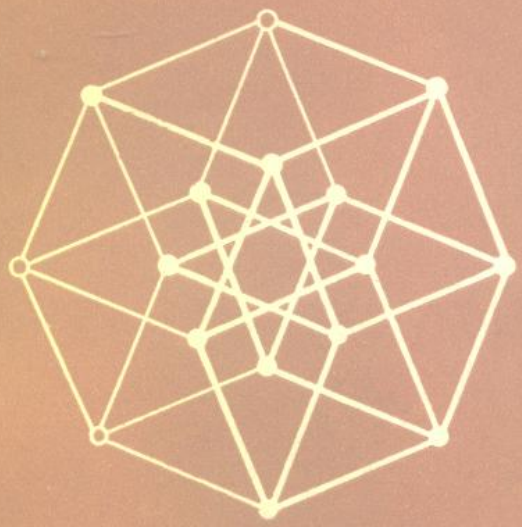


高等学校教学参考书

计算机辅助逻辑综合

沈嗣昌 编

高等教育出版社

高等学校教学参考书

计算机辅助逻辑综合

沈嗣昌 编

高等教育出版社

内 容 提 要

本书是数字系统逻辑设计自动化方面的教学参考书。重点讨论了计算机辅助逻辑综合的理论及方法。

全书共五章。主要内容有：立方符号法及立方运算；单输出函数的综合；多输出函数的覆盖；多级电路的综合；同步时序电路的综合。各章介绍了有关的主要算法，并有例题辅以说明。各章末还给出了实现主要算法的 FORTRAN IV 子程序。本书重点较突出，概念清楚，叙述简明，且便于自学。

本书可作为逻辑设计课程的选修教材，也可供有关专业师生及工程技术人员参考。

高等学校教学参考书

计算机辅助逻辑综合

沈嗣昌 编

*

高等教育出版社出版

新华书店北京发行所发行

常明大同红卫印刷厂印刷

*

开本 787×1092 1/16　印张 14.5　字数 330,000

1982年7月第1版　1983年9月第1次印刷

印数 00,001—9,500

书号 15010·0419　定价 1.85 元

前　言

计算机辅助逻辑综合是数字系统逻辑设计自动化的一个环节。

自动逻辑设计大致可以分成功能描述、逻辑综合和逻辑模拟三个阶段。用 FORTRAN 等高级语言或某种专门的逻辑设计语言描述数字系统或部件的逻辑功能，这称为**功能描述**。**逻辑综合**的任务是：根据编译程序提供的原始数据以及选定的器件，导出就某个价格标准来说较为经济的逻辑网络。用计算机检验上述逻辑网络是否满足预期的逻辑功能，这是**逻辑模拟**的主要目的。本书将集中讨论自动逻辑综合的理论和方法。

逻辑综合包括组合函数综合和时序函数综合。

用表格、卡诺图和逻辑方程描述组合函数不但在手工综合时是十分有用的，而且也是研究自动综合算法的基础。在计算机辅助组合函数综合时，广泛采用立方符号法描述组合函数，并用一系列的立方运算对函数进行变换，以达到预期的综合的目的。本书的第一章将介绍立方符号法、立方运算以及如何用计算机来实现这些运算。第二、第三章将依次讨论综合单输出函数和多输出函数的计算机算法。第四章的题目是多级电路综合。由于实际使用的**与非**门的扇入、扇出都是有限制的，在这一章里将主要讨论如何用这样的**与非**门实现给定的逻辑函数。用一种或多种逻辑模件实现函数有时可以极大地降低电路的成本并提高电路的可靠性。函数分解技术是综合这类电路的一种有用的工具。本章将简要地介绍这种技术。

状态化简和状态分配是综合同步时序电路的两个主要课题。第五章将以较大的篇幅予以讨论。本书将不具体讨论异步时序电路的综合问题。

在每一章里都给出了实现主要算法的 FORTRAN IV 子程序，以使读者能对机助逻辑综合有一个完整的概念。

本书原是南京航空学院四系的选修课教材。在编写时考虑了与当时的数字电路教材的衔接问题，并力求深入浅出，便于自学。编者也希望这本教材能对从事逻辑设计的实际工作的人员有所帮助。

清华大学计算机系何全来、薛熙恩和黎达三位老师审阅了本书，并提出了许多宝贵的修改意见。南京航空学院四系的邵有信老师仔细地校阅了本书的原稿，并参加了程序的调试。在此向上述诸位老师表示衷心的感谢。

由于编者水平有限，书中不免有不妥甚至错误之处，望读者批评指正。

编　者

1982.3.

目　录

第一章　立方符号法及立方运算

以往，我们都习惯于用真值表、布尔方程及卡诺图来描述组合逻辑函数。这些方法对分析、综合逻辑电路是十分有用的。

大多数机助逻辑综合算法都用立方符号法来描述组合逻辑函数。这种方法的核心是把组合逻辑函数看作 n- 维空间中的立方的集合，从而用一系列的立方运算对描述该函数的立方的集合进行变换，以达到预期的逻辑综合的目的。这种方法与前面提到的其他几种方法相比，在本质上并无多大差别。但是利用立方符号法可以很方便地把函数的真值表中的一行或布尔方程中的一项装在计算机的一个或几个字内。

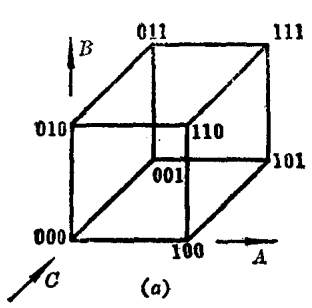
本章将先介绍立方符号法，并导出一些基本的立方运算，为第二章开始的逻辑综合打下基础。为简单起见，本章仅考虑单输出函数的情况。

§1-1　立方符号法

既然立方符号法是常用的一种描述组合逻辑函数的方法，那么什么叫**立方符号法**？它与其他描述组合逻辑函数的方法之间有什么关系？这是我们首先要解决的问题。

一、空间图形及函数阵列

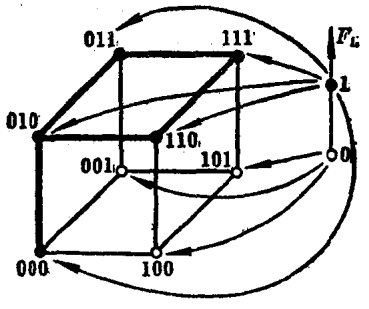
设函数 $y=f(x_1, x_2, \cdots, x_n)$，其中自变量 $x_1, x_2, \cdots, x_n$ 称为**输入变量**，因变量 y 称为**输出变量**。对这样一个有 n 个输入变量（简称 n **变量**）的函数来说，若 $x_1, x_2, \cdots, x_n$ 及 y 均为模拟量，当然可以用 n- 维空间中的几何图形来描述这个函数。现在 $x_1, x_2, \cdots, x_n$ 及 y 均为二值的逻辑变量，那么怎样在 n- 维空间中用几何图形来描述这个逻辑函数呢？为此，我们也使一个输入逻辑变量与空间的一根坐标轴相对应。这样，对于一个三变量函数 $f(A, B, C)$ 来说，就对应了一个 3- 维空间。由于一个逻辑变量只取 **0**、**1** 两种可能的值，三个输入变量的取值仅有八种可能的组合，它们对应了上述的 3- 维空间中的八个点。这八个点恰好是一个 **3- 维立方体**（简称 **3- 维体**或 **3- 立方**）的八个顶点。图 1-1(a) 说明了空间坐标与输入变量之间的对应关系，并标出了这个 3- 维体的八个顶点的坐标。如果函数 f 的输入组合与顶点 P 对应，且相应的函数值为 **1**，则称顶点 P 为函数 f 的**真顶点**或 ***ON* 顶点**；如果函数值为 **0**，则称相应的顶点为 f 的假顶点或 ***OFF* 顶点**。设函数 F_1 的真值表如图 1-1(b) 所示。若用实心点表示真顶点，空心点表示假顶点，那么函数 F_1 就可以用图 1-1(c) 所示的几何图形来表示。由图可见，函数 F_1 在 3- 维空间内有五个真顶点、三个假顶点。在这种情况下也可以用它在空间占有的真顶点的集合



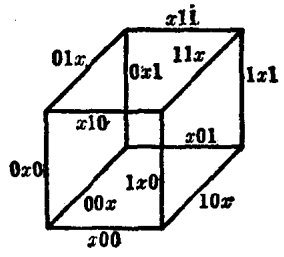
(a)

A	B	C	F_1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

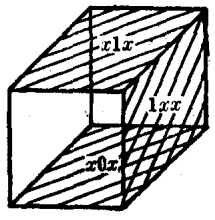
(b)



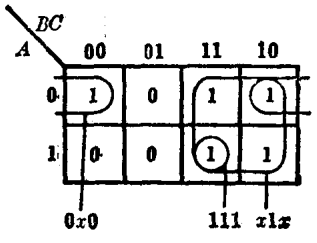
(c)



(d)



(e)



(f)

图 1-1　逻辑函数的立方符号

$$ON(F_1)=\left\{\begin{matrix}0&0&0\\0&1&0\\0&1&1\\1&1&0\\1&1&1\end{matrix}\right\} \tag{1-1}$$

来描述这个函数。

对于图 1-1(a) 中的 3- 维体的任意一条棱，都是由两个(2^1 个)仅有一个坐标不同的顶点组成。例如顶点 **011** 和 **111** 仅第一个坐标不同，它们组成了一条棱，记作 x**11**。3- 维体的十二条棱如图 1-1(d)所示，从而图 1-1(b)所示的函数也可以用棱的集合

$$ON(F_1)=\left\{\begin{matrix}x&1&1\\x&1&0\\0&x&0\end{matrix}\right\} \tag{1-2}$$

来表示。

同样，图 1-1(a)中的 3- 维体的任意一个面，都可以看成是由两条只有一个坐标不同的棱

组成的。例如，$x\mathbf{10}$ 和 $x\mathbf{11}$ 这两条棱可构成一个面，记作 $x\mathbf{1}x$，如图 1-1(*e*)所示。由此，函数 F_1 又可表示成面和棱的集合

$$ON(F_1)=\left\{\begin{matrix}x & \mathbf{1} & x\\ \mathbf{0} & x & \mathbf{0}\end{matrix}\right\} \tag{1-3}$$

一个面实际上是一个 2- 维体，一条棱是一个 1- 维体，分别称它们为**2- 立方**和**1- 立方**，相应地包含 2^2、2^1 个顶点。由此，空间的一个顶点可称为**0- 立方**，它只包含了 2^0 个顶点。空间的一个 r**- 立方**则包含了 2^r 个顶点。

前面我们说 F_1 可用顶点的集合、棱的集合、棱和面的集合来表示，现在我们说它可以用立方的集合来表示。由此推广到更为普遍的情况：任何一个 n 变量的逻辑函数 $y=f(x_1, x_2, \cdots, x_n)$ 都可以用它在 n- 维空间中占有的立方的集合来表示：

$$T(y)=\{t^1, t^2, \cdots, t^p\} \tag{1-4}$$

其中 $t^i=t_1^i\ t_2^i\cdots t_n^i$ 是 n- 维空间中的一个 r- 立方，并且称 $t_j^i\in\{\mathbf{0}, \mathbf{1}, x\}$ 是立方 t^i 的一个**组元**。n 变量函数的每一个立方都由 n 个组元组成。若四变量函数的一个立方 $t=\mathbf{10}x\,x$，则组成它的四个组元分别是 $t_1=\mathbf{1}$、$t_2=\mathbf{0}$、$t_3=x$、$t_4=x$。

由式(1-4)所描述的立方的集合通常都写成阵列的形式，如式(1-1)、(1-2)、(1-3)那样。因此也把表示给定逻辑函数 f 的立方集合简称为**函数的阵列**或**阵列**。

由函数的 ON 顶点组成的立方的集合称为**函数的 *ON* 阵列**。由函数的 OFF 顶点组成的立方的集合称为**函数的 *OFF* 阵列**。如果函数是未完全规定的，则与真值表中函数值未加规定的输入组合对应的顶点被称为 ***DC* 顶点**。由 DC 顶点组成的立方的集合称为**函数的 *DC* 阵列**。图 1-1(*b*)所示的函数是一个完全规定的函数，可以认为

$$OFF(F_1)=\left\{\begin{matrix}\mathbf{0} & \mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{0} & \mathbf{0}\\ \mathbf{1} & \mathbf{0} & \mathbf{1}\end{matrix}\right\}=\left\{\begin{matrix}x & \mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{0} & x\end{matrix}\right\} \tag{1-5}$$

$$DC(F_1)=\varnothing$$

这里我们用 $\varnothing$ 表示**空集**。

如果可以写出函数的 ON、OFF、DC 三个阵列中的两个，那么，我们就完全地描述了这个函数。

由于任何立方都是由一串 **0**、**1**、x 的符号组成的，故称这种表示逻辑函数的方法为**立方符号法**。

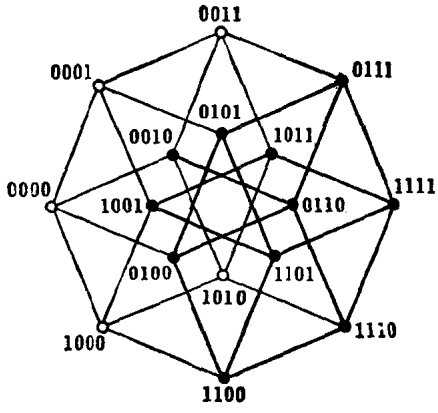
例

设函数 F_2 的空间图形如图 1-2(*a*)所示，它可以用如下的立方集合或阵列来描述：

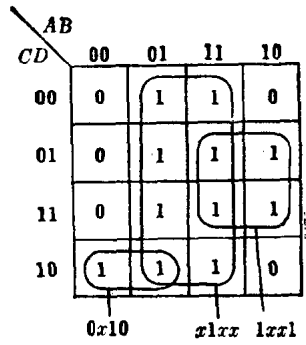
$$ON(F_2)=\left\{\begin{matrix}\mathbf{0} & x & \mathbf{1} & \mathbf{0}\\ \mathbf{1} & x & x & \mathbf{1}\\ x & \mathbf{1} & x & x\end{matrix}\right\} \tag{1-6}$$

$$DC(F_2)=\varnothing$$

其中 $\mathbf{0}x\mathbf{10}$ 是一个 1- 立方，它由 **0010**、**0110** 两个顶点组成；$\mathbf{1}x\,x\mathbf{1}$ 是一个 2- 立方，它由



(a)



(b)

图 1-2　F_2 在4-维空间占有的顶点及其卡诺图

1001、1011、1101、1111 四个顶点组成；$x\,1\,x\,x$ 是一个 3- 立方，由八个顶点组成。立方符号中的 x 的个数就是这个**立方的维数**。

二、0- 复合体与真值表

同一个函数，描述它的函数阵列可能会有许多种不同的形式。但是这个函数在空间占有的顶点是唯一的。对于函数 F_1，它的 ON 阵列可以写成许多不同的形式，式 (1-1)、(1-2)、(1-3) 只是其中的三种形式。但 F_1 在空间的图形即占有的顶点则唯一地如图 1-1(*a*) 所示。今后我们称函数在空间占有的顶点的集合为**函数的 0- 复合体**，记作 K^0。如果这个 0- 复合体由函数的 ON 顶点组成，则称它为**函数的真顶点的 0- 复合体**，记作 $K^0(ON)$，相应地称 $K^0(OFF)$、$K^0(DC)$ 为**函数的假顶点、*DC* 顶点的 0- 复合体**。对于图 1-1(*b*) 所示的函数来说，

$$K^0(ON)=\begin{Bmatrix}0&0&0\\0&1&0\\0&1&1\\1&1&0\\1&1&1\end{Bmatrix} \tag{1-7}$$

$$K^0(OFF)=\begin{Bmatrix}0&0&1\\1&0&0\\1&0&1\end{Bmatrix}$$

$$K^0(DC)=\varnothing$$

显然，0- 复合体与真值表有严格的对应关系。真值表中的一行，随函数值不同而与 $K^0(ON)$ 或 $K^0(OFF)$ 或 $K^0(DC)$ 中的一个元素对应。可以认为 0- 复合体实际上就是写成 0- 立方的集合的形式的真值表。

0- 复合体与函数阵列的不同之处在于前者是顶点的集合，而后者是立方的集合，它的元素可以是任意 r- 立方，$0\leqslant r\leqslant n$。如果函数的 ON 阵列中的每一个元素都是 0- 立方，则它在形式上正好与 $K^0(ON)$ 相同。

三、n- 维空间与卡诺图

图 1-1(b)所示函数的卡诺图如图 1-1(f)所示。对照图 1-1(c)及 1-1(f)可见，空间的一个顶点对应了卡诺图中的一个小方格。顶点的坐标，即 0- 立方的立方符号就是这个小方格的二进制代码。如果该顶点为真顶点，则对应的小方格就填有 **1**；如是假顶点，就填有 **0**。空间的一个 1- 立方对应了图中的一对相邻的小方格。该 1- 立方的立方符号中的一个 x 表示了这一对相邻的小方格的两个二进制代码的不同之处。同样，空间的一个 2- 立方对应了两对相邻的小方格。这个 2- 立方中的两个 x 表示了这两对小方格的二进制代码的不同之处。例如

立　方	**1**	**1**	**1**	**0**	x	**0**	x	**1**	x
					↕		↕		↕
							0	**1**	**0**
小方格	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
				0	**1**	**0**	**1**	**1**	**0**
							1	**1**	**1**
积　项	A	B	C	$\overline{A}$		$\overline{C}$		B	

依此类推，对于空间的任一 r- 立方，均可在卡诺图中找到与它对应的一组小方格。同样，卡诺图中满足一定条件的一组小方格，也必对应了空间的一个立方。因此，可以认为：n 变量的卡诺图实质上就是 n- 维空间的平面表示。

以后我们经常用卡诺图来表示立方在空间的位置及它所占有的顶点，而不去画复杂的立体图形。

四、函数阵列与布尔表达式

函数阵列与函数的一个积之和表达式相对应，这一点是很显然的。阵列中的一个 r- 立方对应了卡诺图中的一组小方格，后者对应了积之和表达式中的一个积项。反过来，函数的积之和表达式对应了函数的阵列。

如果已知函数的积之和表达式，我们只要把式中的各个积项都用一个 r- 立方来表示，并写成阵列的形式，即得函数的阵列。把积项写成立方符号的方法是：与积项中原变量对应的组元取立方符号 **1**，与补变量对应的组元取立方符号 **0**，与未出现的变量对应的组元取立方符号 x，并把全部符号按变量的次序排好。

例

函数 F_3 的积之和表达式为

$$F_3 = ABCD + \overline{A}C\overline{D} + \overline{B}\overline{C}$$

则对应的函数阵列为

$$T(F_3) = \begin{Bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & x & \mathbf{1} & \mathbf{0} \\ x & \mathbf{0} & \mathbf{0} & x \end{Bmatrix}$$

必须注意，对于 n 变量的函数，组成某个积项的变量数可能少于 n 个，但与它对应的 r-立方中的组元数必须是 n 个。在书写这个积项时，变量的先后次序是任意的，但与它对应的 r-立方中的各组元必须按一定的次序排列。

五、立方符号的代码

前已说明，任一逻辑函数都可用阵列来表示。现在进一步说明怎样在计算机的内存中表示这个阵列。

任何立方都是由一串 **0**、**1**、x 的符号组成的。要在计算机的内存中表示立方 t，例如 $t=x\,\mathbf{1}\,\mathbf{0}\,x\,\mathbf{1}$，首先要对它的每一个组元进行编码。最为直截了当的方法是用一个十进制数表示一个符号。例如，用十进制数 1 表示符号 "**1**"，用十进制数 0 表示符号 "**0**"，用 2 表示符号 "x"，并把立方的每一个组元存放在计算机的一个字内。这样一来，立方 $x\,\mathbf{1}\,\mathbf{0}\,x\,\mathbf{1}$ 就可用一个一维数组 (2, 1, 0, 2, 1) 来表示。但是这种做法显然是太浪费了。因为，对一个 n 变量的函数来说，为表示一个立方就要占有 n 个字。如果函数阵列有 m 个立方，则仅为存入这些数据就占用了 $n\cdot m$ 个字。因此，比较常用的方法是对每个符号进行二进制编码，并把每一个立方装在计算机的一个或几个字内。这里我们介绍两种大体相同的编码方法。

如果我们用代码 **01** 表示符号 "**0**"，**10** 表示 "**1**"，**11** 表示 "x"，则立方 $x\,\mathbf{1}\,\mathbf{0}\,x\,\mathbf{1}$ 就可以表示为 **1110011110**。如果计算机的字长为 B，那么变量数 $n\leqslant\frac{B}{2}$ 的立方就可以装进一个字内。如果 $n>\frac{B}{2}$，则需要 $\lceil(n-1)/(B/2)\rceil+1$ 个字才能存贮一个立方。$\lceil a\rceil$ 表示取小于等于 a 的最大整数。另一方面，如果变量数较少，例如只有八个，而机器的字长却有 32，按理说可以在一个字内存贮两个立方，但是这样做将会使计算复杂化。因此，我们最多在一个字内装一个立方。为简化我们的讨论起见，今后只考虑 $n\leqslant\frac{B}{2}$，即一个立方装在一个字内的情况。这样，对于有 m 个立方的阵列 A 来说，可以用一个长度为 m 的一维数组来表示。如果机器的字长为 32，且

$$A=\left\{\begin{matrix} x & 1 & 0 & x & 0 & 0 & x & 1 \\ x & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & x & 1 & x & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & x & 0 & 1 & 1 & 1 \\ 1 & x & x & 1 & x & x & 1 & 0 \end{matrix}\right\}$$

则内存中相应单元的状态应为

```
00 00 00 00 00 00 00 00 11 10 01 11 01 01 11 10
00 00 00 00 00 00 00 00 11 10 10 10 10 01 10 01
00 00 00 00 00 00 00 00 01 11 10 11 10 10 01 10
00 00 00 00 00 00 00 00 10 01 10 11 01 10 10 10
00 00 00 00 00 00 00 00 10 11 11 10 11 11 10 01
```

用二进制代码表示立方的另一种方法如表 1-1 所示，u 和 v 分别代表计算机的两个字。

表 1-1　立方符号的另一种二进制代码

组元 / 代码	t_i		
	x	1	0
u_i	0	1	0
v_i	0	1	1

如果立方 $t=x\,1\,0\,x\,1$，则相应的字内所存的二进制代码为

$$u=01001$$

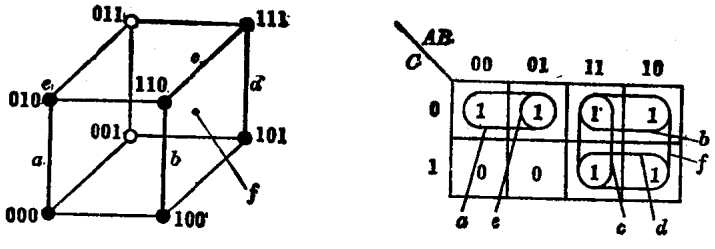
$$v=01101$$

在采用这种代码时，若 $n\leqslant B$，则每一立方占两个字。对于大多数设计题目来说，$n\leqslant B$ 这个条件是可以满足的，故每一立方仅占两个字。

为以后叙述方便起见，我们简称前一种方法为第一种代码，后一种方法为第二种代码。要说明的一点是，立方符号法只是用立方的概念描述组合逻辑函数的许多方法中的常用的一种。由于表示立方的方法不同，相应的代码也就不同。下面，我们仅以立方符号法及上述两种代码为例，来叙述立方运算及如何在计算机上实现这些运算。

§1-2　基本的立方运算

在图 1-3(a)中，画出了在空间处于不同位置的几个立方，图 1-3(b)为相应的卡诺图。由于各个立方在空间的位置不同，使它们两两间的相互关系也不相同。下面讨论这些不同的关系所具有的不同的性质及相应的立方运算。



(a)　　(b)

$a=0\,x\,0$, $b=1\,x\,0$, $c=11x$, $d=1\,x\,1$, $e=0\,1\,0$, $f=1\,x\,x$

图 1-3　立方间的相互关系

一、包含关系及吸收运算

1. 包含关系　在图 1-3 中，立方 e 只占有了或只覆盖了顶点 **010**，但立方 a 不但覆盖了顶点 **000**，还覆盖了 **010**，显然立方 e 包含在立方 a 之中，或者说立方 a 包含了立方 e。同样，

立方 f 覆盖了四个顶点 **100**、**101**、**110**、**111**，立方 b 覆盖了两个顶点 **100**、**110**，显然立方 f 包含了立方 b。立方 a 虽也只覆盖了两个顶点，但是并未包含在立方 f 内。由此我们可以说：如果 t^j 所覆盖的全部顶点都为 t^i 所覆盖，则立方 t^j 包含于立方 t^i，记作 $t^j \Rightarrow t^i$。根据这个定义，立方 c 包含于立方 f，但它并不包含于立方 d。因为 c 所覆盖的顶点之一 **110** 未被 d 所覆盖。

如果 t^i 对应的 0- 复合体为 $K^0(t^i)$，t^j 对应的 0- 复合体为 $K^0(t^j)$，则满足包含关系的条件可改写为：若 $K^0(t^j) \subseteq K^0(t^i)$ ①，则 $t^j \Rightarrow t^i$。

立方间的包含关系是可传递的，即如果 $t^1 \Rightarrow t^2$、$t^2 \Rightarrow t^3$，则 $t^1 \Rightarrow t^3$。包含关系是不可逆的，即 $t^1 \Rightarrow t^2$，但 $t^2 \Rightarrow t^1$ 不一定成立。

2. 如何判别立方间的包含关系 对照立方 **010** 包含于 **0**x**0**，**1**x**0** 包含于 **1**xx，可以归纳出判别有 n 个组元的立方 $t^j = t_1^j t_2^j \cdots t_n^j$ 是否包含于 $t^i = t_1^i t_2^i \cdots t_n^i$ 的准则是：若对于所有的 $t_k^i \in \{\mathbf{0}, \mathbf{1}\}$（即 t^i 的每一个非 x 组元），均有 $t_k^i = t_k^j$，则 $t^j \Rightarrow t^i$。这个准则也可叙述为：如果存在一个 $t_k^i \in \{\mathbf{0}, \mathbf{1}\}$，满足 $t_k^i \neq t_k^j$，则 $t^j \not\Rightarrow t^i$。

例

$t^1 =$ **100**x**1**、$t^2 =$ **1**x**0**xx，对于 t^2 的两个非 x 组元，满足 $t_1^2 = t_1^1 =$ **1**、$t_3^2 = t_3^1 =$ **0**，$\therefore$ $t^1 \Rightarrow t^2$。

例

$t^1 = xxx$**1**、$t^2 =$ **0**x**11**，存在有 $t_1^2 =$ **0** $\neq t_1^1$，$\therefore t^1 \not\Rightarrow t^2$。但 t^1 的唯一的一个非 x 组元 $t_4^1 = t_4^2 =$ **1**，$\therefore$ $t^2 \Rightarrow t^1$。

现在考虑如何用计算机判别立方间的包含关系。设 t^i 和 t^j 如下所示：

```
          ↓ ↓ ↓   ↓
t^i = 0 0 0 1 1 1 x x x = 0 1  0 1  0 1  1 0  1 0  1 0  1 1  1 1  1 1
t^j = 0 1 x 0 1 x 0 1 x = 0 1  1 0  1 1  0 1  1 0  1 1  0 1  1 0  1 1
t^i · t^j               = 0 1  0 0  0 1  0 0  1 0  1 0  0 1  1 0  1 1
                            ↑    ↑    ↑         ↑
```

这两个立方有九对组元，代表 t_k^i 和 t_k^j 之间可能出现的九种情况。箭头指出了 $t_k^i \in \{\mathbf{0}, \mathbf{1}\}$、且 $t_k^i \neq t_k^j$ 的四对组元。如果采用第一种代码，则与这两个立方对应的内存单元中的状态如等式右侧所示。由上列的第三个式子可见，若 $t^j \Rightarrow t^i$，则必有

$$t_k^i \cdot t_k^j = t_k^j \qquad k = 1, 2, \cdots, n$$

即

$$t^i \cdot t^j = t^j$$

这个表达式的 FORTRAN 语句为

IAND(T(I), T(J)).EQ.T(J)②

如果这个表达式的值为真，则 $t^j \Rightarrow t^i$。

3. 立方集合间的包含关系 立方间的包含关系可以推广到立方集合与立方集合之间。

---

① 注意符号 $\subseteq$ 与 $\Rightarrow$ 之间的差别。$A \subseteq B$ 表示 A 是 B 的子集，即 A 中的每一个元素一定是 B 中的一个元素。如 $A =$ {**1**x**0**}，$B =$ {**1**xx}，显然，$A \subsetneq B$。$t^j \Rightarrow t^i$ 表示 t^j 所覆盖的顶点都被 t^i 所覆盖，因此，**1**x**0** $\Rightarrow$ **1**xx。

② IAND(X, Y)表示进行整型量 X、Y 的位对位与运算。若 X=**0011**，Y=**1010**，则运算的结果为 **0010**。

如果有两个立方集合 T^1 和 T^2，当且仅当 T^1 所覆盖的顶点均为 T^2 所覆盖时，即 $K^0(T^1)\subseteq K^0(T^2)$ 时，我们说 T^1 包含于 T^2，记作 $T^1\Rightarrow T^2$。

若有三个立方的集合：

$$T^1=\begin{Bmatrix}1&0&0\\0&1&x\end{Bmatrix}\qquad T^2=\begin{Bmatrix}1&x&0\\0&1&x\end{Bmatrix}\qquad T^3=\begin{Bmatrix}1&x&0\\0&1&1\end{Bmatrix}$$

显然

$$K^0(T^1)=\begin{Bmatrix}1&0&0\\0&1&0\\0&1&1\end{Bmatrix}\qquad K^0(T^2)=\begin{Bmatrix}1&0&0\\1&1&0\\0&1&0\\0&1&1\end{Bmatrix}\qquad K^0(T^3)=\begin{Bmatrix}1&0&0\\1&1&0\\0&1&1\end{Bmatrix}$$

从而有 $\qquad T^1\Rightarrow T^2\qquad T^3\Rightarrow T^2\qquad T^1\not\Rightarrow T^3$

4. 吸收运算 利用立方间的包含关系可以消去或吸收被包含的立方。设立方集合 T 覆盖的 0- 复合体为 $K^0(T)$。若 T 中的两个元素 t^i 和 t^j，满足 $t^j\Rightarrow t^i$，则必然存在 $K^0(t^j)\subseteq K^0(t^i)$，即被 t^j 所覆盖的全部顶点必均被 t^i 所覆盖，故 $T-t^j$ 必仍覆盖了 $K^0(T)$，因此我们可以从 T 中消去 t^j。

从阵列 T 中消去了所有这样的 t^j 以后，得到 T'。我们称由 T 求 T' 的运算叫做**吸收运算**，记作 $T'=\mathbf{A}(T)$。显然 $\mathbf{A}(T)\subseteq T$。

例

$T=\{a,\ b,\ c,\ d,\ e,\ f\}$，如图 1-3 所示。由于 $e\Rightarrow a$、$b\Rightarrow f$、$c\Rightarrow f$、$d\Rightarrow f$，故

$$T'=\mathbf{A}(T)=\{a,\ f\}$$

这种运算与逻辑代数中的吸收定理 $ab+a=a$ 相对应。在上面的例子中，与 T 对应的逻辑表达式为 $\bar{A}\bar{C}+A\bar{C}+AB+AC+\bar{A}B\bar{C}+A$，应用吸收定理后，可变换成 $\bar{A}\bar{C}+A$，即与 T' 相对应。

二、并运算

如果有两个阵列 $T^1=\{t^1,\ t^2,\ \cdots,\ t^s\}$、$T^2=\{t^{s+1},\ t^{s+2},\ \cdots,\ t^p\}$，它们的**并** $T=T^1\cup T^2=\{t^1,\ t^2,\ \cdots,\ t^s,\ t^{s+1},\ t^{s+2},\ \cdots,\ t^p\}$。因此，如果 T^1 和 T^2 覆盖的 0- 复合体分别为 $K^0(T^1)$ 和 $K^0(T^2)$，则阵列 T 既覆盖了 $K^0(T^1)$ 的全部 0- 立方，也覆盖了 $K^0(T^2)$ 的全部 0- 立方。设 $T^1=\{a,\ d\}$、$T^2=\{b,\ e\}$，立方 a、b、d、e 如图 1-3 所示，则 $T=T^1\cup T^2=\{a,\ b,\ d,\ e\}$。这种运算和逻辑代数中的**或**运算相对应。

在两个阵列合并之后，可能有一些元素包含于另一些元素，这些被包含的元素可以从中消去，以便减少阵列中的元素数。所以并运算常和吸收运算同时进行，并称为有吸收的并运算，记作 $\mathbf{A}(T^1\cup T^2)$。

在上面这个例子中，$T=\mathbf{A}(T^1\cup T^2)=\{a,\ b,\ d\}$。

三、立方相交及相交运算

1. 立方相交 如果有两个立方 t^i 和 t^j，在它们所覆盖的顶点中有一个或一个以上是公

共的，则我们称这两个立方是**相交**的。这些公共顶点称为这两个立方的交，记作 $t=t^i \cap t^j$。

从图 1-3 可以看出，立方 c 和 d 有公共顶点 **111**，是相交的，它们的交就是这个公共顶点 **111**。同理，立方 c 和 b 也是相交的，而 b 和 d 则是不相交的。如果两个立方不相交，我们可称它们的交为空集 $\varnothing$ 或**空顶点** ψ。

检查图 1-3 所示的各个立方表明，立方 t^i 和 t^j，如果存在一对或一对以上的组元，满足 $t_k^i=\mathbf{1}$、$t_k^j=\mathbf{0}$，或 $t_k^i=\mathbf{0}$、$t_k^j=\mathbf{1}$，则这两个立方就不相交。例如，$b=\mathbf{1}x\mathbf{0}$、$d=\mathbf{1}x\mathbf{1}$，它们共有三对组元，其中第三对组元符合上述条件，因而是不相交的。

2. 相交运算 立方的相交运算就是判别这两个立方是否相交，并求出它们的交，即公共顶点的集合。

根据两个立方相交的定义，相交运算可归纳为表 1-2 所示的坐标表，并按如下规则进行：如果存在一对或一对以上的组元满足 $t_k^i \cap t_k^j=\varepsilon$，则 $t^i \cap t^j=\varnothing$；否则 $t^i \cap t^j=t$，其中 $t_k=t_k^i \cap t_k^j$。

表 1-2 相交运算的坐标表

$\cap$		t_k^j		
		0	**1**	x
t_k^i	**0**	**0**	ε	**0**
	1	ε	**1**	**1**
	x	**0**	**1**	x

例

$a=\mathbf{0}x\mathbf{0}$、$c=\mathbf{11}x$、$d=\mathbf{1}x\mathbf{1}$、$f=\mathbf{1}xx$，根据上述规则可得 $c \cap d=\mathbf{111}$、$c \cap f=\mathbf{11}x$、$a \cap c=\varnothing$。

3. 相交运算的基本性质

① $a \cap b=b \cap a$

② $(a \cap b) \cap c=a \cap (b \cap c)$

③ $(a \cap b) \cup (a \cap c)=a \cap (b \cup c)$

④ $a \cup (b \cap c)=(a \cup b) \cap (a \cup c)$

由此可以导出立方与立方集合、立方集合与立方集合间的交运算：

$$t \cap T=\bigcup_{t^i \in T}(t \cap t^i)$$

$$T^1 \cap T^2=\bigcup_{t^i \in T^1}(t^i \cap T^2)$$

例

已知立方 $t=\mathbf{00}x$，立方集合 $T=\{t^1, t^2\}=\{\mathbf{0}x\mathbf{0}, x\mathbf{01}\}$，则

$$t \cap T=(t \cap t^1) \cup (t \cap t^2)=(\mathbf{00}x \cap \mathbf{0}x\mathbf{0}) \cup (\mathbf{00}x \cap x\mathbf{01})$$
$$=\mathbf{000} \cup \mathbf{001}=\{\mathbf{000}, \mathbf{001}\}$$

例

已知 $T^1=\{\mathbf{00}, \mathbf{11}\}$、$T^2=\{\mathbf{10}, \mathbf{11}\}$，则

$$T^1 \cap T^2=(\mathbf{00} \cap \{\mathbf{10}, \mathbf{11}\}) \cup (\mathbf{11} \cap \{\mathbf{10}, \mathbf{11}\})$$
$$=((\mathbf{00} \cap \mathbf{10}) \cup (\mathbf{00} \cap \mathbf{11})) \cup ((\mathbf{11} \cap \mathbf{10}) \cup (\mathbf{11} \cap \mathbf{11}))=\mathbf{11}$$

立方的相交运算与逻辑代数中的与运算是相对应的。在上面的例子中，和 T^1 对应的布尔表达式为 $\bar{A}\bar{B}+AB$，和 T^2 对应的布尔表达式为 $A\bar{B}+AB$。$(\bar{A}\bar{B}+AB)(A\bar{B}+AB)=AB$，正好与 $T^1\cap T^2$ 相对应。

4. 用计算机实现相交运算 现在我们来讨论如何用计算机求立方的交。从表 1-2 可以看出，如果某对组元出现下面两种情况：

$$\mathbf{0}\cap\mathbf{1}=\varepsilon \quad (\mathbf{01}\cdot\mathbf{10}=\mathbf{00})$$
$$\mathbf{1}\cap\mathbf{0}=\varepsilon \quad (\mathbf{10}\cdot\mathbf{01}=\mathbf{00})$$

则这两个立方就不相交。若我们仍用第一种代码，如上式右边括号所示，则有 $t_k^i\cdot t_k^j=\mathbf{00}$。所以我们仅需判别 $t^i\cdot t^j$ 的代码中是否出现 **00**，就可以决定 t^i、t^j 是否相交。下列各式说明了判别的方法：

	↓	↓								
$t^i=$	**0**	**0**	**0**	**1**	**1**	**1**	x	x	$x=$	**01**	**01**	**01**	**10**	**10**	**10**	**11**	**11**	**11**
$t^j=$	**0**	**1**	x	**0**	**1**	x	**0**	**1**	$x=$	**01**	**10**	**11**	**01**	**10**	**11**	**01**	**10**	**11**
$R=t^i\cdot t^j$									$=$	**01**	**00**	**01**	**00**	**10**	**10**	**01**	**10**	**11**
$2\times R$（左移一位）									$=$	**10**	**00**	**10**	**01**	**01**	**00**	**11**	**01**	**10**
$s=R+(2\times R)$									$=$	**11**	**00**	**11**	**01**	**11**	**10**	**11**	**11**	**11**
$MASK$									$=$	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
$s\cdot MASK$									$=$	**10**	**00**	**10**	**00**	**10**	**10**	**10**	**10**	**10**
											↑		↑					

其中第三式再次表明了如 t^i、t^j 不相交就会出现 **00** 代码的情况。为判别是否存在这样的代码设置了一个供判别用的二进制数 $MASK$，并进行所示的一系列运算，最后的结果表明在 **00** 代码对应的位置上 $s\cdot MASK$ 的组元的代码也是 **00**，而其他各组元的代码均与 $MASK$ 的代码相同，由此我们可以得出结论：若 $s\cdot MASK\neq MASK$，则 $t^i\cap t^j=\varnothing$。描述上述过程的 FORTRAN 语句是

```
R=IAND(T(I), T(J))
IF(IAND(MASK, IOR(R, ISLL(R, 1))). NE. MASK) ①
```

当该条件不能成立时，R 就是这两个立方的交。

顺便指出，如果 $t^i\cap t^j=t^j$，则 $t^j\Rightarrow t^i$。因此，包含关系只是相交关系中的一种特殊情况。

四、相邻、相接和相容运算

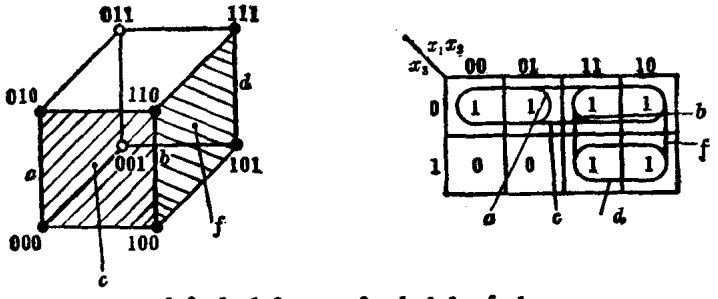
1. 相邻和相接 任意两个 r- 立方，$t^i=t_1^it_2^i\cdots t_n^i$、$t^j=t_1^jt_2^j\cdots t_n^j$，当且仅当：

① 存在且仅存在一对组元 t_k^i、t_k^j，满足 $t_k^i=\mathbf{1}$、$t_k^j=\mathbf{0}$，或 $t_k^i=\mathbf{0}$、$t_k^j=\mathbf{1}$；

② $t_l^i=t_l^j$，其中 $l=1, 2, \cdots, n$，但 $l\neq k$，则称 t^i 和 t^j **相邻**。在图 1-4 所示的各立方中，立方 $a=\mathbf{0}x\mathbf{0}$、$b=\mathbf{1}x\mathbf{0}$，有 $a_1=\mathbf{0}$、$b_1=\mathbf{1}$、$a_2=b_2=x$、$a_3=b_3=\mathbf{0}$，故二者相邻。立方 b、d 之间也

① ISLL(X, 1)表示把整型量 X 左移 1 位。若机器的字长为 4，X=**1011**，则 I=ISLL(X, 1)=**0110**。
IOR(X, Y)表示对整型量 X、Y 进行位对位的或运算。若 X=**0011**、Y=**1010**，则 I=IOR(X, Y)=**1011**。

因有类似关系而相邻。相邻的立方在空间或在卡诺图中都处于相邻的位置。若两个立方相邻，则它们的维数必然相等。两个相邻的 r- 立方共有 2^r 对相邻的顶点，这 2^r 对顶点可以构成一个新的 $(r+1)$-立方。例如上述的两个 1- 立方 $\mathbf{0}x\mathbf{0}$、$\mathbf{1}x\mathbf{0}$ 共有 2^1 对顶点(**010**, **110**)、(**000**, **100**)相邻，且可合并成一个新的立方 $c=x\,x\,\mathbf{0}$，它是一个 2- 立方。



$a=0x0,\ b=1x0,\ c=xx0,\ d=1x1,\ f=1xx$

图 1-4　相邻和相接关系

如果两个立方 t^i、t^j，只满足上述第一个条件，即存在且仅存在一对组元 t_k^i、t_k^j，满足 $t_k^i=\mathbf{1}$、$t_k^j=\mathbf{0}$，或 $t_k^i=\mathbf{0}$、$t_k^j=\mathbf{1}$，则称 t^i 与 t^j **相接**。在图 1-4 中立方 $a=\mathbf{0}x\mathbf{0}$、$f=\mathbf{1}x\,x$，仅有第一对组元 $a_1=\mathbf{0}$、$f_1=\mathbf{1}$ 满足上述第一个条件，故两个立方是相接的。

若两个立方 t^i 和 t^j 相接，它们的维数未必相同，但必存在有 r 对均为 x 的组元 ($r=0,\ 1,\ 2,\ \cdots$)。这样，t^i 中将有 2^r 个顶点与 t^j 中对应的 2^r 个顶点一一相邻，它们构成了 2^r 对相邻的顶点对。这些相邻的顶点对可以构成一个新的立方，这个新的立方不包含于任何一个原来的立方。在上面的例子中 $d=\mathbf{1}x\mathbf{1}$，是一个 1- 立方，$c=x\,x\,\mathbf{0}$，是一个 2- 立方，它们的第二对组元均为 x，故 $r=1$，从而有 2^1 对顶点相邻，即 **101** 与 **100** 相邻，**111** 与 **110** 相邻，这两对顶点构成一个新的立方 $f=\mathbf{1}x\,x$，是一个 2- 立方。

可以看到，立方相邻是立方相接的一种特殊情况。

2. 相容运算　两个立方 t^i 和 t^j，它们的**相容立方**是指不包含在 t^i 内也不包含在 t^j 内而包含于 $t^i \cup t^j$ 内的最大的立方。判别是否存在并求出这样的立方的运算叫**相容运算**，记作 $t^i * t^j$。

显然，当且仅当两个立方 t^i、t^j 相接时，才存在相应的相容立方，这时 $t^i * t^j = t$，否则就不存在相容立方，即 $t^i * t^j = \varnothing$。

相容运算与相交运算有相同的坐标表，如表 1-3 所示。只是使用规则修改为：当存在且仅存在有一对组元，满足 $t_k^i * t_k^j = \varepsilon$，则 $t = t^i * t^j = (t_1^i * t_1^j)\,(t_2^i * t_2^j) \cdots (t_{k-1}^i * t_{k-1}^j)\,x\,(t_{k+1}^i * t_{k+1}^j) \cdots (t_n^i * t_n^j)$，否则，$t^i * t^j = \varnothing$。

表 1-3　相容运算的坐标表

$*$		t_k^j		
		0	1	x
t_k^i	0	0	ε	0
	1	ε	1	1
	x	0	1	x