



---

# 逻辑综合

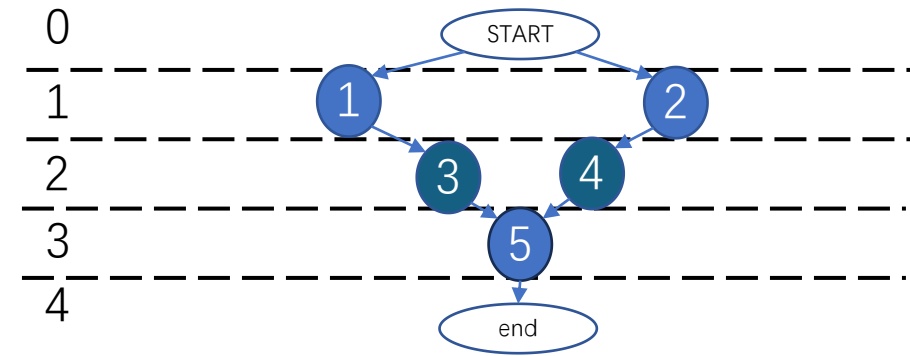
Logic Synthesis

---

# 最优共享问题

## Optimum Sharing Problem

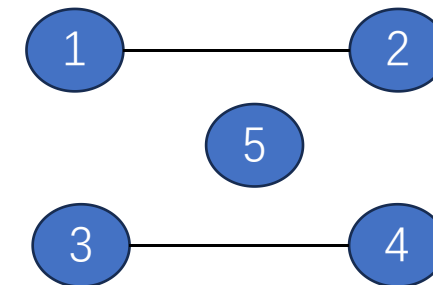
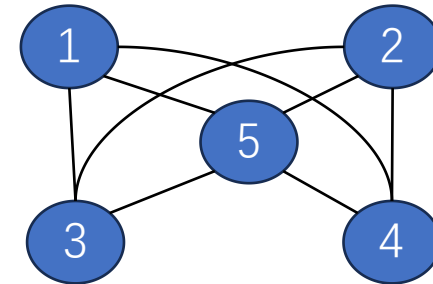
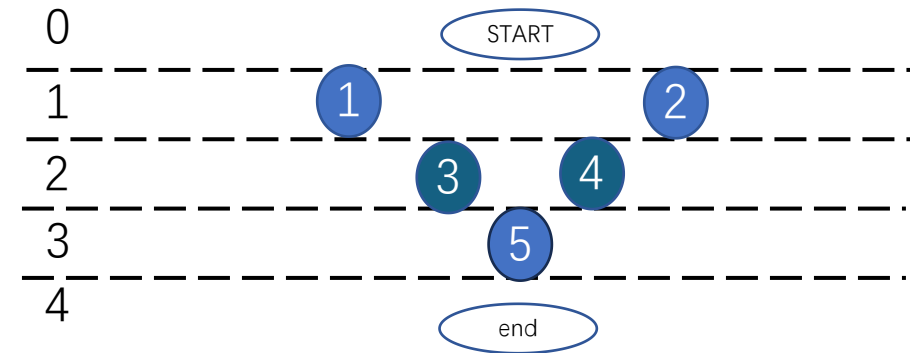
- 输入：已调度的顺序图
  - 已经明确操作的并发性
- 独立考虑操作类型
  - – 问题分解
  - – 针对每种资源类型进行分析



# 兼容与冲突

## Compatibility and Conflicts

- 能兼容的操作：
  - – 相同类型
  - – 非并发
- 兼容图 (Compatibility graph) :
  - – 顶点: 操作
  - – 边: 兼容关系
- 冲突图 (Conflict graph) :
  - – 兼容图的补图





# 兼容与冲突

## Compatibility and Conflicts

- 兼容图 (Compatibility graph) :
  - 将图划分为最少数量的团 (clique)
  - 计算团覆盖数  $\kappa(G)$
- 冲突图 (Conflict graph) :
  - 用尽可能少的颜色对顶点进行着色
  - 计算染色数  $\chi(G)$
- NP 完全问题 (NP-complete problems)



# 兼容图 / 冲突图具有特殊性质

The compatibility/conflict graphs have special properties

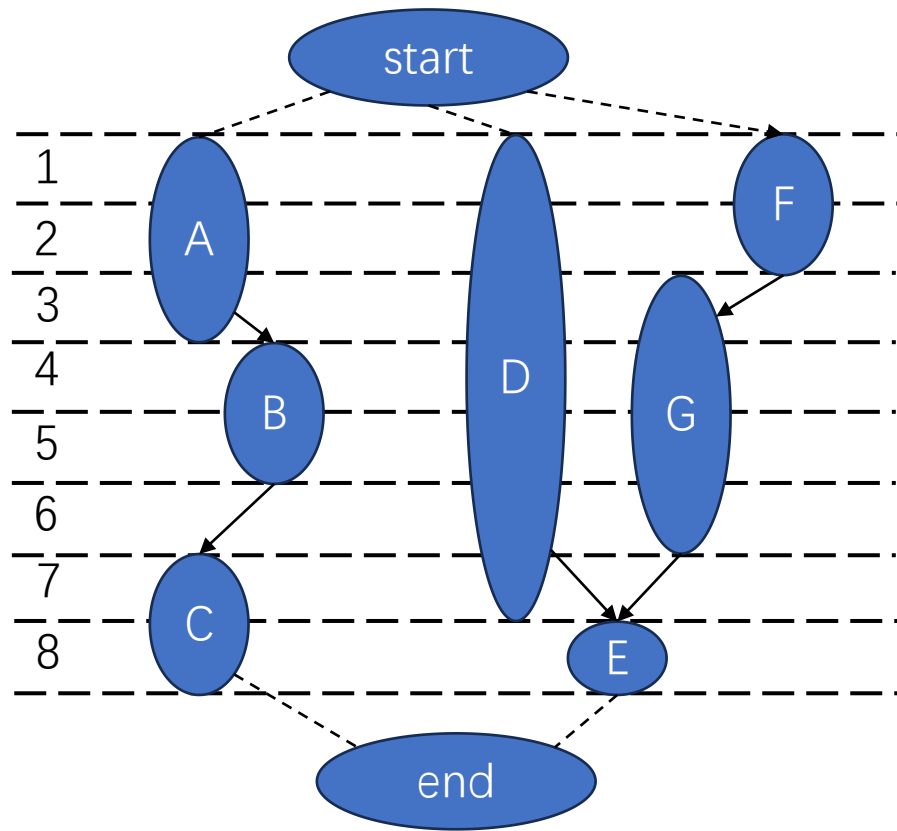
- Compatibility graph 兼容图
  - Comparability graph 比较图
- Conflict graph 冲突图
  - Interval graph 区间图
- 多项式时间算法：
  - Golumbic's algorithm
  - Left-edge algorithm



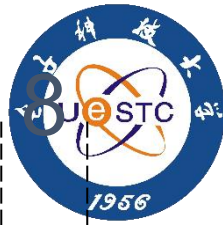
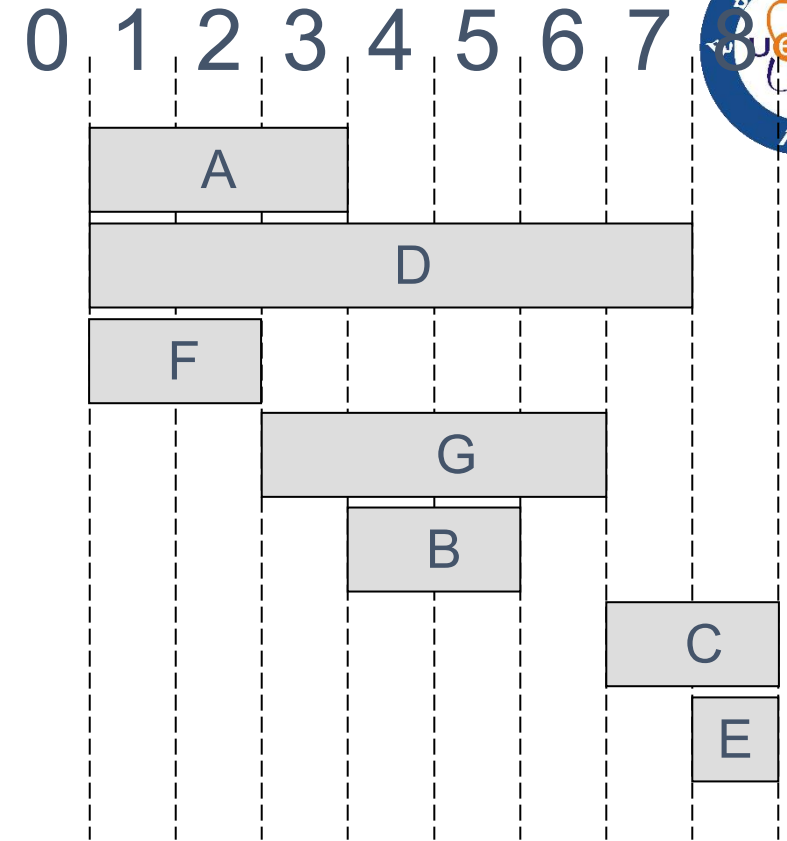
# 左边缘算法

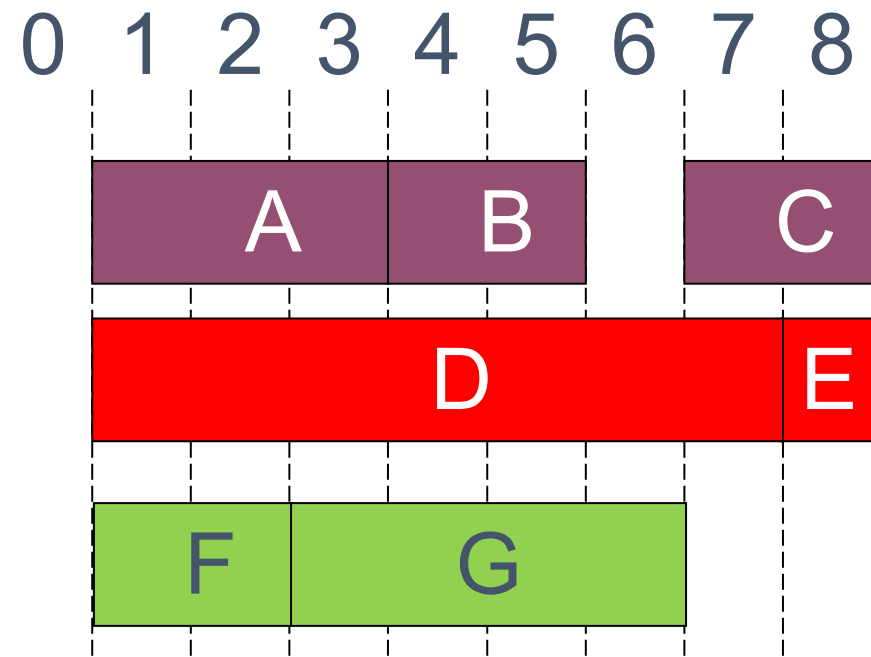
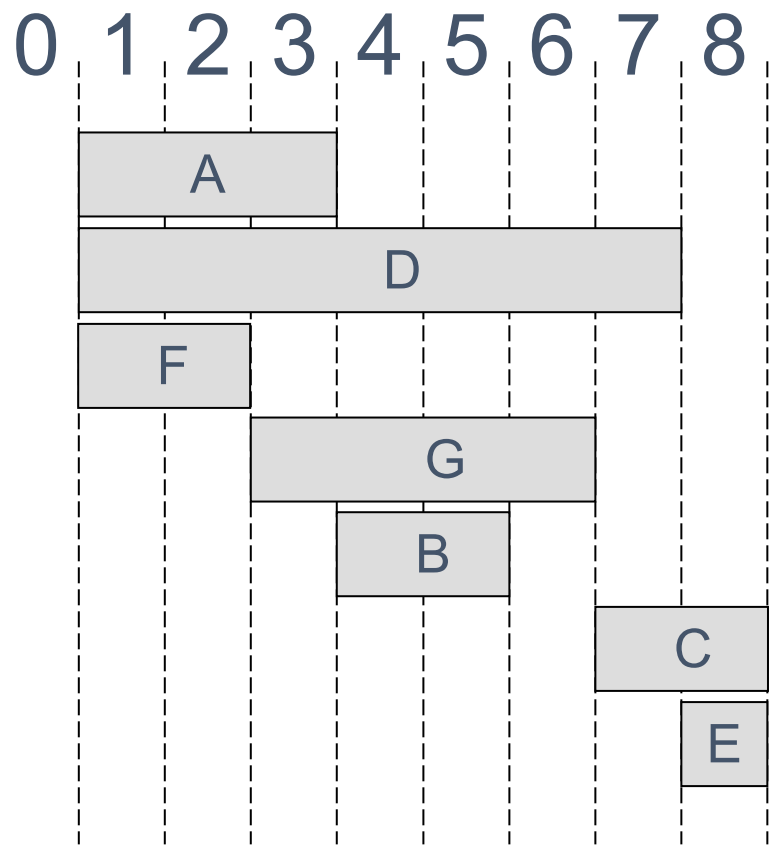
Left-edge algorithm

- 输入：
  - 一组具有左边界和右边界的区间（即开始时间和结束时间）
  - 一组颜色（初始时只有一种颜色）
- 步骤：
  - 按照左边界对区间进行排序
  - 从排序后的列表中，使用第一种颜色分配互不重叠的区间
  - 当所有可能分配的区间都用完后，增加颜色计数器并重复上述过程



A: 1-3  
 B: 4-5  
 C: 7-8  
 D: 1-7  
 E: 8-8  
 F: 1-2  
 G: 3-6









# 用ILP形式化绑定问题

ILP formulation of binding

- 布尔变量  $b_{ir}$ 
  - 表示操作  $i$  是否和资源  $r$  绑定

- 最小化:
$$\sum_{r=1}^a \sum_{i=1}^n b_{i,r}$$

- 约束

1. 对每个操作  $i$  有:

$$\sum_{r=1}^a b_{i,r} = 1$$

2. 对每个资源  $r$ , 对每个冲突关系  $\langle v_i, v_j \rangle$  有:

$$b_{ir} + b_{jr} \leq 1$$



---

# 正式进入逻辑综合

---

# 逻辑综合与优化

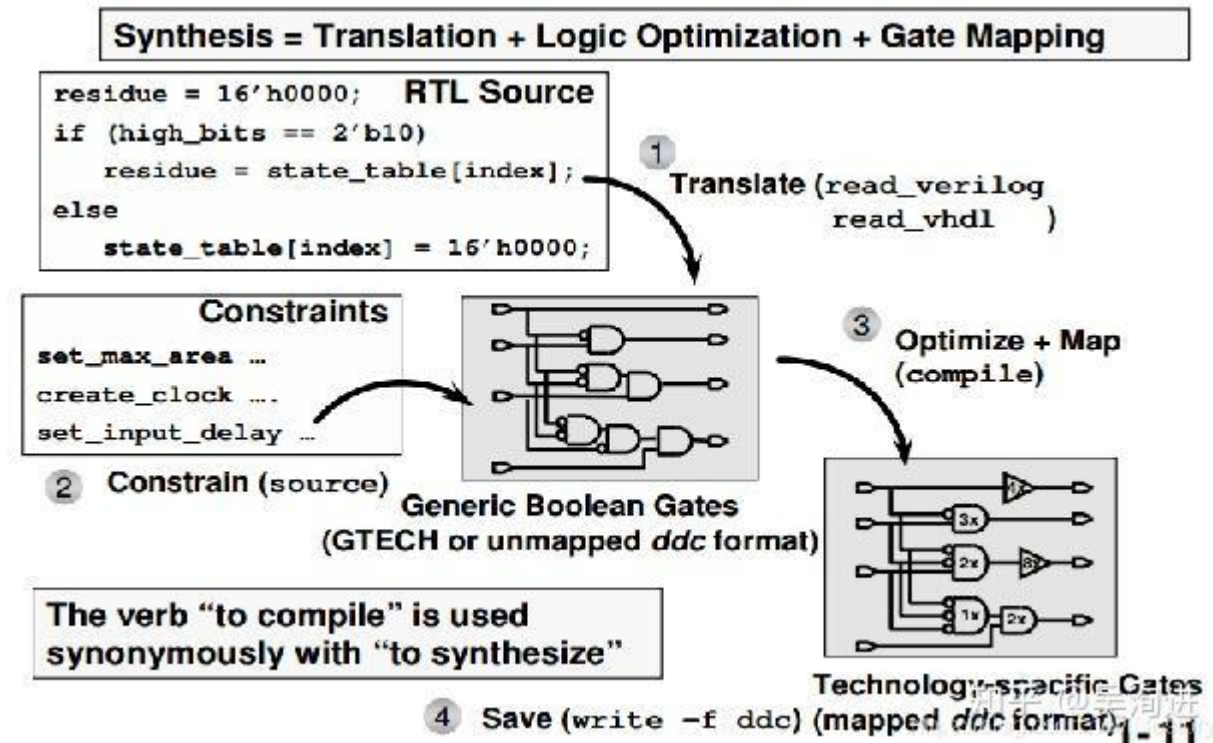
## Logic Synthesis and Optimization

- 逻辑综合是将较高层次的描述(*RTL*级)自动地转换到较低抽象层次描述(门级网表)的一种方法。
- 输入：
  - *RTL*描述的程序模块或者原理图文件或波形文件
  - 工艺库(如*TTL*工艺库, *MOS*工艺库), 包含一些标准的单元
  - 约束条件,用于决定综合过程中的逻辑优化方法。约束条件中一般包含时间, 面积, 速度, 功耗, 负载要求和优化方法等。
- 输出: 门级网表

# 逻辑综合与优化的步骤

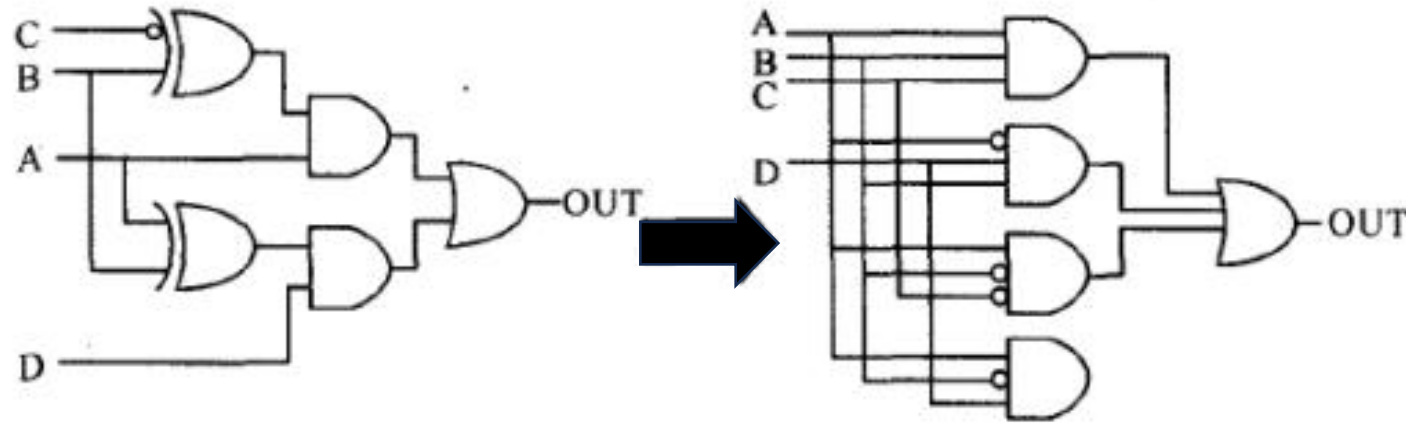
## Logic Synthesis and Optimization steps

- 第一步 (Translation): 转换过程, 将 RTL描述转换成未优化的门级布尔描述(如与门, 或门, 触发器, 锁存器等)。
- 第二步 (Logic Optimization): 布尔优化过程, 将一个非优化的布尔描述转化成一个优化的布尔描述的过程。
- 第三步 (Mapping): 门级映射过程, 取出经优化的布尔描述, 利用从工艺库中得到的逻辑和定时的信息生成门级网表, 确保得到的门级网表能达到设计的性能和面积要求。



# 组合逻辑优化

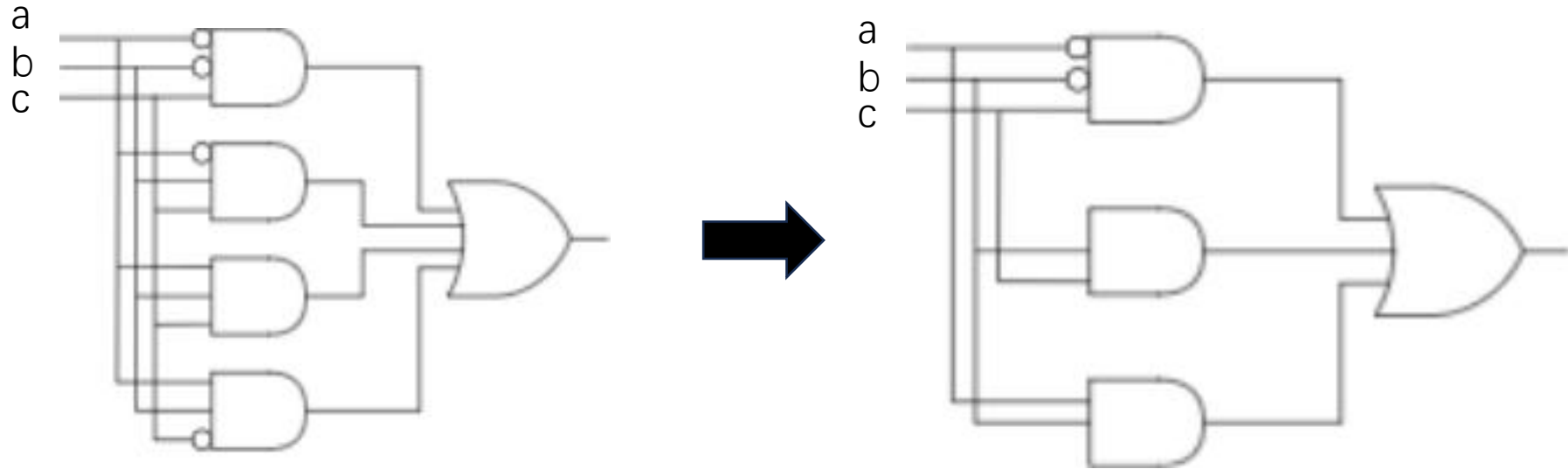
## Combinational Logic Optimization



$$\begin{aligned} \text{OUT} &= (C' \oplus B)A + (A \oplus B)D \\ &= A(BC + B'C') + D(AB' + A'B) \\ &= ABC + A'BD + AB'C' + AB'D \end{aligned}$$

# 组合逻辑优化

## Combinational Logic Optimization



$$\begin{aligned}
 x &= a'b'c + a'bc + abc + abc' \\
 &= a'b'c + a'bc + abc + abc + abc' \\
 &= a'b'c + (a' + a)bc + ab(c + c') \\
 &= a'b'c + bc + ab
 \end{aligned}$$



# 组合逻辑优化

## Combinational Logic Optimization

- 在组合逻辑优化环节，其中一个重要部分是布尔表达式的最小化，即找到最简单门级电路实现给定的布尔表达式。



---

# 布尔函数

Boolean Function

---



# 布尔函数

## Boolean Function

- 设  $B = \{0, 1\}$
- 布尔函数  $f$  是一个具有  $n$  个输入  $x_1, x_2, \dots, x_n$  和  $m$  个输出  $y_1, y_2, \dots, y_m$  的函数:

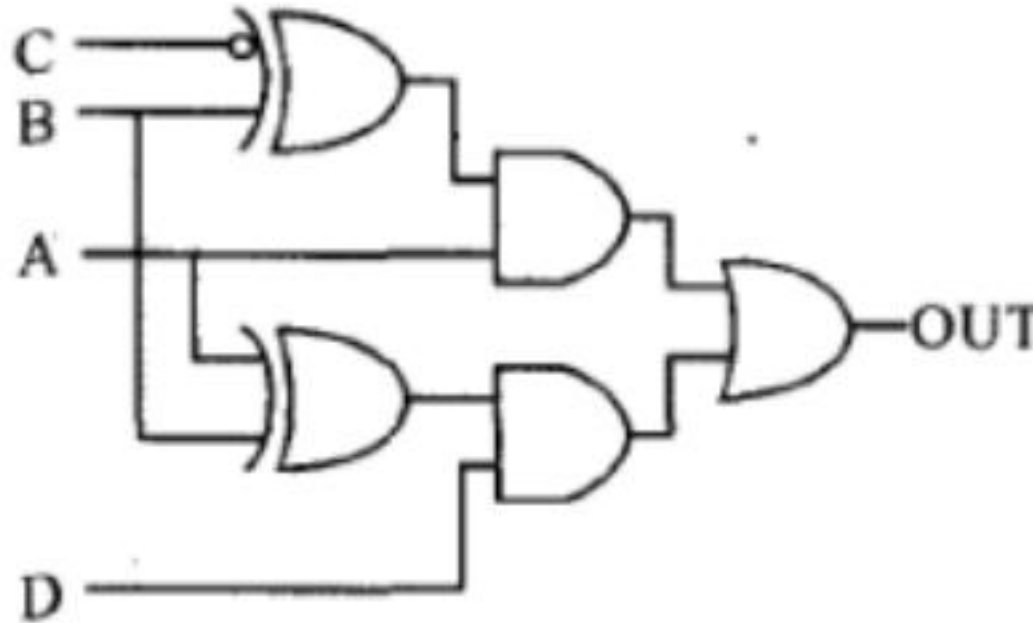
单个输出函数:  $f: B^n \rightarrow B$

多个输出函数:  $f: B^n \rightarrow B^m$

# 布尔函数

## Boolean Function

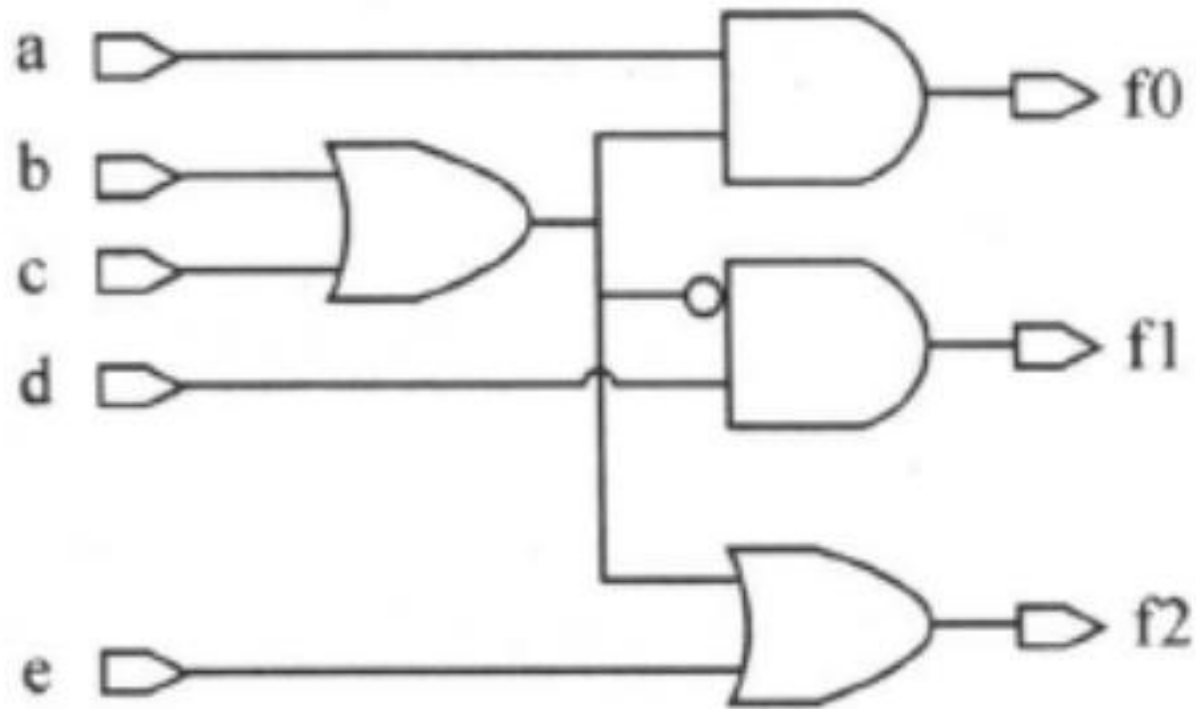
单个输出函数:  $f: B^n \rightarrow B$



# 布尔函数

## Boolean Function

多个输出函数:  $f: B^n \rightarrow B^m$



# 布尔函数

## Boolean Function

- 设  $B = \{0, 1\}$
- 布尔函数  $f$  是一个具有  $n$  个输入  $x_1, x_2, \dots, x_n$  和  $m$  个输出  $y_1, y_2, \dots, y_m$  的函数:

单个输出函数:  $f: B^n \rightarrow B$

多个输出函数:  $f: B^n \rightarrow B^m$

# 布尔函数

## Boolean Function

- 设  $B = \{0, 1\}$ ,  $Y = \{0, 1, 2\}$
- 布尔函数  $f$  是一个具有  $n$  个输入  $x_1, x_2, \dots, x_n$  和  $m$  个输出  $y_1, y_2, \dots, y_m$  的函数:

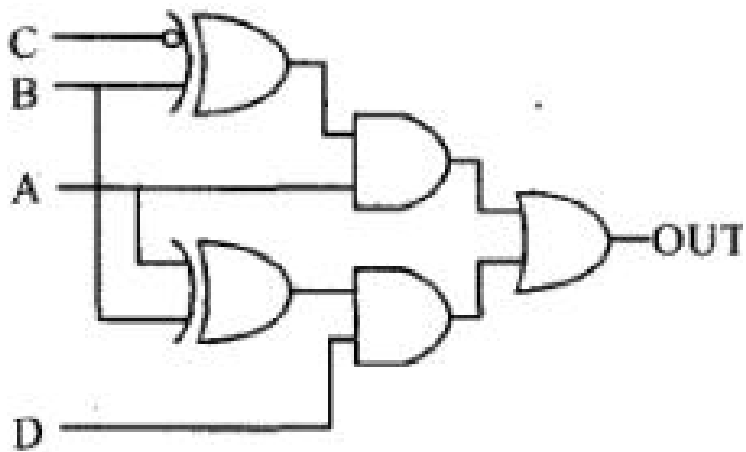
单个输出函数:  $f: B^n \rightarrow Y$

多个输出函数:  $f: B^n \rightarrow Y^m$

# 不关心

don' t care

- 在某些输入情况下，我们根本不在乎输出是什么，是 0 还是 1 都无所谓
- 可能有什么样的情况：
  - 永远不会发生的输入模式

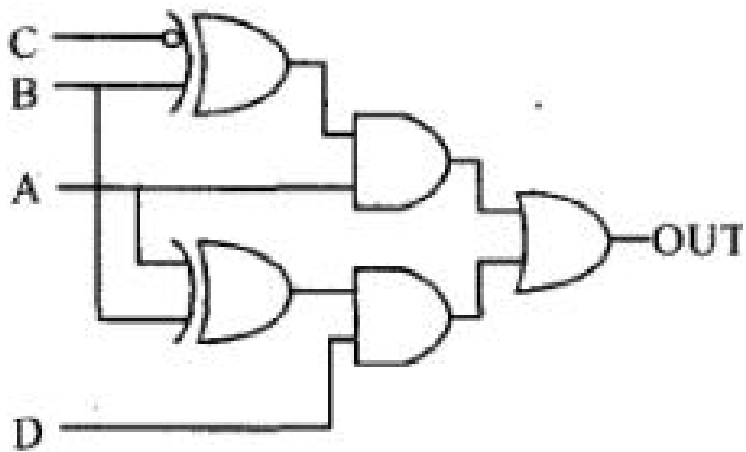


	A	B	C	D	OUT
1	0	0	0	0	0
2	0	0	0	1	0
3	0	0	1	0	0
4	0	0	1	1	0
5	0	1	0	0	0
6	0	1	0	1	1
7	0	1	1	0	0
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	1
11	1	0	1	0	0
12	1	0	1	1	1
13	1	1	0	0	0
14	1	1	0	1	0
15	1	1	1	0	1
16	1	1	1	1	1

# 不关心

don' t care

- 在某些输入情况下，我们根本不在乎输出是什么，是 0 还是 1 都无所谓
- 可能有什么样的情况：
  - 永远不会发生的输入模式

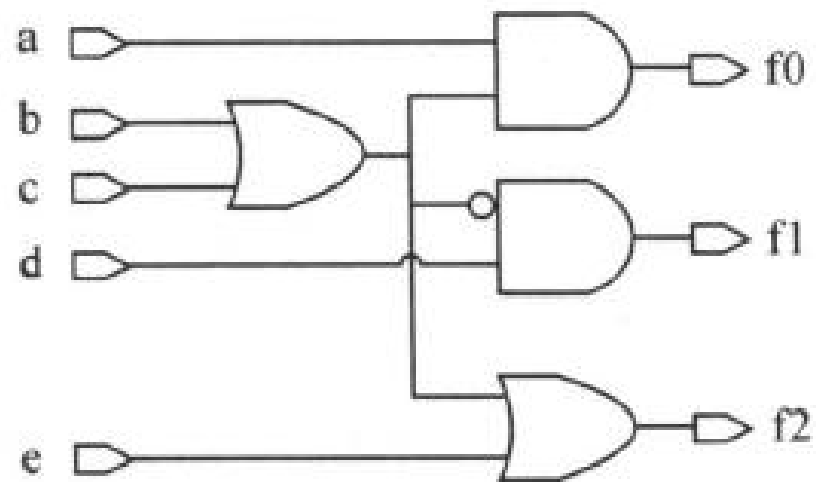


	A	B	C	D	OUT
1	0	0	0	0	0
2	0	0	0	1	0
3	0	0	1	0	0
4	0	0	1	1	0
5	0	1	0	0	0
6	0	1	0	1	1
7	0	1	1	0	0
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	1
11	1	0	1	0	2
12	1	0	1	1	2
13	1	1	0	0	2
14	1	1	0	1	2
15	1	1	1	0	2
16	1	1	1	1	2

# 不关心

don' t care

- 在某些输入情况下，我们根本不在乎输出是什么，是 0 还是 1 都无所谓
- 可能有怎样的情况：
  - 永远不会发生的输入模式
  - 某些输出永远不会被观察到的输入模式





## 定义

- 对于每个组件  $f_i$ ,  $i = 1, 2, \dots, m$ , 定义:
  - ON\_SET: 使得  $f_i(x) = 1$  的输入值  $x$  的集合
  - OFF\_SET: 使得  $f_i(x) = 0$  的输入值  $x$  的集合
  - DC\_SET: 使得  $f_i(x) = 2$  的输入值  $x$  的集合 (即“无关项”)

## 例子

A	B	C	D	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

- ON-SET =  $\{(0\ 1\ 0\ 1), (0\ 1\ 1\ 1), (1\ 0\ 0\ 0), (1\ 0\ 0\ 1), (1\ 0\ 1\ 1), (1\ 1\ 1\ 0), (1\ 1\ 1\ 1)\}$
- OFF-SET =  $\{(0\ 0\ 0\ 0), (0\ 0\ 0\ 1), (0\ 0\ 1\ 0), (0\ 0\ 1\ 1), (0\ 1\ 0\ 0), (0\ 1\ 1\ 0), (1\ 0\ 1\ 0), (1\ 1\ 0\ 0), (1\ 1\ 0\ 1)\}$
- DC-SET =  $\{\}$

完全指定函数(Completely specified function)

## 定义

- 对于每个组件  $f_i$ ,  $i = 1, 2, \dots, m$ , 定义:
  - ON\_SET: 使得  $f_i(x) = 1$  的输入值  $x$  的集合
  - OFF\_SET: 使得  $f_i(x) = 0$  的输入值  $x$  的集合
  - DC\_SET: 使得  $f_i(x) = 2$  的输入值  $x$  的集合 (即“无关项”)
- 完全指定函数(Completely specified function):
  - 对所有  $f_i$ , 有  $DC\_SET = \text{空集}$
- 不完全指定函数(Incompletely specified function):
  - 存在某个  $f_i$ , 使得  $DC\_SET \neq \text{空集}$

## 例子

A	B	C	D	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	2
1	0	1	1	2
1	1	0	0	2
1	1	0	1	2
1	1	1	0	2
1	1	1	1	2

- ON-SET =  $\{(0\ 1\ 0\ 1), (0\ 1\ 1\ 1), (1\ 0\ 0\ 0), (1\ 0\ 0\ 1)\}$
- OFF-SET =  $\{(0\ 0\ 0\ 0), (0\ 0\ 0\ 1), (0\ 0\ 1\ 0), (0\ 0\ 1\ 1), (0\ 1\ 0\ 0), (0\ 1\ 1\ 0)\}$
- DC-SET =  $\{(1\ 0\ 1\ 0), (1\ 0\ 1\ 1), (1\ 1\ 0\ 0), (1\ 1\ 0\ 1), (1\ 1\ 1\ 0), (1\ 1\ 1\ 1)\}$

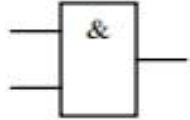

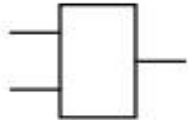
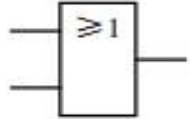
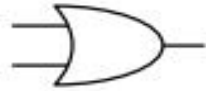
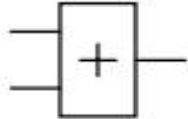
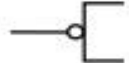
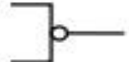
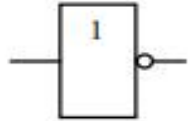
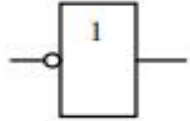
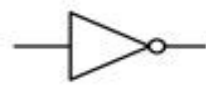
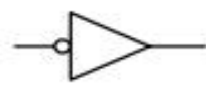
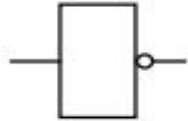
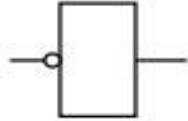
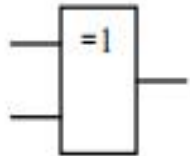

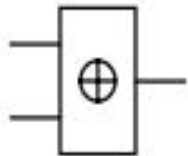
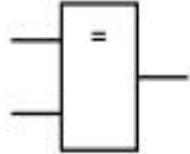
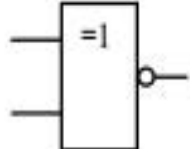

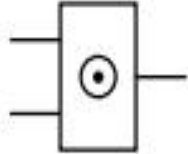
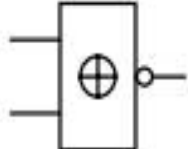
不完全指定函数(Incompletely specified function)



# 逻辑函数的表达方式

- 1. 电路表示
- 2. 代数表示
- 3. 表格表示
- 4. 图形表示
- 5. 面向计算机的表示

表 C1 基本逻辑门电路图形符号

名称	GB/T 4728.12-1996		国外流行图形符号	曾用图形符号
	限定符号	国标图形符号		
与门	&			
或门	$\geq 1$			
非门	  逻辑非入和出	 	 	 
异或门	$=1$			
同或门	$=$	 		 

符号:  $A \cdot B$  或  $A \wedge B$  或  $A \& B$ 符号:  $A + B$  或  $A \vee B$  或  $A | B$ 符号:  $A'$  或  $\bar{A}$  或  $!A$ 符号:  $A \oplus B$  或  $A \wedge B$   
 $= A' B + B' A$ 符号:  $A \odot B$  或  $!(A \wedge B)$   
 $= AB + A' B'$ 

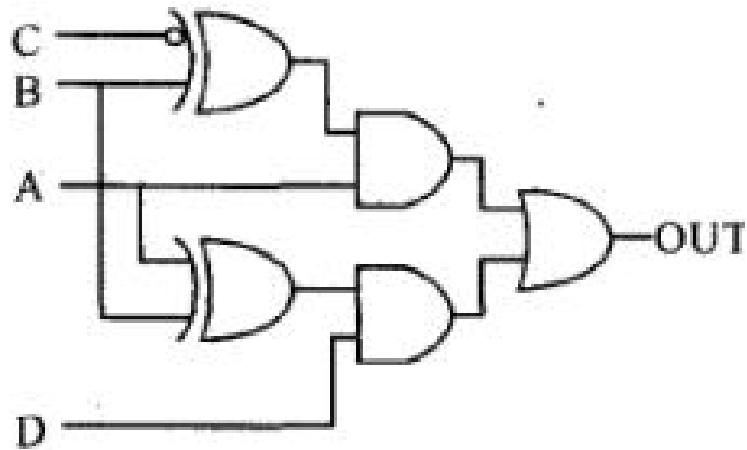
①	交换律	$a + b = b + a$	$ab = ba$
②	结合律	$a + (b + c) = (a + b) + c$	$a(bc) = (ab)c$
③	分配律	$a + bc = (a + b)(a + c)$	$a(b + c) = ab + ac$
④	0—1律 <sub>1</sub>	$0 + a = a$	$1 \cdot a = a$
⑤	0—1律 <sub>2</sub>	$1 + a = 1$	$0 \cdot a = 0$
⑥	互补律	$a + \bar{a} = 1$	$a \cdot \bar{a} = 0$
⑦	吸收律 <sub>1</sub>	$a + ab = a$	$a(a + b) = a$
⑧	吸收律 <sub>2</sub>	$a + \bar{a}b = a + b$	$a(\bar{a} + b) = ab$
⑨	重叠律	$a + a = a$	$a \cdot a = a$
⑩	反演律	$\overline{a + b} = \bar{a}\bar{b}$	$\overline{ab} = \bar{a} + \bar{b}$
⑪	对合律	$\overline{\bar{a}} = a$	
⑫	包含律	$ab + \bar{a}c + bc = ab + \bar{a}c$	$(a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c)$

## 2. 代数表示

- 布尔文字 (Boolean literals) :
  - 变量及其补码 Eg:  $a, a', b, \dots$
- 乘积 (Product or cube) :
  - literals的乘积 Eg:  $ab', a'b', \dots$
- 最小项 (Minterm) :
  - **每个变量**都以原形式或否定形式出现一次, 且“与”在一起,  
通常只列出值为1的项
- 蕴含项 (Implicant) :
  - 由一个或多个最小项合并而成的更广泛的表达式 Eg:  $*b'$



## 2. 代数表示-例子



$$\text{OUT} = A(BC + B'C') + D(AB' + A'B)$$

literals: A A' B B' C C' D

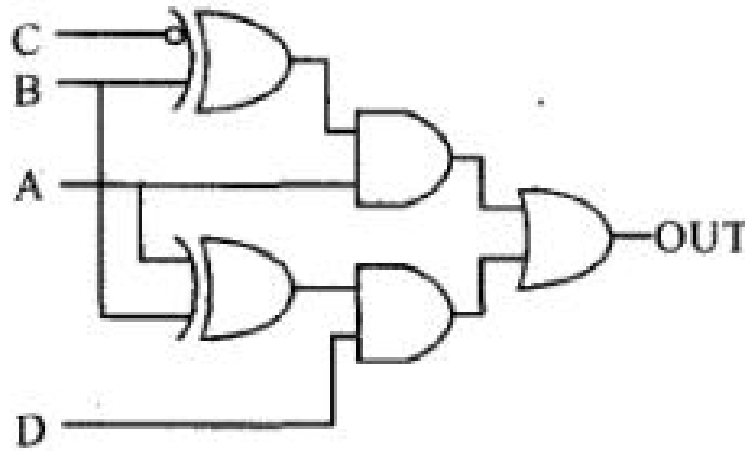
product: BC B'C' AB' A'B

minterm: ABCD ABCD' AB'C'D AB'C'D'

AB'CD A'BCD A'BC'D

A	B	C	D	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

## 2. 代数表示-例子



implicant

$$OUT = A(BC + B'C') + D(AB' + A'B)$$

literals: A A' B B' C C' D

product: BC B'C' AB' A'B

minterm: ABCD ABCD' AB'C'D AB'C'D'

AB'CD A'BCD A'BC'D

$$ABCD + ABCD' + AB'C'D + AB'C'D' + AB'CD + A'BCD + A'BC'D$$

ABC\*

AB'C'\*

AB'\*D

A'B\*D



### 3. 表格表示

- 真值表(Truth table)
  - 函数所有最小项的列表
- 蕴含项表(Implicant table)
  - 定义函数的蕴含项列表
- 注意：
  - 蕴含项表的大小比真值表小

## 真值表-例子

- $x = ab + a'c$ ;  $y = ab + bc + ac$

	abc	x	y
	000	0	0
	001	1	0
	010	0	0
*11	011	1	1
	100	0	0
	101	0	1
11*	110	1	1
	111	1	1

## 蕴含项表-例子

- $x = ab + a'c$ ;  $y = ab + bc + ac$

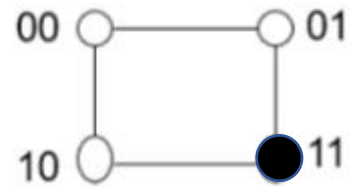
abc	x	y
001	1	0
*11	1	1
101	0	1
11*	1	1

## 4. 图形表示-几何表示

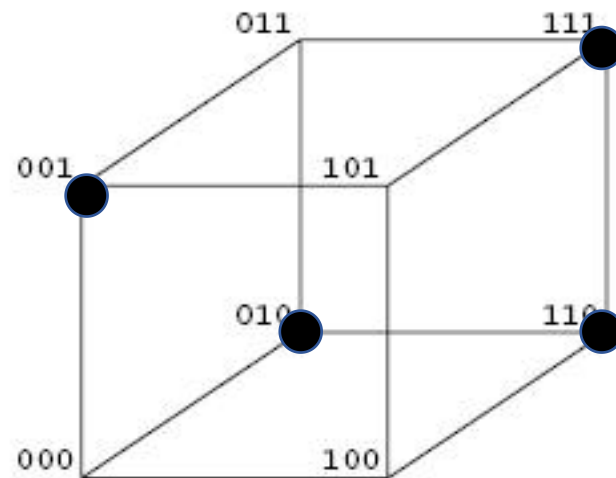
• 一个变量



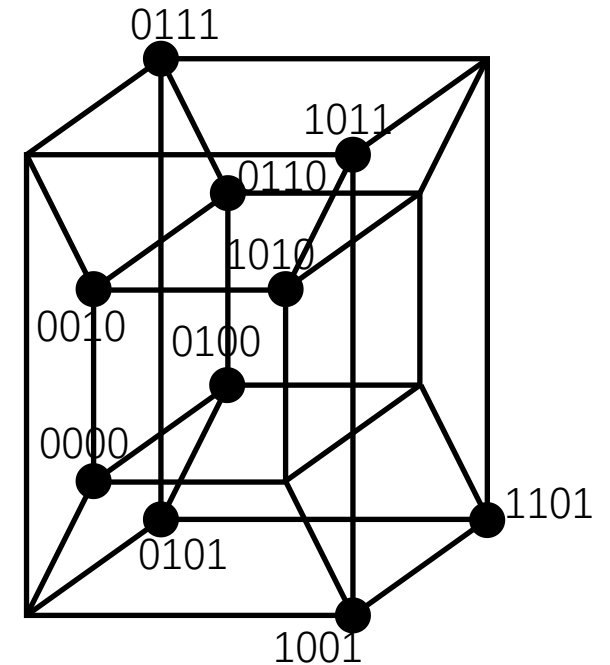
• 两个变量



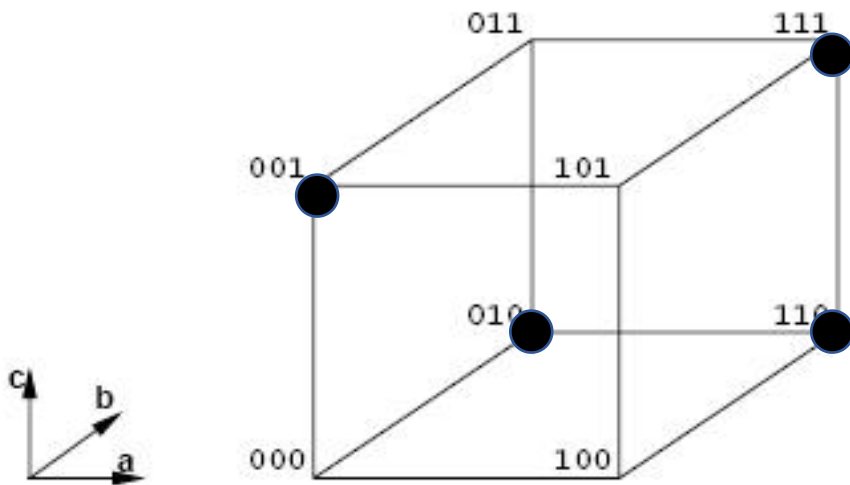
• 三个变量



• 四个变量



# 最小项的几何表示-例子

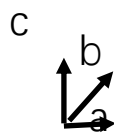


$$F = A'B'C + A'BC' + AB'C' + ABC$$

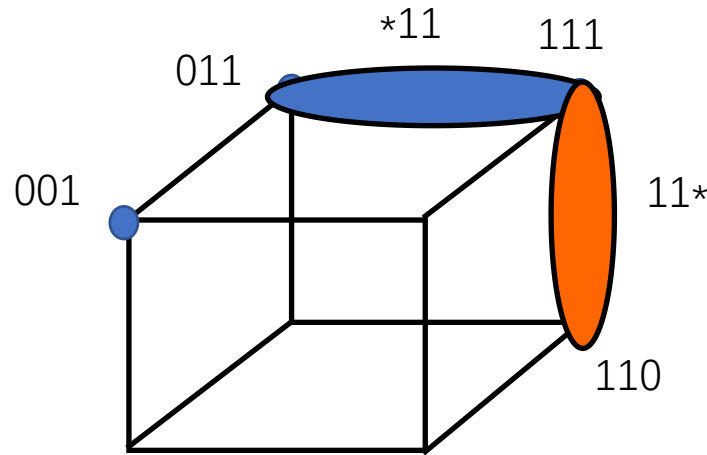
- ON-SET =  $\{(0\ 0\ 1), (0\ 1\ 0), (1\ 0\ 0), (1\ 1\ 1)\}$
- OFF-SET =  $\{(0\ 1\ 1), (1\ 0\ 1), (1\ 1\ 0), (0\ 0\ 0)\}$

# 蕴含项的几何表示-例子

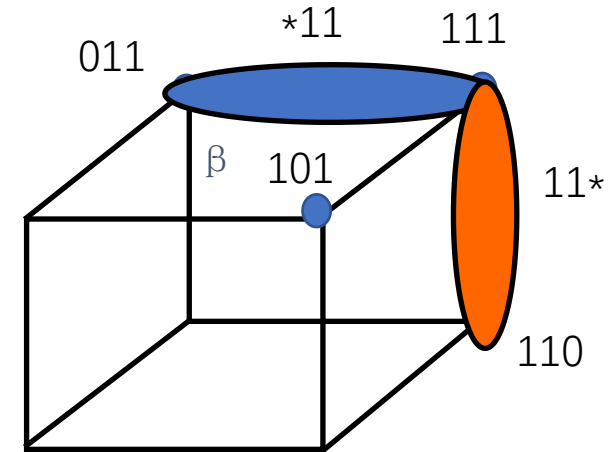
abc	xy
001	10
*11	11
101	01
11*	11



- $x = a'b'c + a'bc + abc + abc'$
- $y = a'bc + ab'c + abc + abc'$



f1



f2



## 4. 图形表示-卡诺图

- 卡诺图是一种包含一些小方块的几何图形，图中每个小方块代表一个单元，**每个单元对应一个最小项**。两个相邻的最小项（即只有一位不同的最小项）在卡诺图中必然也是相邻的。卡诺图中相邻的含义是：
  - 几何相邻性，即几何位置上相邻，也就是左右紧挨着或者上下相接
  - 对称相邻性，即图形中对称位置的单元是相邻的。

		AB			
		00	01	11	10
CD	00	(1)		(1)	
	01			(1)	
	11	(1)	(1)	(1)	(1)
	10				

$$F = A'B'C'D' + ABC'D' + ABC'D + A'B'CD + A'BCD + ABCD + AB'CD$$

# 卡诺图 — 蕴含项

## karnaugh map - implicant

- 卡诺图上任何 $2^n$ 个标1的方格，且他们相邻或对称，则可以合并为一项，每一个合并的圈都是一个蕴含项，例如下图有：
- 一次蕴含项：1100, 1101, 0101, 1011, 0010, 0110, 1110, 1010
- 二次蕴含项：110\*, 101\*, \*101, 0\*10, \*110, 1\*10, 11\*0, \*010

		AB			
		00	01	11	10
CD	00	0	0	1	0
	01	0	1	1	0
	11	0	0	0	1
	10	1	1	1	1

# 卡诺图 — 蕴含项

## karnaugh map - implicant

- 卡诺图上任何 $2^n$ 个标1的方格，且他们相邻或对称，则可以合并为一项，每一个合并的圈都是一个蕴含项，例如下图有：
- 一次蕴含项：1100, 1101, 0101, 1011, 0010, 0110, 1110, 1010
- 二次蕴含项：110\*, 101\*, \*101, 0\*10, \*110, 1\*10, 11\*0, \*010

		AB			
		00	01	11	10
CD	00	0	0	1	0
	01	0	1	1	0
	11	0	0	0	1
	10	1	1	1	1

# 卡诺图 — 蕴含项

## karnaugh map - implicant

- 卡诺图上任何 $2^n$ 个标1的方格，且他们相邻或对称，则可以合并为一项，每一个合并的圈都是一个蕴含项，例如下图有：
- 一次蕴含项：1100, 1101, 0101, 1011, 0010, 0110, 1110, 1010
- 二次蕴含项：110\*, 101\*, \*101, 0\*10, \*110, 1\*10, 11\*0, \*010
- 四次蕴含项：\*\*10

		AB			
		00	01	11	10
CD	00	0	0	1	0
	01	0	1	1	0
	11	0	0	0	1
	10	1	1	1	1

# 面向计算机的表示



- 矩阵
  - 稀疏
  - 不同编码方式
- 二叉决策图(BDD)
  - 解决稀疏性和效率问题



# 逻辑函数的表达方式

- 1. 电路表示
- 2. 代数表示
- 3. 表格表示
- 4. 图形表示
- 5. 面向计算机的表示



---

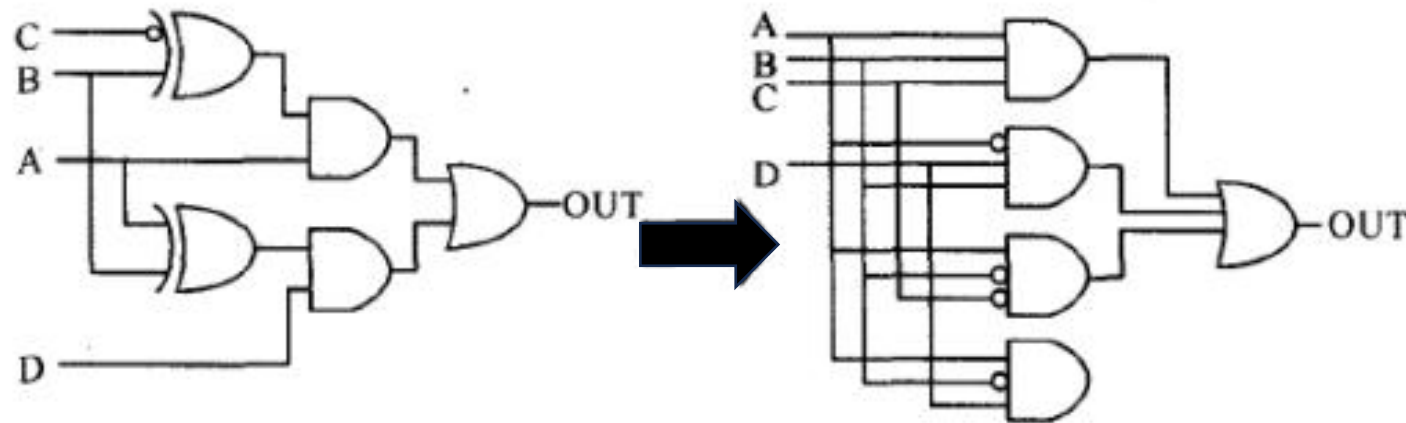
# 两级逻辑优化

Two-level Logic Optimization

---

# 两级逻辑

Two-level logic



$$OUT = A(B \odot C) + D(A \oplus B)$$

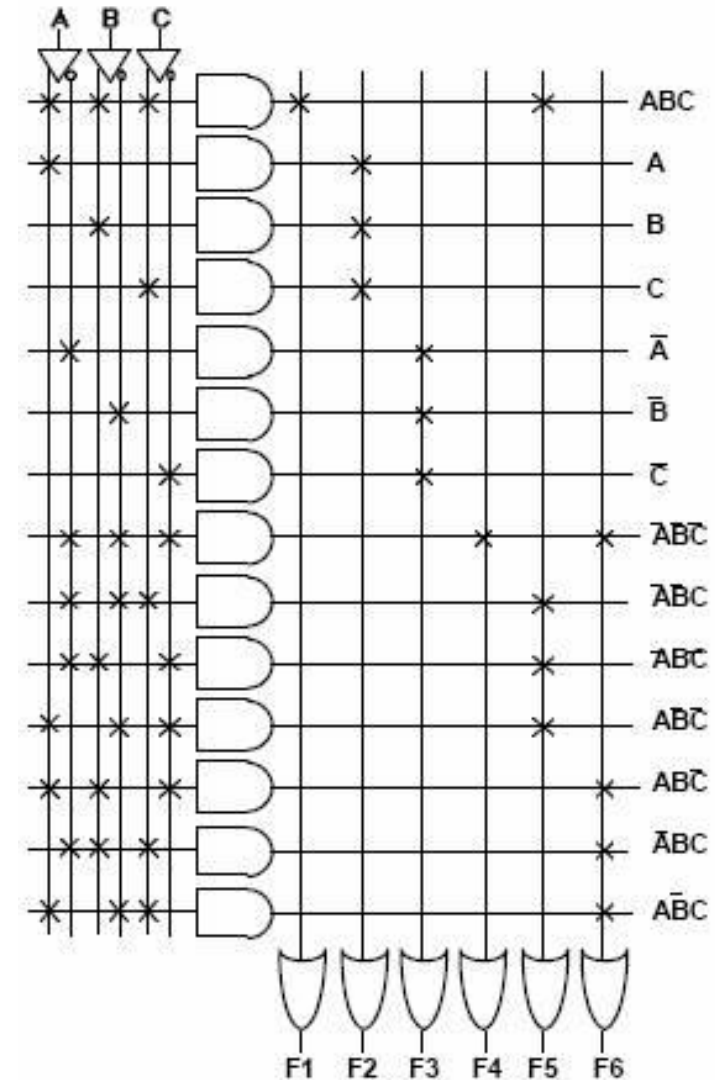
$$OUT = ABC + AB'C' + AB'D + A'BD$$



# 可编程逻辑阵列

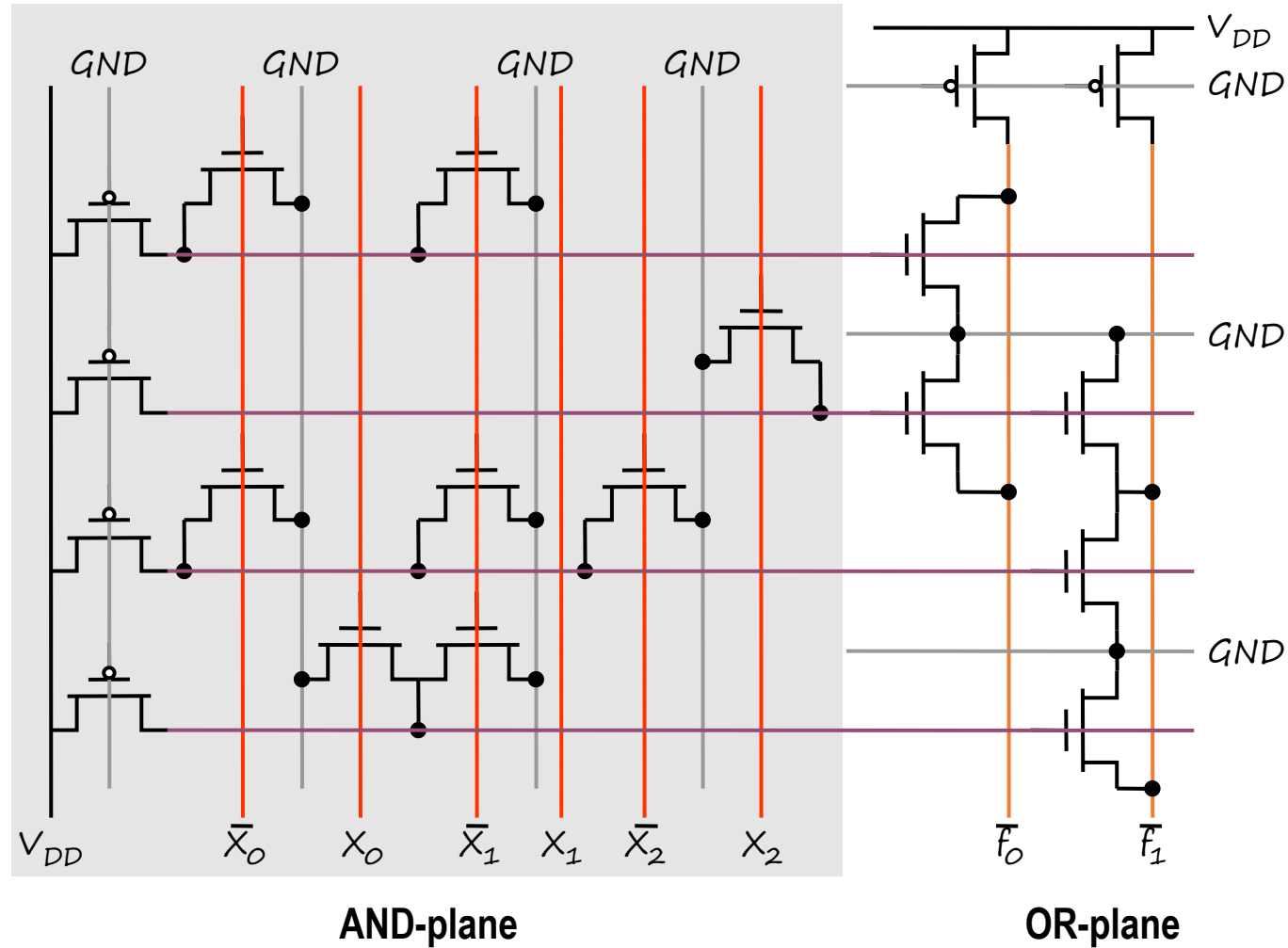
## Programmable logic arrays

- 可编程逻辑阵列（PLA）是一种通过可配置的与阵列和或阵列实现用户自定义逻辑功能的可编程集成电路。
  - 实现任何多输出功能
  - FPGA的前身
  - 由模块生成器生成布局
  - 在七十年代/八十年代相当流行
- 优点
  - 简单，时间可预测
- 缺点
  - 面积可能大，难以支持大规模逻辑设计



# 可编程逻辑阵列

## Programmable logic arrays



# 可编程逻辑阵列

Programmable logic arrays

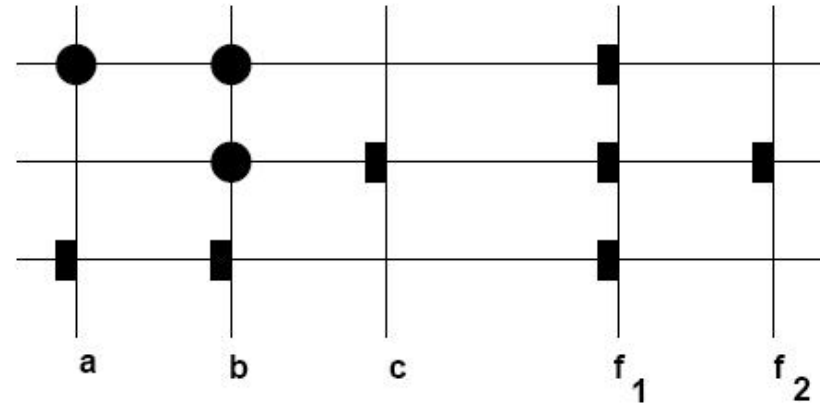
◆  $f_1 = a'b' + b'c + ab$ ;  $f_2 = b'c$

00✕ 10

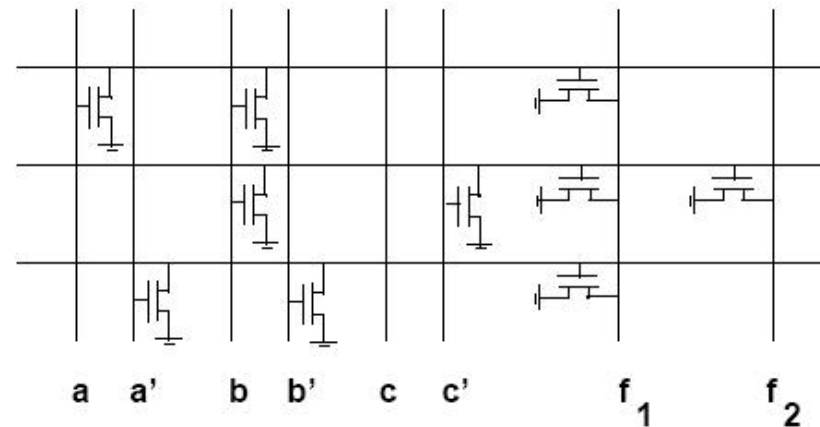
✕01 11

11✕ 10

(a)



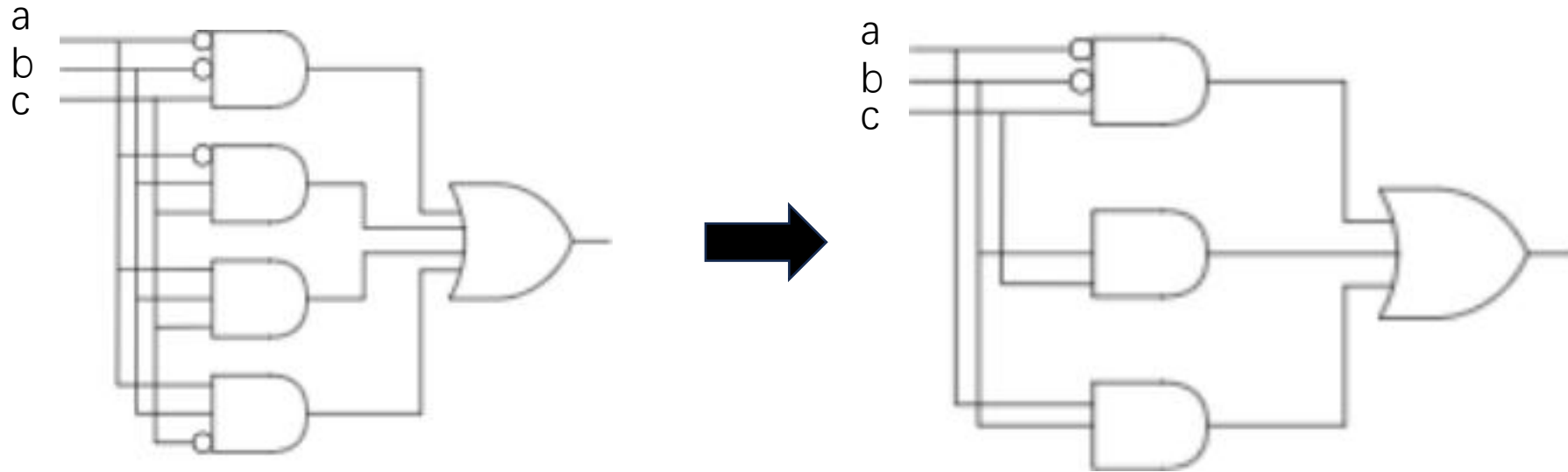
(b)



(c)

# 两级逻辑优化

## Two-level logic optimization

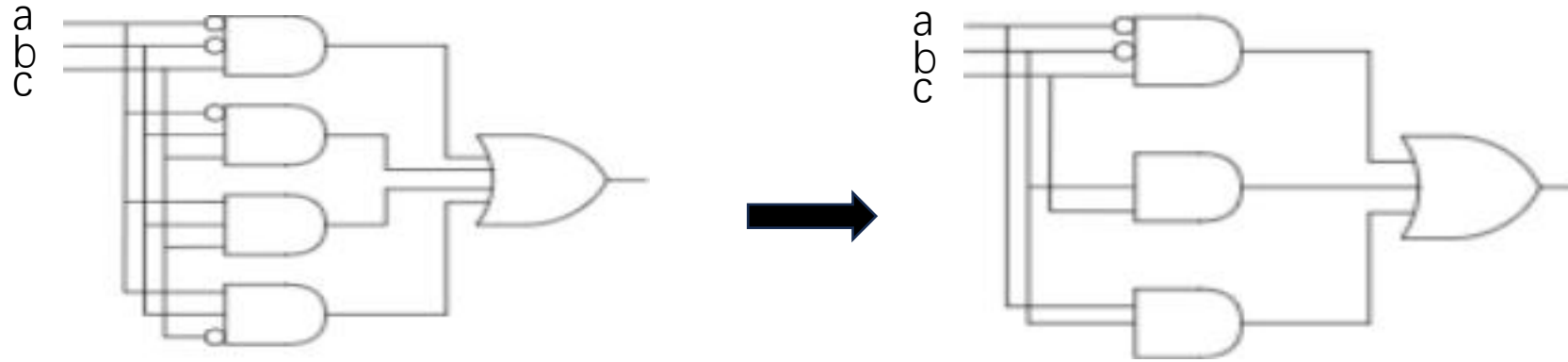


$$x = a'b'c + a'bc + abc + abc'$$

$$x = a'b'c + bc + ab$$

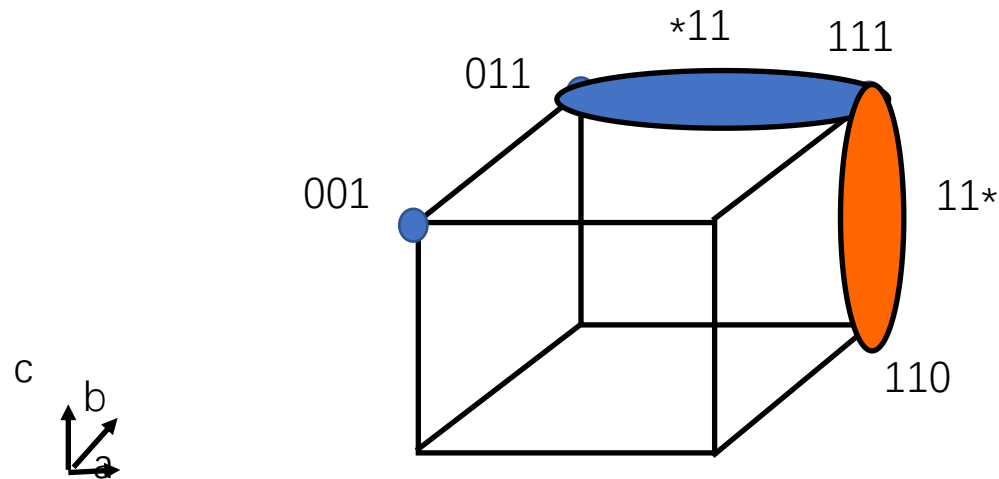
# 两级逻辑优化

## Two-level logic optimization



$$x = a'b'c + a'bc + abc + abc'$$

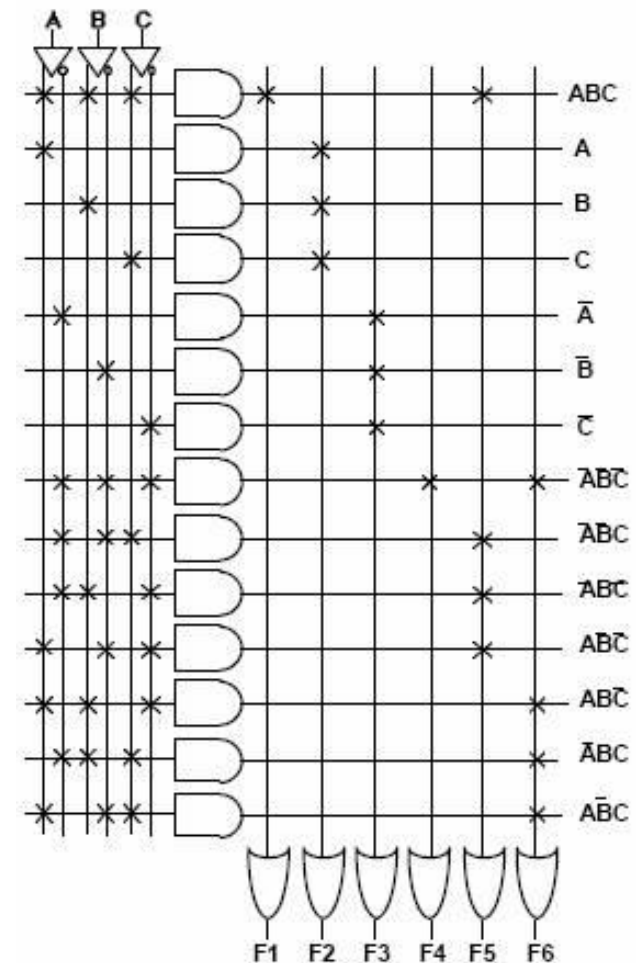
$$x = a'b'c + bc + ab$$



# 两级逻辑优化

## Two-level logic optimization

- 目标：
  - 主要目标是减少乘积（products）的数量，假设所有乘积的成本相同
  - 次要目标是减少布尔文字（literals）的数量
- 基本原理：
  - 乘积对应于PLA行
  - 布尔文字对应于晶体管



# 例子



假设公司里有A、B、C、D四个员工，公司总共需要做5件事情，目前公司有没有冗员？

- A: 125
  - B: 123
  - C: 245
  - D: 124
- 
- 如果裁掉员工C，只保留员工A，B，D，有没有冗员？
  - 使得员工数最少的解法是什么？

# 定义

- 最小覆盖 (Minimum cover) :
  - 具有最少蕴含项 (implications) 的函数覆盖
  - 全局最优
- 非冗余覆盖 (Minimal cover or irredundant cover) :
  - 不完全是另一个函数覆盖的超集
  - 不能删除任何蕴含项
  - 局部最优



# 例子



假设公司里有A、B、C、D四个员工，公司总共需要做5件事情，目前公司有没有冗员？

- A: 125
  - B: 123
  - C: 245
  - D: 124
- 
- 如果裁掉员工C，只保留员工A，B，D，有没有冗员？
  - 使得员工数最少的解法是什么？

# 例子



假设公司里有A、B、C、D四个员工，公司总共需要做5件事情，目前公司有没有冗员？有没有老板的眼中钉？

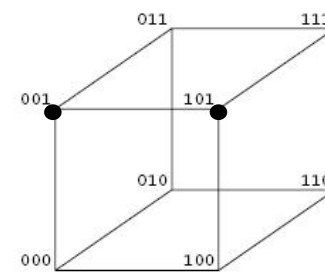
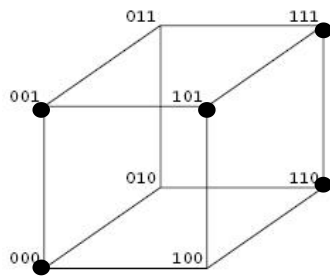
- A: 125
- B: 123
- C: 24
- D: 124

# 定义

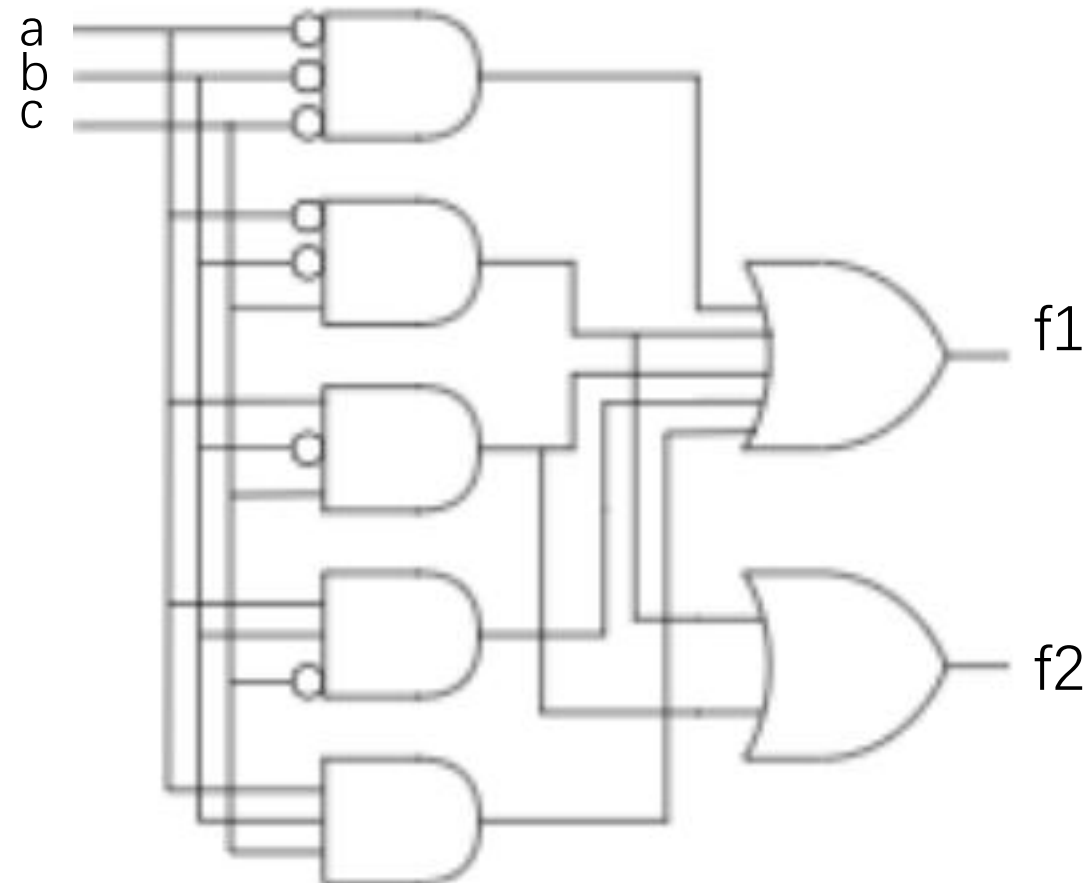
- 最小覆盖 (Minimum cover) :
  - 具有最少蕴含项 (implications) 的函数覆盖
  - 全局最优
- 非冗余覆盖 (Minimal cover or irredundant cover) :
  - 不完全是另一个函数覆盖的超集
  - 不能删除任何蕴含项
  - 局部最优
- 单个蕴含项下的非冗余覆盖 (Minimal w.r.t. single-implicant containment) :
  - 没有蕴含项包含另一个蕴含项
  - 弱局部最优

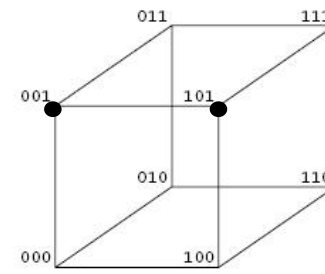
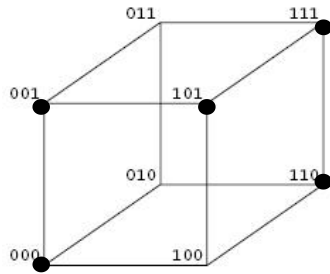
# 定义

- 最小覆盖 (Minimum cover) :
  - 具有最少蕴含项 (implications) 的函数覆盖
  - 全局最优
- 非冗余覆盖 (Minimal cover or irredundant cover) :
  - 不完全是另一个函数覆盖的超集
  - 不能删除任何蕴含项
  - 局部最优
- 单个蕴含项下的非冗余覆盖 (Minimal w.r.t. single-implicant containment) :
  - 没有蕴含项包含另一个蕴含项
  - 弱局部最优



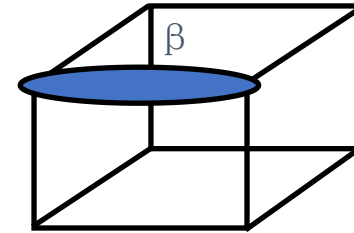
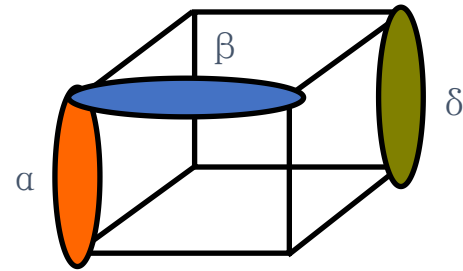
$$f_1 = a'b'c' + a'b'c + ab'c + abc + abc'; \quad f_2 = a'b'c + ab'c$$





$$f_1 = a'b'c' + a'b'c + ab'c + abc + abc'; \quad f_2 = a'b'c + ab'c$$

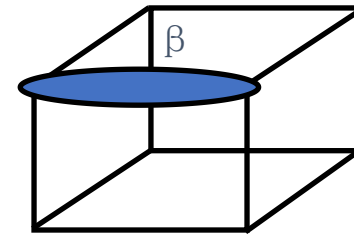
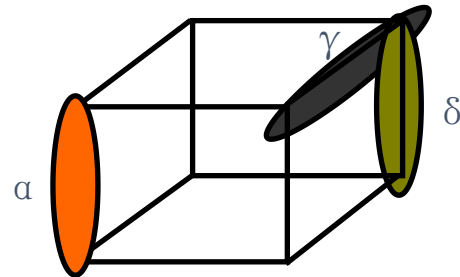
最小覆盖



$$f_1 = a'b' + b'c + ab$$

$$f_2 = b'c$$

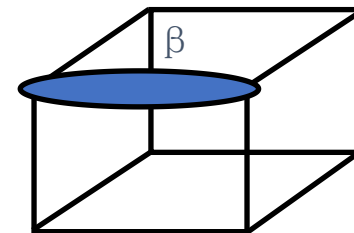
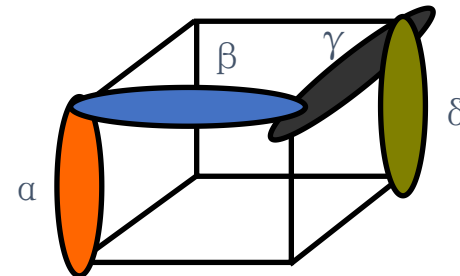
非冗余覆盖



$$f_1 = a'b' + ac + ab$$

$$f_2 = b'c$$

单个蕴含项下的非冗余覆盖

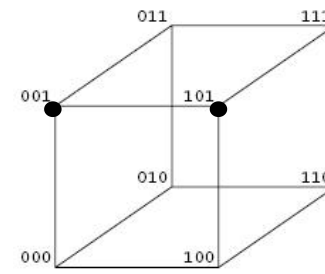
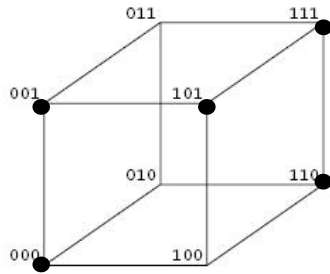


$$f_1 = a'b' + b'c + ac + ab$$

$$f_2 = b'c$$

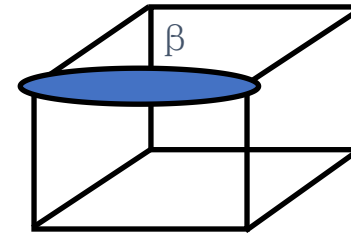
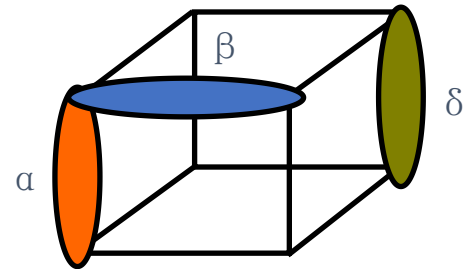
f1

f2

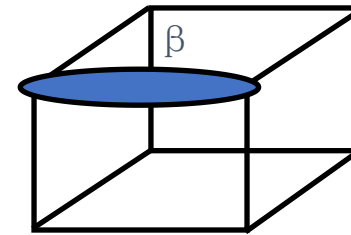
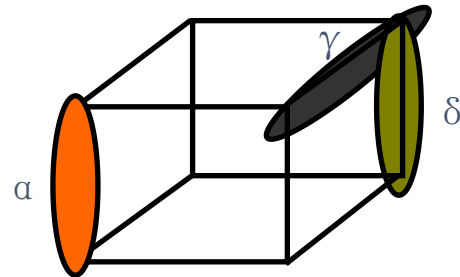


$$f_1 = a'b'c' + a'b'c + ab'c + abc + abc'; \quad f_2 = a'b'c + ab'c$$

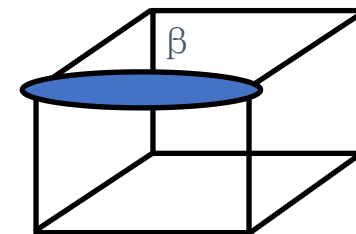
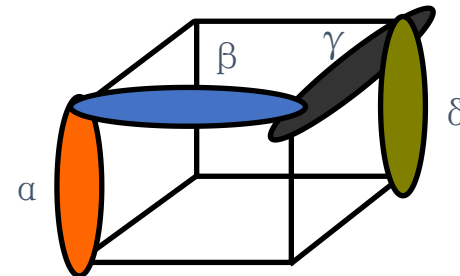
最小覆盖



非冗余覆盖

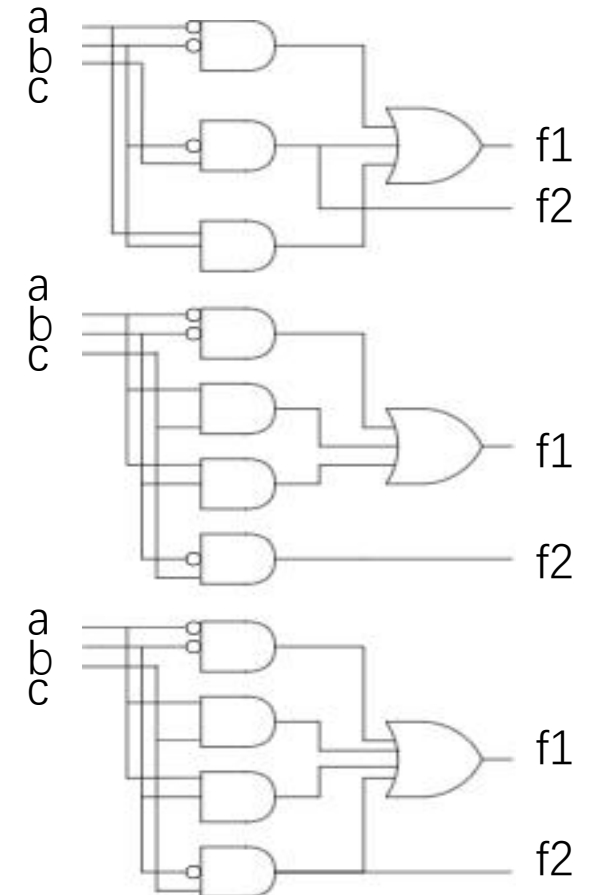


单个蕴含项下的非冗余覆盖



f1

f2



# 两级逻辑优化

## Two-level logic minimization

- 精确逻辑最小化：
  - 目标是计算出一个最小覆盖。对于大型函数是困难的
  - Quine-McCluskey方法（奎因-麦克拉斯基算法）
- 启发式逻辑最小化：
  - 致力于在短时间内计算近似的最小覆盖，这通常足够快速但可能不是最优解。
  - MINI, PRESTO, ESPRESSO





# 精确的逻辑最小化

## Exact logic minimization

- Quine定理：
  - 最小覆盖一定是质覆盖
- 因此：
  - 可以在质蕴含项中搜索最小覆盖
- Quine-McCluskey（奎因-麦克拉斯基算法）方法
  - 计算质蕴含项
  - 确定最小覆盖



# 计算质蕴含项

1. 确认一次蕴含项, 例如:

$$F = AB'CD + A'B'CD' + A'BCD' + ABCD' + AB'CD' + A'BC'D$$

1011

0010

0110

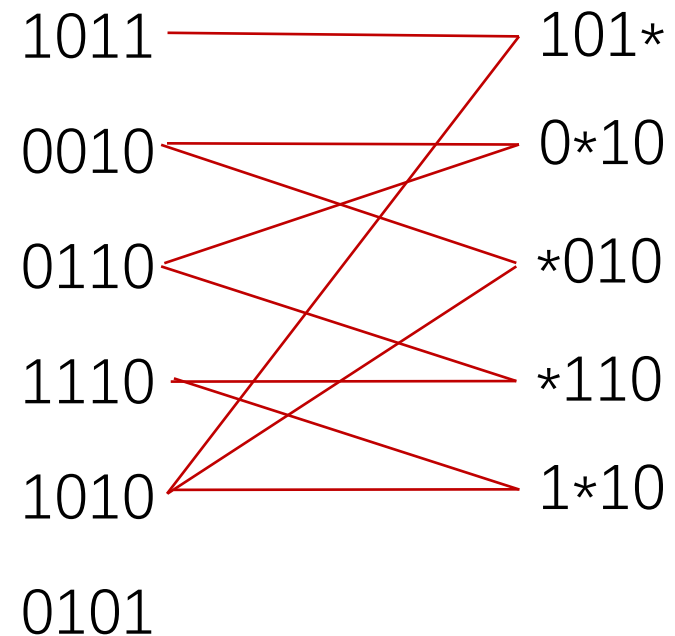
1110

1010

0101

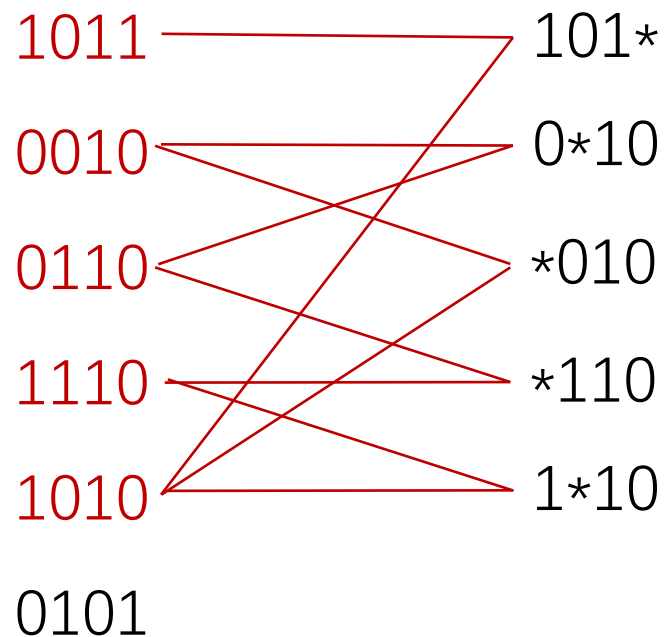
# 计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量



# 计算质蕴含项

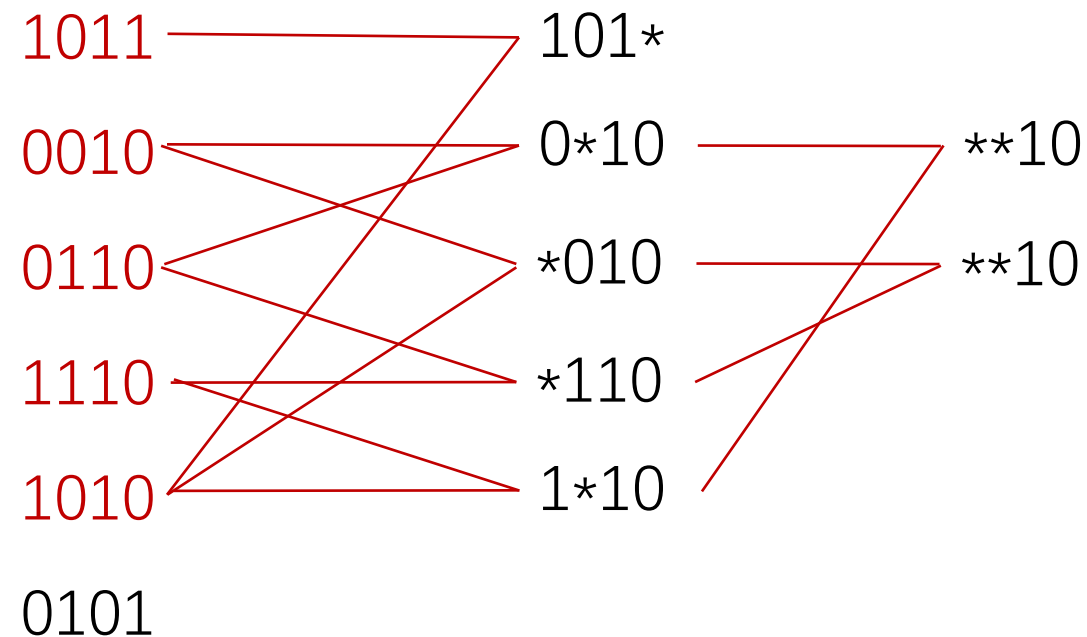
1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项



重复步骤2和步骤3  
直到没有剩余的蕴  
含项可以被合并

# 计算质蕴含项

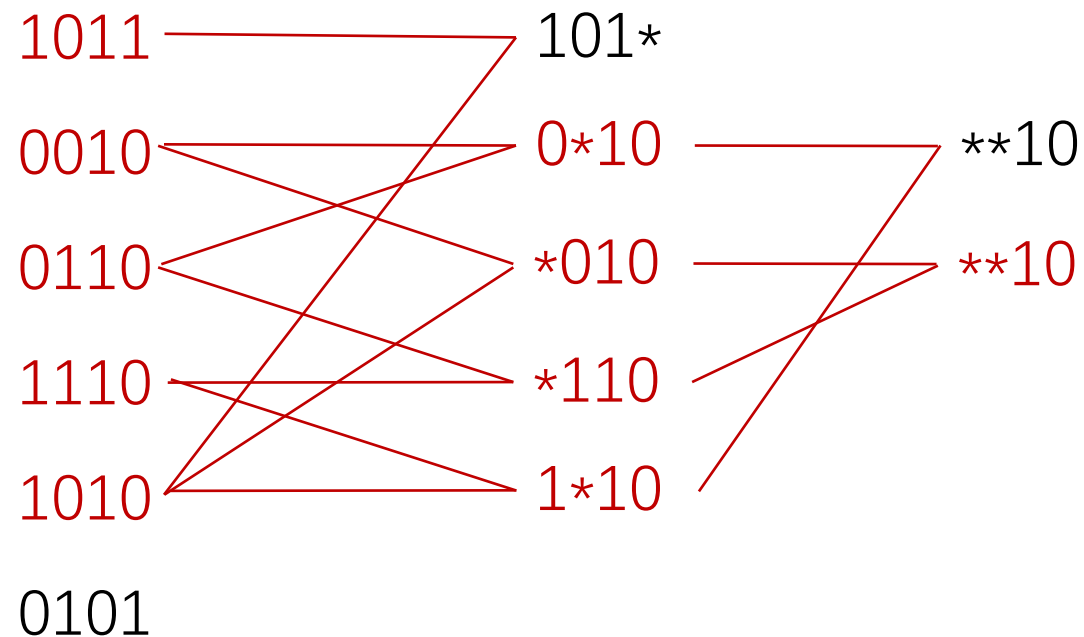
1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项



重复步骤2和步骤3  
直到没有剩余的蕴  
含项可以被合并

# 计算质蕴含项

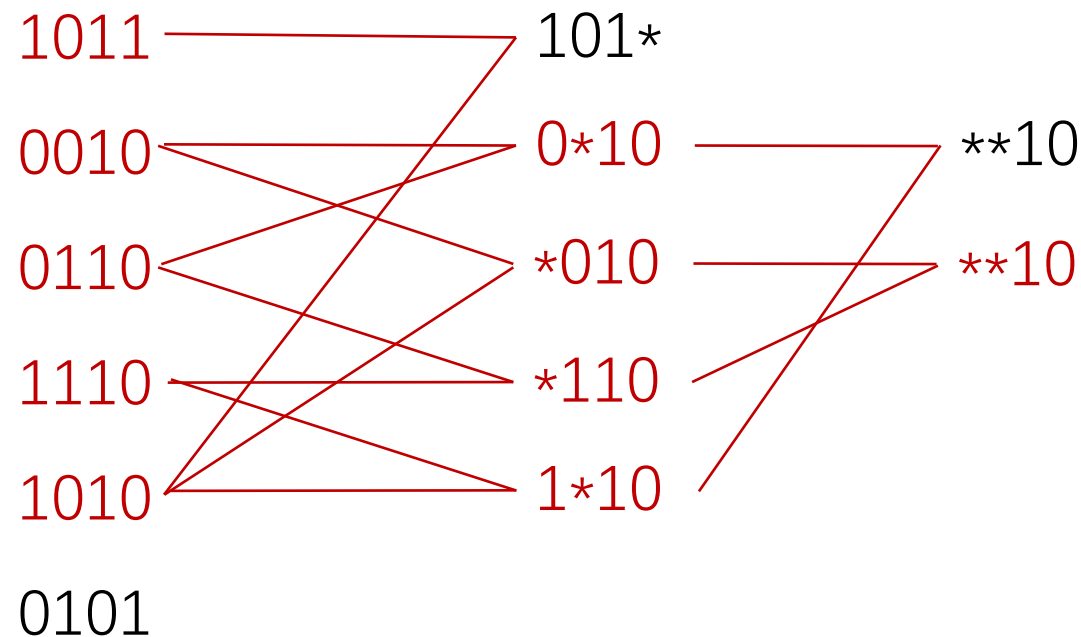
1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项



重复步骤2和步骤3  
直到没有剩余的蕴  
含项可以被合并

# 计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项



重复步骤2和步骤3  
直到没有剩余的蕴  
含项可以被合并

最终得到的质蕴含项：

0101  
101\*  
\*\*10

## 一个小优化-分组



0010

0110

1010

0101

1110

1011



# 随堂作业-先不用交



请计算以下布尔函数的质蕴含项：

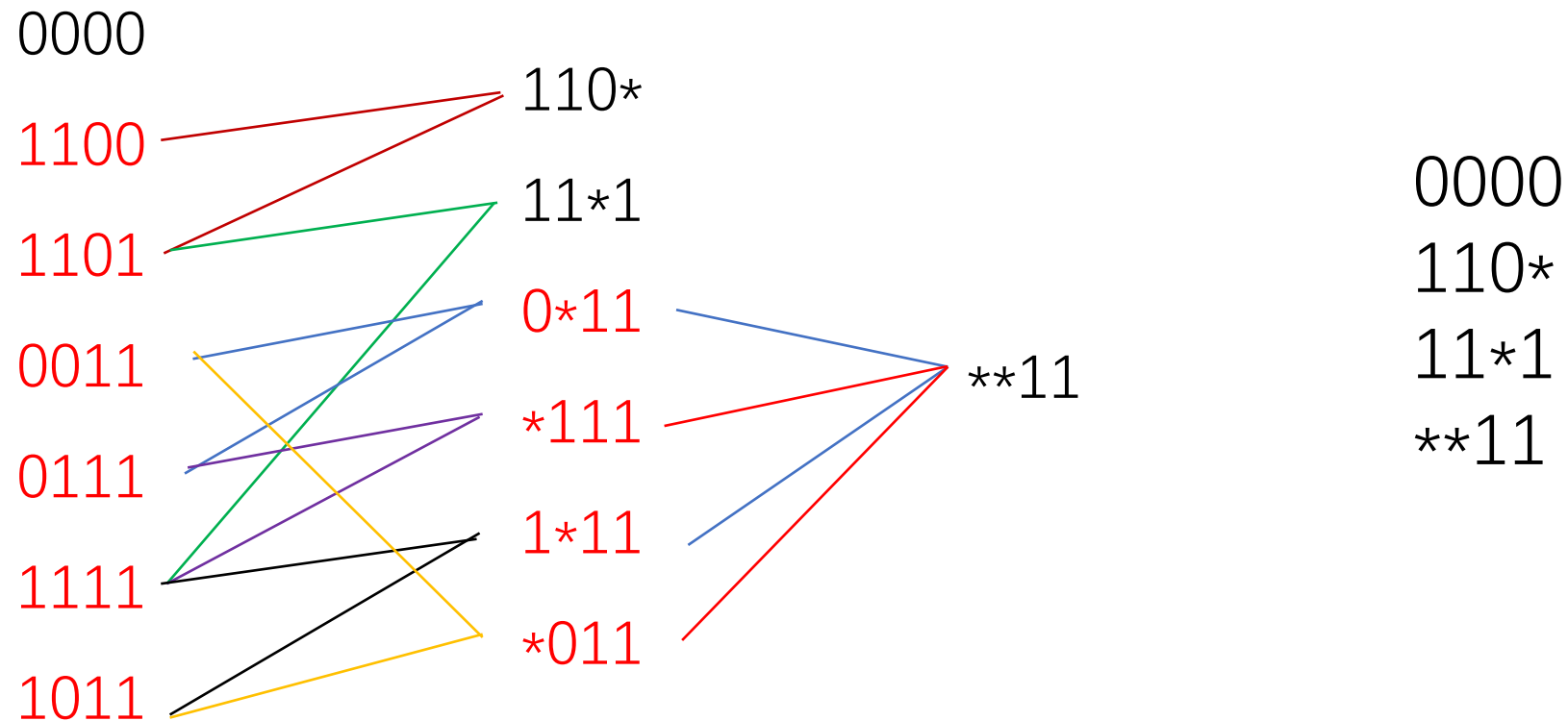
$$F = A'B'C'D' + ABC'D' + ABC'D + A'B'CD + A'BCD + ABCD + AB'CD$$

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项
4. 重复步骤2和步骤3直到没有剩余的蕴含项可以被合并

# 随堂作业-先不用交

请计算以下布尔函数的质蕴含项：

$$F = A'B'C'D' + ABC'D' + ABC'D + A'B'CD + A'BCD + ABCD + AB'CD$$





# 最低覆盖率的早期方法

## Minimum cover early methods

- 简化表格
  - 迭代地识别必要元素，将它们保存在覆盖中。
  - 移除已覆盖的最小项。
- Petrick 方法
  - 以积之和 (POS) 形式写覆盖子句
  - 将积之和形式展开为和之积 (SOP) 的形式
  - 选择最小的立方项
  - 注意：展开子句的成本是指数级的



# 质蕴含项表

## Prime implicant table

- 质蕴含项表是一个二进制矩阵表格，表格的：
  - 列 表示所有的质蕴含项
  - 行 表示所有需要被覆盖的最小项
- 在表格中用 ✓ （或 1） 标记哪些质蕴含项可以覆盖哪些最小项。

# 例子

## Example

- $f = a'b'c' + a'b'c + ab'c + abc + abc'$

- 蕴含项表:

	abc	f
$\alpha$	00*	1
$\beta$	*01	1
$\gamma$	1*1	1
$\delta$	11*	1

- 质蕴含项表:

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

## 随堂作业(续)-先不用交

请列出以下布尔函数的质蕴含项表：

$$F=A'B'C'D'+ABC'D'+ABC'D+A'B'CD+A'BCD+ABCD+AB'CD$$

- 质蕴含项表是一个二进制矩阵表格，表格的：
  - 列 表示所有的质蕴含项
  - 行 表示所有需要被覆盖的最小项
- 在表格中用 ✓ （或 1） 标记哪些质蕴含项可以覆盖哪些最小项。



## 通过简化表格确定最小覆盖

1. 对所有只包含一项“1”的行（图中的红色标记），对其项“1”所在的列画竖线，并对竖线所经过的项“1”画横线

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1



# 通过简化表格确定最小覆盖

2. 找出为包含最大数量未被划线的项“1”的列，若有多列，则必有多种化简结果  
对其划竖线，并对当前项1及竖线所经过的项“1”画横线

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

## 通过简化表格确定最小覆盖

3. 若经过划线后还存在未划线项，继续步骤2直到无未划线项为止，最后的化简结果（最小覆盖）即为所有划竖线的列对应的项的和

	$\alpha(a'b')$	$\beta(b'c)$	$\gamma(ac)$	$\delta(ab)$
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$f = a'b' + b'c + ab$$

## 随堂作业(续)



请确定以下布尔函数的最小覆盖：

$$F=A'B'C'D'+ABC'D'+ABC'D+A'B'CD+A'BCD+ABCD+AB'CD$$

1. 对所有只包含一项“1”的行，对其项“1”所在的列画竖线，并对竖线所经过的项“1”画横线
2. 找出为包含最大数量未被划线的项1的列，若有多列，则必有多种化简结果对其划竖线，并对当前项1及竖线所经过的项1画横线
3. 若经过划线后还存在未划线项，继续步骤2直到无未划线项为止，最后的化简结果（最小覆盖）即为所有划竖线的列对应的项的和

# 随堂作业-结果

	A'B'C'D'	ABC'	ABD	CD
0000	1	0	0	0
1100	0	1	0	0
1101	0	1	1	0
0011	0	0	0	1
0111	0	0	0	1
1111	0	0	1	1
1011	0	0	0	1

$$F=A'B'C'D'+ABC'+CD$$