interpretation of the rules or the resulting action.
- Once a rule is satisfied and an action is selected, no other rule need be examined.
- If several actions can result from satisfying rule, the order in which the actions are executed doesn't matter.

Expansion of Immaterial Cases

- Improperly specified immaterial entries (I) cause most decision-table contradictions.
- If a condition's truth value is immaterial in a rule, satisfying the rule doesnot depend on the condition. It doesn't mean that the case is impossible.
- For example,
- Rule 1: " if the persons are male and over 30, then they shall receive a 15% raise"
- Rule 2: "but if the persons are female, then they shall receive a 10% raise."
- The above rules state that age is material for a male's raise, but immaterial for determining a female's raise.

Expansion of Immaterial Cases

	RULE 1	RULE 2	RULE 3	RULE 4
CONDITION 1	YES	YES	NO	NO
CONDITION 2	YES	I	NO	I
CONDITION 3	NO	YES	NO	I
CONDITION 4	NO	YES	NO	YES

	Rule 2.1	Rule 2.2	Rule 4.1	Rule 4.2	Rule 4.3	Rule 4.4
CONDITION1	YES	YES	NO	NO	NO	NO
CONDITION2	YES	NO	YES	YES	NO	NO
CONDITION3	YES	YES	YES	NO	NO	YES
CONDITION4	YES	YES	YES	YES	YES	YES

The expansion of an Inconsistent Specification

	Rule 1	Rule 2
Condition 1 Condition 2 Condition 3 Condition 4	Yes I Yes No	Yes No I No
Action 1 Action 2	Yes No	No yes

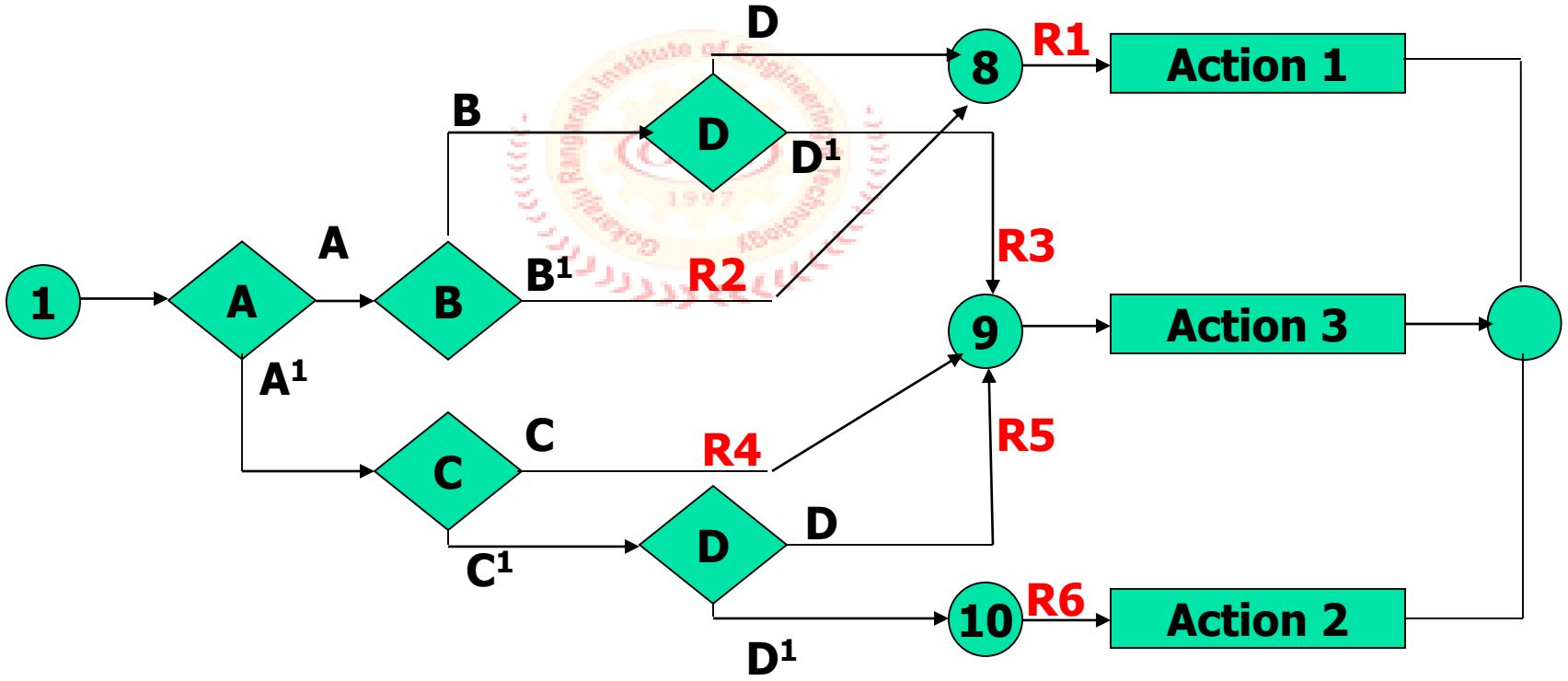
Rule 1.1	Rule 1.2	Rule 2.1	Rule 2.2
Yes Yes Yes No	Yes No Yes No	Yes No Yes No	Yes No No No
Yes No	Yes No	No Yes	No Yes

Test Case Design

- Test case design by decision tables begins with examining the specification's consistency and completeness.
- This is done by expanding all immaterial cases and checking the expanded tables.
- Once the specification have been verified, the objective of the test case is to show that the implementation provides the correct action for all combinations of predicate values.

Decision tables and Structure

- Decision tables can also be used to examine a program's structure.



	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Condition A	Yes	Yes	Yes	No	No	No
Condition B	Yes	No	Yes	I	I	I
Condition C	I	I	I	Yes	No	No
Condition D	Yes	I	No	I	Yes	No
Action 1	Yes	Yes	No	No	No	No
Action 2	No	No	Yes	Yes	Yes	No
Action 3	No	No	No	No	No	Yes

The decision table corresponding to the previous decision tree

Predicates & Relational Operators

- A predicate is implemented as a process whose outcome is a truth functional value.
- Predicates are based on relational operators, of which the arithmetic relational operators are most common.
- A sample of some other relational operators are: is a member of……,is a subset of……..,is a substring of……., is above……, is below…….

Case statements and Multivalued Logic

- Predicates need not be restricted to binary truth values (TRUE/FALSE). There are multiway predicates, of which FORTRAN three-way IF is the most problematic and the case statement the most useful.

What goes Wrong with Predicates?

- Several things can go wrong with predicates, especially if the predicate has to interpreted in order to express it as a predicate over input values.
  - The wrong relational operator is used.
  - The predicate expression of a compound predicate is incorrect.
  - The wrong operands are used
  - The processing leading to the predicate is faulty.

Boolean Algebra

- Steps taken to get the predicate expression of a path:
- Label each decision with an uppercase letter that represents the truth value of the predicate. The YES/TRUE branch is labeled with a letter and the NO/FALSE branch with the same letter over scored.
- The truth value of a path is the product of the individual labels.
- If two or more paths merge at a node, the fact is expressed by use of a plus sign (+) which means "OR".

The Rules of Boolean Algebra

- Boolean Algebra has three operators:
- x – meaning AND. Also called multiplication. A statement such s AB means "A and B both are true". This symbol is usually left out as ordinary Algebra.
- + - meaning OR. "A+B" means either A is true or B is true or both.
- A^1 meaning NOT. Also negation or complementation. This is read as either not A or A bar.

Laws of Boolean Algebra

- $A+A=A$
- $A^1+A^1=A^1$
- $A+1=1$
- $A+0=A$
- $A+B=B+A$
- $A+A^1=1$
- $AA=A$
- $A^1A^1=A^1$
- $AX1=A$
- $AX0=0$
- $AB=BA$
- $AA^1=0$
- $(A^1)^1=A$
- $0^1=1$
- $1^1=0$
- $(A+B)^1=A^1B^1$
- $(AB)^1=A^1+B^1$
- $A(B+C)=AB+AC$
- $(AB)C=A(BC)$
- $(A+B)+C=A+(B+C)$
- $A+A^1B=A+B^1$
- $A+AB=A$

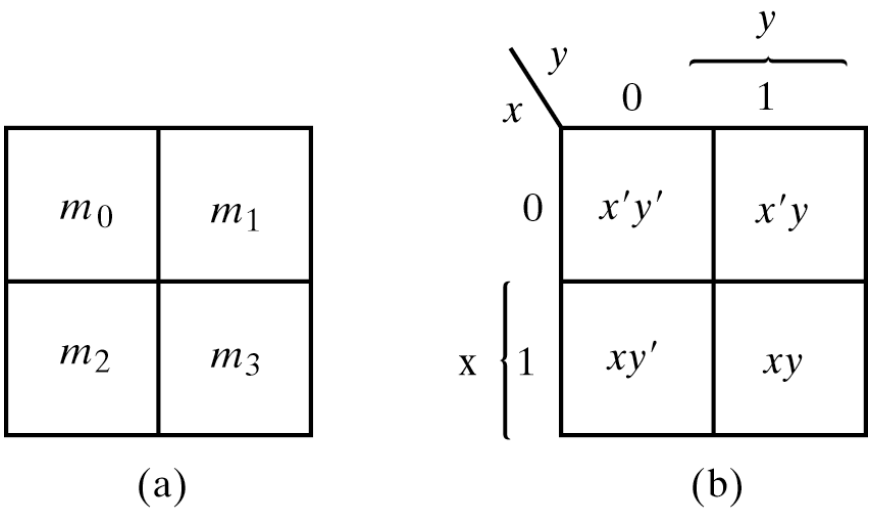
KV-Charts

- The Boolean algebraic expressions are used to determine which cases are interesting and which combination of predicate values should be used to reach which node.
- If you had deal with expressions in four or five or six variables, you could get bogged down in the algebra and make as many errors in the designing test cases as there are bug in the routine you're tesing.
- The karnaugh – Veitch chart reduces boolean algebraic manipulations to graphical trivia.

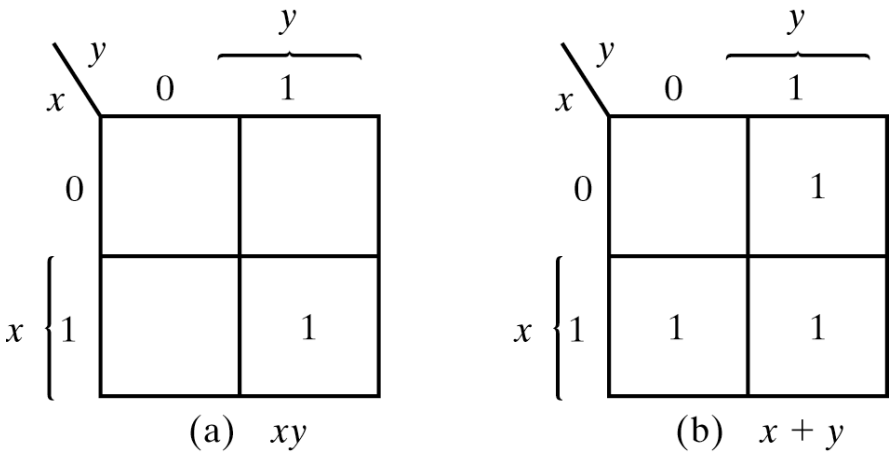
One Variable Map

	A = 0	A = 1	
0	0	0	The function is never true
A	0	1	The function is true when A is true
A^1	1	0	The function is true when A is false
1	1	1	The function is always true

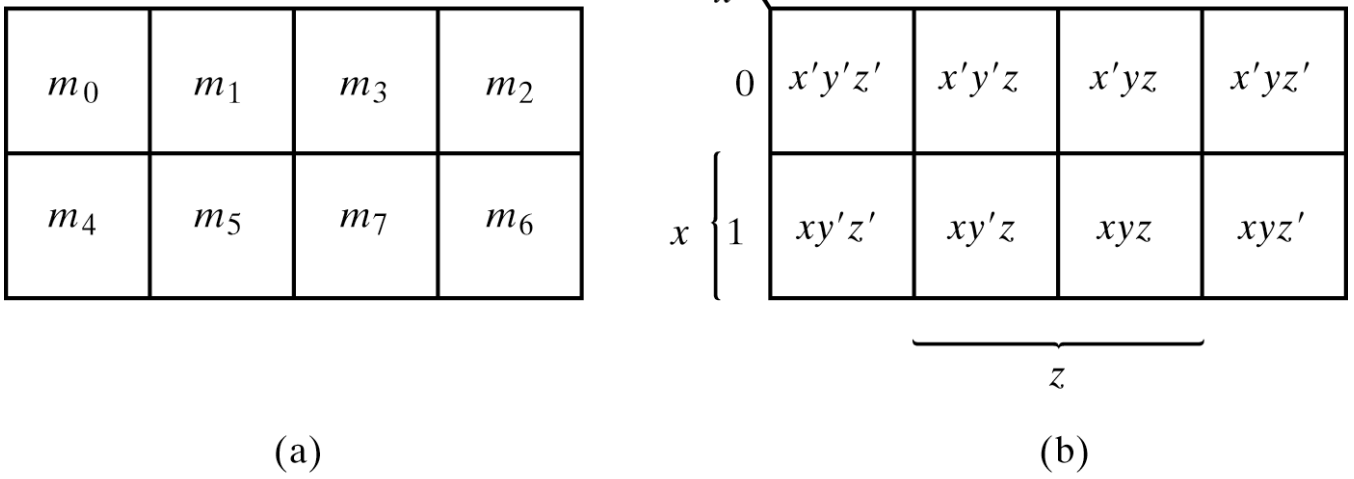
Two Variable Map



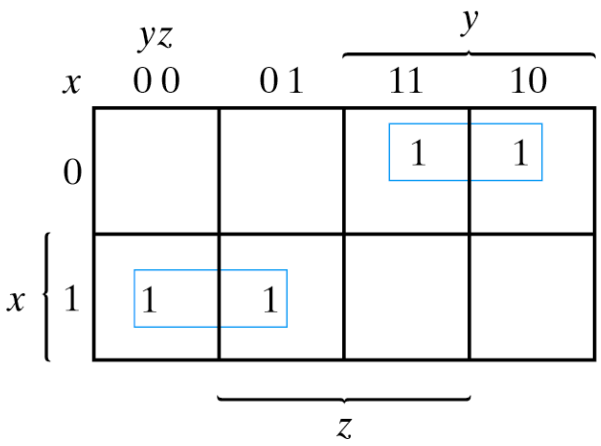
Representation of Functions in the Map



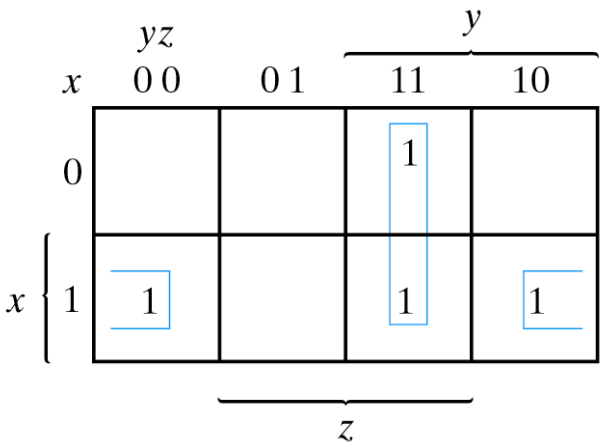
Three variable Map



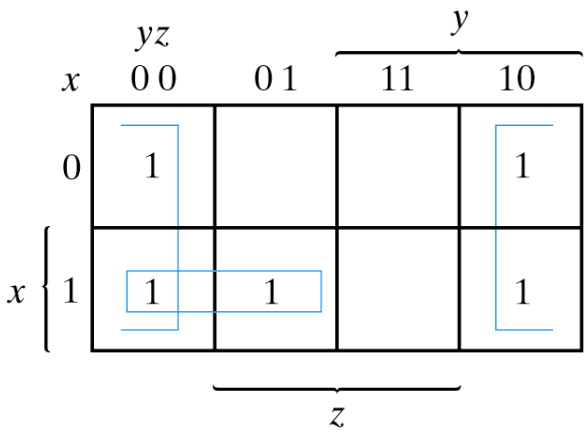
Three variable map-example



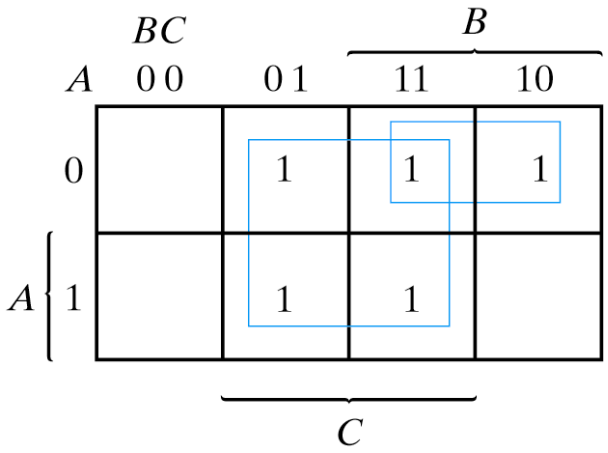
$$F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$$



$$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$$



$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$$

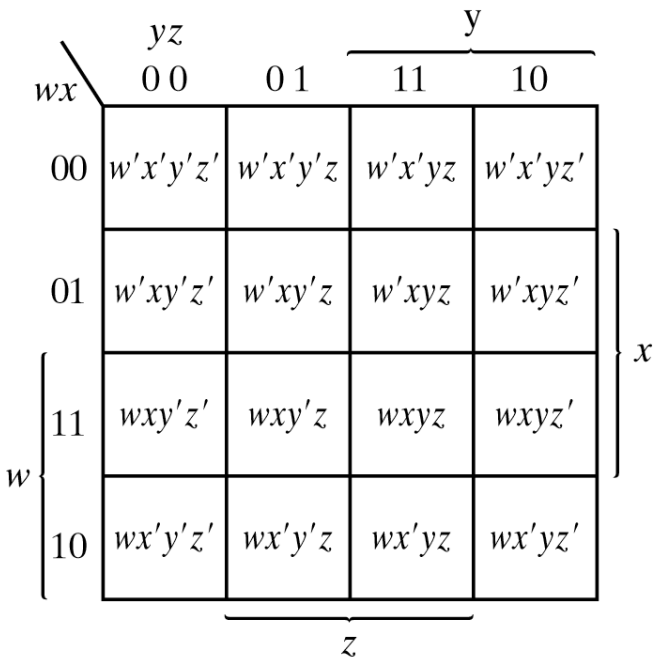


$$A'C + A'B + AB'C + BC = C + A'B$$

Four variable map-example

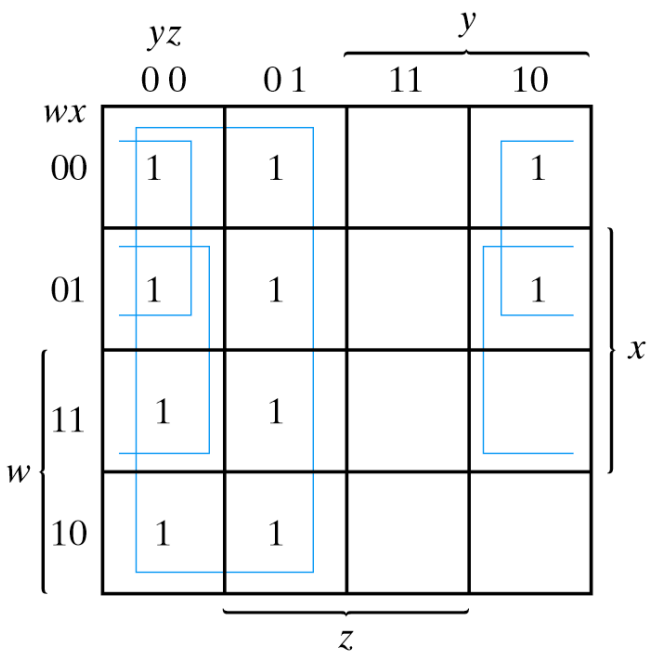
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

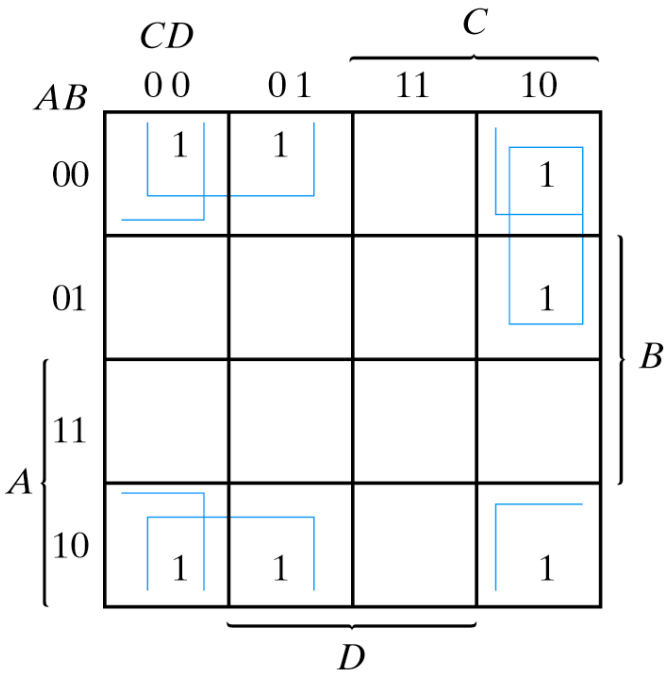


(b)

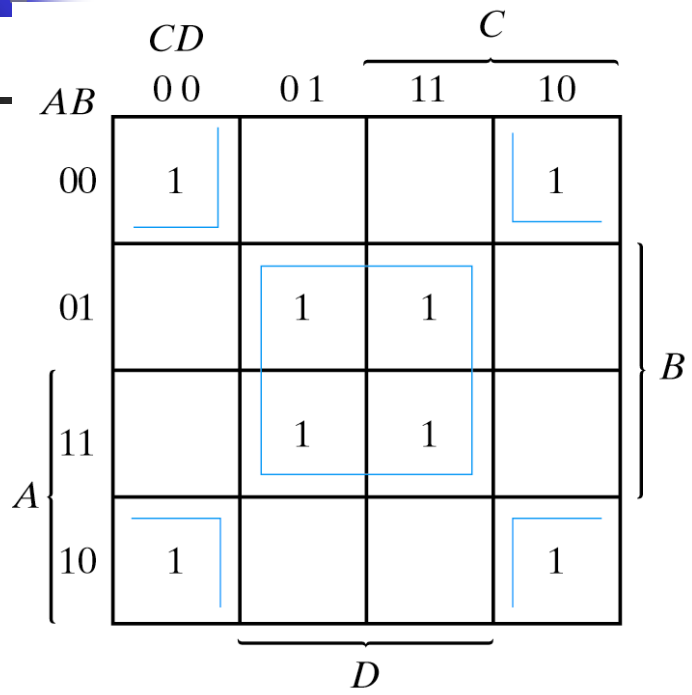
Four variable map-example



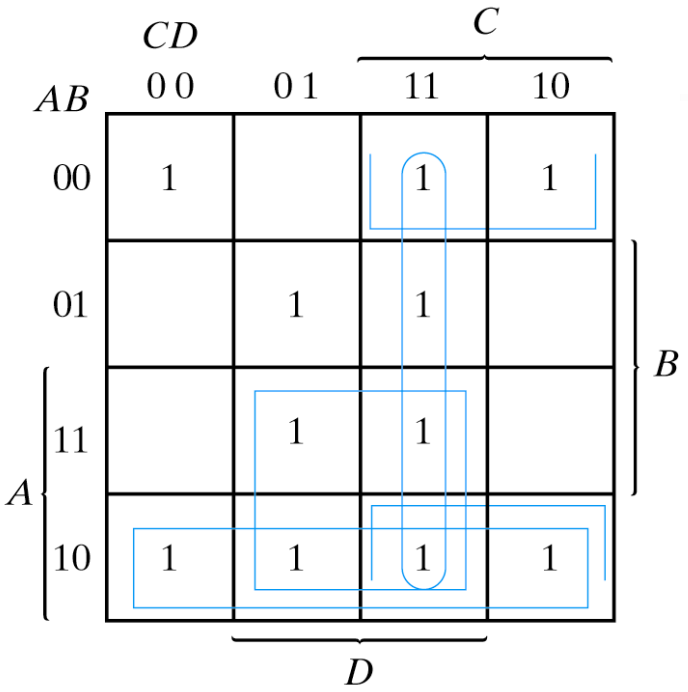
$$F(w, x, y, z)$$
$$= \Sigma\,(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$$



$$A'B'C + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$$

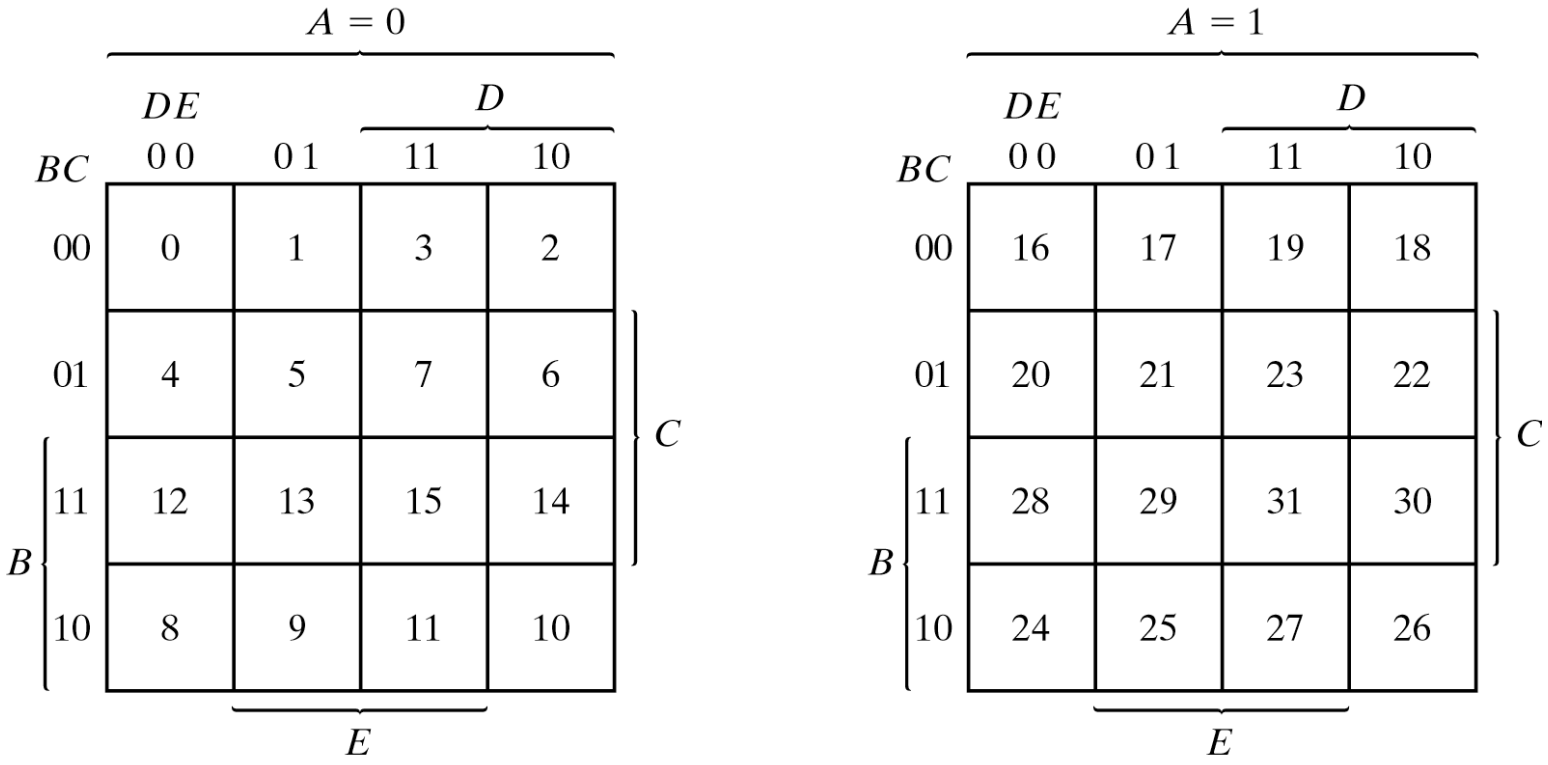


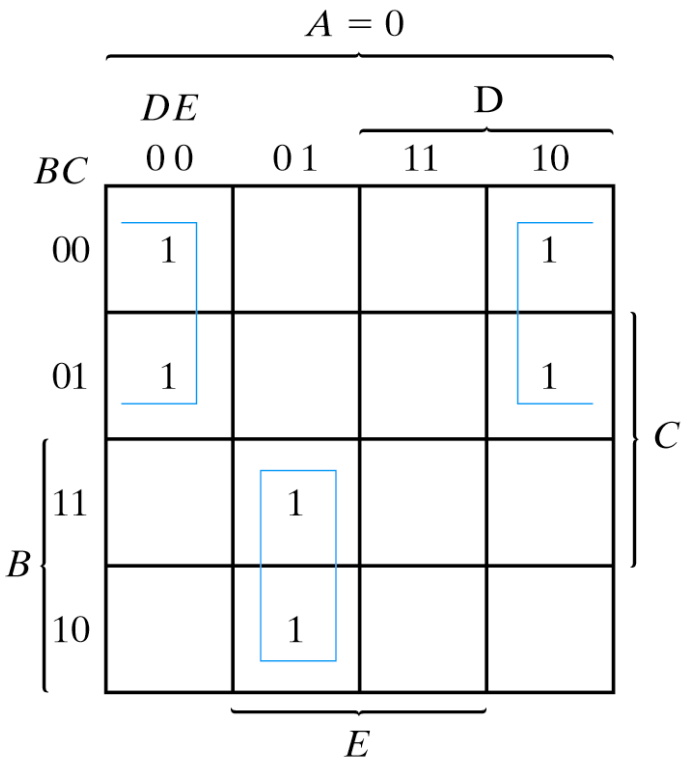
(a) Essential prime implicants BD and B′D′

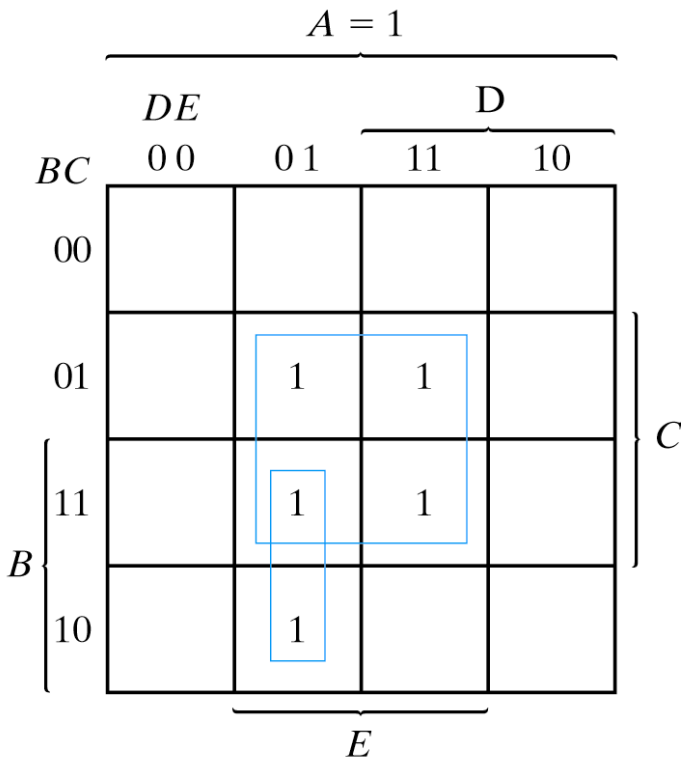


(b) Prime implicants CD, B′C AD, and AB′

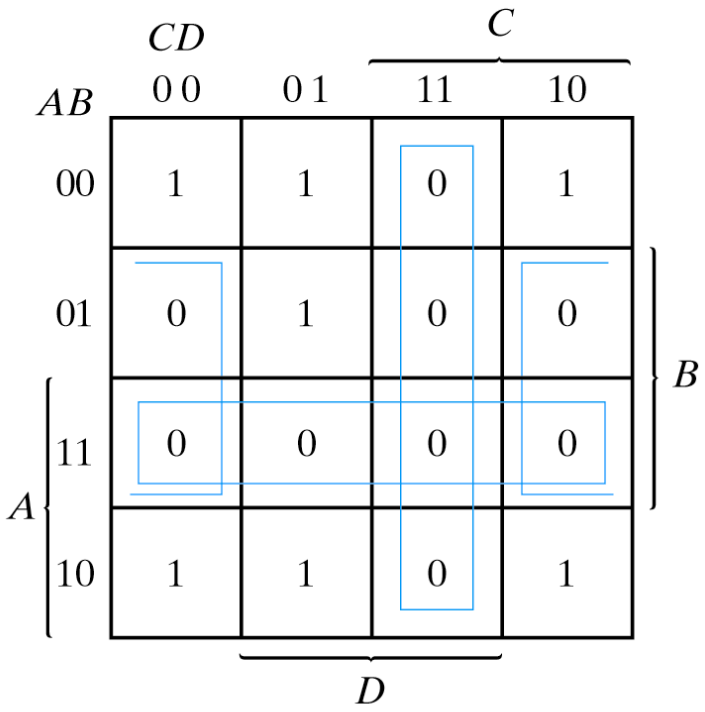
Five variable map-example




A = 0
DE
D
BC
0 0
0 1
11
10
00
01
11
10
B
C
E
1
1
1
1
1
1

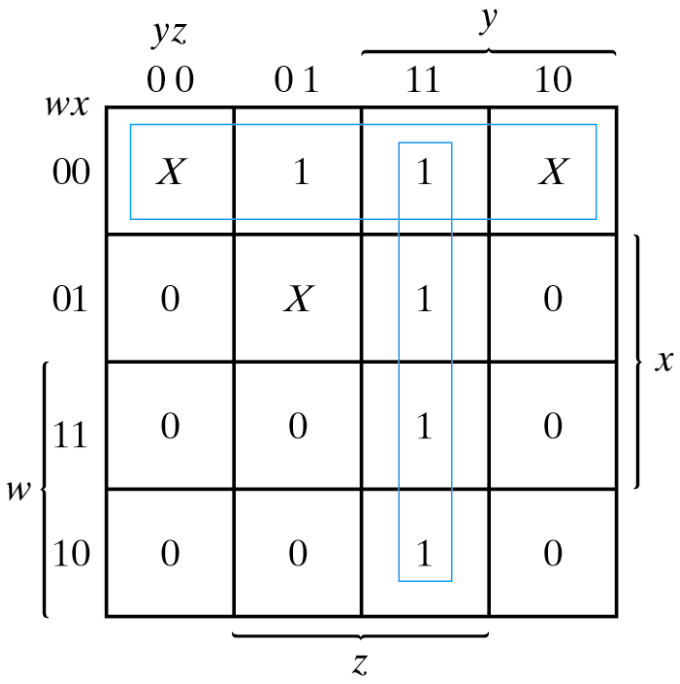

A = 1
DE
D
BC
0 0
0 1
11
10
00
01
11
10
B
C
E
1
1
1
1
1
1

$$F = A'B'E' + BD'E + ACE$$

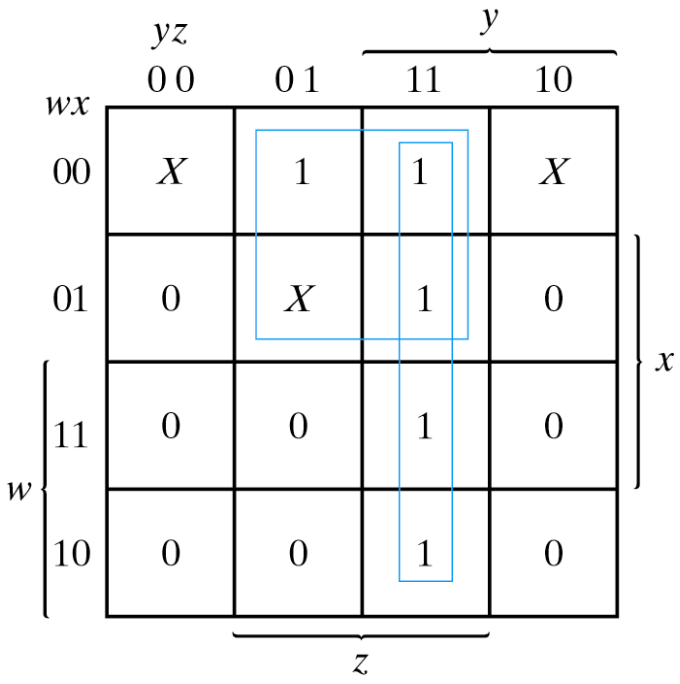


$$F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$$

$$= B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$$



(a) $F = yz + w'x'$



(a) $F = yz + w'z$

Example

- Use a Karnaugh map to minimize

F(A,B,C,D)=AB′C′D′+A′B′C′D′+ABC′D+A′BCD+ABD+B′CD′+A′B

- Some of the terms in the above sum of products form does not contain all the variables.
- F(A,B,C,D)=AB′C′D′+A′B′C′D′+ABC′D+A′BCD+ABD+B′CD′+A′B
- Multiply those terms with the missing variables in $(X+X^1)$ form.

- F(A,B,C,D)=AB′C′D′+A′B′C′D′+ABC′D+A′BCD+ABD+B′CD′+A′B
- ABD=ABD(C+C′)=ABCD+ABC′D
- B′CD′=B′CD′(A+A′)=AB′CD′ +A′B′CD′
- A′B= A′B(C+C′)(D+D′)
  =(A′BC+A′BC′)(D+D′)
  =A′BCD+A′BCD′+ A′BC′D+A′BC′D′

- Finally, the function can be written as

F(A,B,C,D)=AB′C′D′+A′B′C′D′+ABC′D+A′BCD+
ABCD+ABC′D+AB′CD′+A′B′CD′+A′BCD+A′BCD′+
A′BC′D+A′BC′D′=∑(8,0,13,7,15,13,10,2,7,6,5,4)
=∑(0,2,4,5,6,7,8,10,13,15)

- F(A,B,C,D)=∑(0,2,4,5,6,7,8,10,13,15)

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

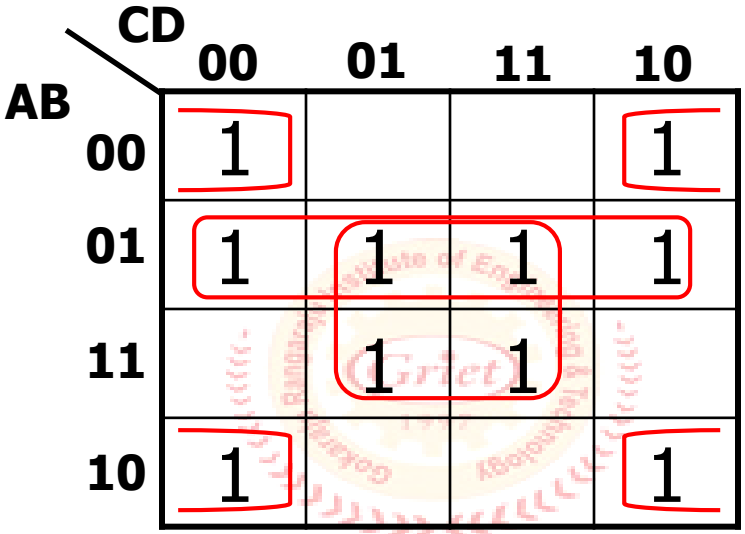
(a)

wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

y (11, 10); x (01, 11); w (11, 10); z (01, 11)

(b)

Four-variable Map



$$F(A,B,C,D)=\Sigma(0,2,4,5,6,7,8,10,13,15)=A^1B+BD+B^1D^1$$