

第10章 逻辑与开关

真理是什么呢？亚里士多德认为逻辑与它有关。他的讲义合集《工具论》（Organon，可追溯到公元前4世纪）是最早的关于逻辑的详细著作。对于古希腊人而言，逻辑是追寻真理的过程中用于分析语言的一种手段，因此它被认为是一种哲学。亚里士多德的逻辑学的基础是三段论。最有名的三段论（它并非是在亚里士多德的著作中发现的）是：

*All men are mortal;
Socrates is a man;
Hence, Socrates is mortal.*

（所有的人都是要死的；
苏格拉底是人；
所以，苏格拉底是要死的。）

在三段论中，两个前提被假设是正确的，并由此推出结论。

苏格拉底之死这个例子看上去似乎太直白了，但还有许多其他不同的三段论。例如，考虑下面两个由19世纪数学家Charles Dodgson（也就是Lewis Carroll）提出的前提：

*All philosophers are logical;
An illogical man is always obstinate.*

（所有的哲学家都是有逻辑头脑的；
一个没有逻辑头脑的人总是顽固的。）

它所能推出的结论一点儿也不明显。（事实上，结论是“一些顽固的人不是哲学家（Some obstinate persons are not philosophers）”）请注意结论中一个出乎意料且令人迷惑的词“一些（some）”。

两千多年来，数学家们对亚里士多德的逻辑理论苦苦思索，试图用数学符号和操作符来表现它。19世纪以前，唯一能接近这个目标的人是莱布尼兹（1648—1716），他早年涉足逻辑学领域，后来转向其他学科（比如说，他几乎和牛顿同时独立地发明了微积分）。

接下来有所突破的是乔治·布尔。

乔治·布尔1815年生于英格兰，他周围的环境对他的成长很不利。他父亲是鞋匠，而母亲曾是女仆，英国森严的等级制度使布尔学不到什么有别于父辈的东西。但是，靠着他自己强烈的好奇心及父亲的帮助（其父对科学研究、数学和文学有浓厚的兴趣），年轻的乔治自学了上层阶级男孩才能学到的课程，包括拉丁文、希腊语及数学。由于他早年在数学方面发表的论文，1849年，布尔被任命为爱尔兰Cork市的皇后大学数学系的首席教授。

19世纪中期的几位数学家在逻辑理论的数学定义上做了一些工作（最著名的是迪摩根），但只有布尔有真正概



念上的突破。他最早的贡献是发表的一本很简短的书《The Mathematical Analysis of Logic, Being an Essay Towards a Calculus of Deductive Reasoning》(1847), 接着又发表了一篇很长且充满抱负的文章:《An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities》(1854), 简称为《The Laws of Thought》。1864年的一天, 布尔在雨中赶去上课时不幸感染上了肺炎, 不治身亡, 享年 49岁。

我们可以从布尔在 1854年所著书的题目中看出他富于野心的想法: 由于充满理性的人脑用逻辑去思考, 那么, 如果能用数学来表征逻辑, 我们也就可以用数学来描述大脑是如何工作的。当然, 现在看来这种想法似乎十分幼稚。(但却超越了他所在的年代。)

布尔发明了一种和传统代数看起来、用起来都十分相似的代数。在传统代数中, 操作数(通常是字母)代表数字, 而操作符(多是“+”或“×”)指明这些操作数如何结合到一起。一般我们可用传统代数解决类似下面的问题: 如果安娜有 3磅豆腐, 贝蒂的豆腐是安娜的 2倍, 卡门的豆腐比贝蒂多 5磅, 迪尔德丽的豆腐是卡门的 3倍。那么, 迪尔德丽有多少豆腐呢?

为了计算这个问题, 我们首先把语句转化为算术式子, 用四个字母代表每个人拥有豆腐的数量, 即:

$$A = 3$$

$$B = 2 \times A$$

$$C = B + 5$$

$$D = 3 \times C$$

可以通过代入把上述四个表达式合为一个式子, 最后执行加法和乘法, 即:

$$D = 3 \times C$$

$$D = 3 \times (B + 5)$$

$$D = 3 \times ((2 \times A) + 5)$$

$$D = 3 \times ((2 \times 3) + 5)$$

$$D = 33$$

当做传统代数题时, 要遵循一定的规则。这些规则可能已经和实践融为一体, 以至于我们不再认为它们是规则, 甚至忘记了它们的名字。但规则确实是任何形式的数学的基础。

第一个规则是加法与乘法的交换律, 即我们可以在操作符两边交换操作数的位置:

$$A + B = B + A$$

$$A \times B = B \times A$$

相反, 减法和除法是不满足交换律的。

加法和乘法也满足结合律, 即:

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

$$A \times (B \times C) = (A \times B) \times C$$

最后, 乘法对加法可以进行分配:

$$A \times (B + C) = (A \times B) + (A \times C)$$

传统代数的另外一个特点是它总是处理数字，如豆腐的重量或鸭子的数量，火车行驶的距离或家庭成员的年龄。是布尔超凡的智慧使代数脱离了数字的概念而变得更加抽象。在布尔代数中（布尔的代数最终被这样命名）操作数不是指数字，而是指集（类）。一个类仅仅表示一组事物，也就是后来熟知的集合。

让我们来讨论一下猫。猫或公或母，为方便起见，我们用字母 M 指代公猫的集合，用 F 指代母猫的集合。记住，这两个符号并不代表猫的数量，公猫或母猫的数量随着小猫仔的出生和老猫的不幸离去而变化，这两个字母代表的是猫的种类——具有某种特点的猫。因而我们不说公猫，而是用 M 来代表它们。

我们也可以使用其他字母代表猫的颜色。例如，用 T 代表黄褐色的猫，用 B 代表黑猫，用 W 代表白猫，而用 O 代表所有其他颜色的猫。

最后（至少就这个例子而言），猫要么是阉过的要么是有生育能力的。我们用字母 N 代表阉过的猫，而用 U 代表有生育能力的猫。

在传统代数中，操作符 $+$ 和 \times 被用于表示加法和乘法。在布尔代数中，同样用到了 $+$ 和 \times 。这似乎会引起混淆。人人都知道在传统代数中如何对数字进行加和乘，但是我们如何对“类”进行加和乘呢？

事实上，在布尔代数中我们并不真正地做加或乘，相反，这两个符号有着完全不同的意思。

在布尔代数中，符号 $+$ 意味着两个集合合并，两个集合的合并就是包含第一个集合的所有成员及第二个集合的所有成员。例如， $B+W$ 表示黑猫和白猫的集合。

布尔代数中的符号 \times 意味着取两个集合的交集，两个集合的交集包含的元素既在第一个集合中，也在第二个集合中。例如， $F \times T$ 代表了一种猫的集合，这个集合中的猫既是母猫又是黄褐色的。与传统代数一样，我们可以把 $F \times T$ 写成 $F \cdot T$ 或简写为 FT （这正是布尔代数所期望的）。你可以把这两个字母看成是连在一起的两个形容词：黄褐色的母猫。

为避免传统代数和布尔代数之间的混淆，有时候用符号 \cup 和 \cap 而不用 $+$ 和 \times 来表示并运算和交运算。但布尔对数学的解放性的部分影响是使熟悉的操作符更加抽象，所以，我们决定坚持他的决定，而不为他的代数引入新的符号。

交换律、结合律和分配律在布尔代数中均适用。而且，在布尔代数中，操作符 $+$ 可以对 \times 进行分配，这在传统代数中是不成立的，即：

$$W + (B \times F) = (W + B) \times (W + F)$$

这个式子表示白猫（ W ）和黑色母猫（ $B \times F$ ）的并集和等式右边两个集合的交集是一样的，这两个集合是白猫和黑猫的并集（ $W+B$ ）及白猫和母猫的并集（ $W+F$ ）。要掌握这个规则有些困难，但它的确有用。

为了使布尔代数更加完整，我们还需要两个符号。这两个符号看上去像数字，但它们并不真的是数字，因为有时候它们和数字有些不同。符号“ 1 ”在布尔代数中表示“整个宇宙（全集）”，也就是我们所谈论的每件事物。本例中，符号“ 1 ”表示“所有的猫”。这样：

$$M + F = 1$$

即母猫和公猫的并集是所有的猫。同样，黄褐色猫、黑猫、白猫及其他颜色的猫的并集也是所有的猫，即：

$$T+B+W+O=1$$

你也可以这样表示所有的猫：

$$N+U=1$$

符号1可以用一个减号 - 来排除一些事物。例如：

$$1 - M$$

表示除了公猫以外的所有猫。排除公猫以后的全集就是母猫的集合：

$$1 - M = F$$

我们所需要的另外一个符号是“0”。在布尔代数中，“0”表示空集，即不含任何事物的集合。当求取两个完全相互排斥的集合的交集时，空集就产生了。例如，既是母的又是公的猫的集合可以表示为：

$$F \times M = 0$$

注意，符号1和0有时的用法与传统代数相同。例如，所有的猫和母猫求交集即是母猫这个集合：

$$1 \times F = F$$

空集和母猫求交集还是空集：

$$0 \times F = 0$$

空集和母猫的并是母猫这个集合：

$$0+F = F$$

但有时与传统代数中得到的结果就不太一样了。例如，所有的猫和母猫的并集是所有的猫：

$$1+F = 1$$

这个表达式在传统代数中是没有意义的。

由于F代表母猫的集合， $1 - F$ 代表所有其他猫的集合，则这两个集合的并集是1：

$$F + (1 - F) = 1$$

并且它们的交集是0：

$$F \times (1 - F) = 0$$

历史上，这个公式代表了逻辑中一个十分重要的概念，即矛盾律。它表明一个事物不能同时是它自己和它自己的反面。

使布尔代数和传统代数看起来完全不同的是下面这个表达式：

$$F \times F = F$$

这个式子在布尔代数中有着完美的意义：母猫的集合和母猫的集合的交集仍旧是母猫的集合。但若F代表一个数字的话，这个公式显然就不对了。布尔认为：

$$X^2 = X$$

是使他的代数与传统代数区分开来的唯一表达式。另一个按照传统代数看起来很有趣的布尔表达式是：

$$F + F = F$$

母猫和母猫的并集仍是母猫这个集合。

布尔代数为解决亚里士多德的三段论提供了一个数学方法。再看看这个著名三段论的两个前提：

所有的人都是要死的；

苏格拉底是人。

我们用字母P代表所有人的集合，M代表要死的东西的集合，S代表苏格拉底。那么，所谓“所有的人都是要死的”意味着什么呢？它其实表示了所有人的集合和所有要死的东西的集合的交集是所有人这个集合，即：

$$P \times M = P$$

而 $P \times M = M$ 这个式子是错误的，因为要死的東西还包括猫、狗、榆树等等。

而“苏格拉底是人”意味着苏格拉底这个集合（非常小）和所有人的集合（很大）的交集是苏格拉底这个集合：

$$S \times P = S$$

由于从第一个式子中知道 $P = P \times M$ ，所以可以把它代入第二个式子，即：

$$S \times (P \times M) = S$$

根据结合律，上式等同于：

$$(S \times P) \times M = S$$

但我们已经知道 $S \times P$ 等于S，所以上式可简化为：

$$S \times M = S$$

现在计算完毕。这个表达式告诉我们，苏格拉底和所有要死东西的集合的交集是苏格拉底，也就是说苏格拉底是要死的。相反，如果认为 $S \times M$ 等于0，那么结论就是苏格拉底不会死。再如果，若 $S \times M$ 等于M，则能推出的结论就是苏格拉底是唯一会死去的東西，而其他任何东西都是不朽的！

用布尔代数来证明显而易见的事实似乎有些小题大做（尤其当考虑到苏格拉底早已在2400年以前就去世了时），不过，布尔代数还可以用来判断一些事物是否满足一定的标准。也许有一天，你走进宠物店对店员说：“我想要一只阉过的公猫，白的或黄褐色的均可；或者要一只没有生殖能力的母猫，除了白色，其他任何颜色均可；或者只要是只黑猫，我也要。”店员对你说：“看来您想要的猫是下面的式子表示的集合中的一只：

$$(M \times N \times (W+T)) + (F \times N \times (1-W)) + B$$

对吗？”你回答道：“是的，完全正确！”

为了证明店员是正确的，你可能想放弃并和交的概念而转向“OR（或者/或）”和“AND（并且/与）”。大写这两个词是因为虽然在通常情况下它们代表语言中的概念，但它们也代表了布尔代数中的操作。当求两个集合的并集时，你实际上是从第一个集合“或”从第二个集合中取得事物放入结果集合里。当求两个集合的交集时，满足条件的事物必定在第一个集合中“并且”也在第二个集合中。此外，每当你看见后跟减号的1，你可以使用单词“NOT（非）”

来表示。小结如下：

- + (以前表示求并集) 现在表示 OR。
- × (以前表示求交集) 现在表示 AND。
- 1- (以前表示从全集中排除一些事物) 现在表示 NOT。

这样，刚才的表达式可以写成下面的形式：

$$(M \text{ AND } N \text{ AND } (W \text{ OR } T)) \text{ OR } (F \text{ AND } N \text{ AND } (\text{NOT } W)) \text{ OR } B$$

这与你的口头描述已经十分接近了。注意圆括号是如何清楚地表达出你的意图的。你想要的猫来自下面三个集合之一：

$$(M \text{ AND } N \text{ AND } (W \text{ OR } T))$$

或

$$(F \text{ AND } N \text{ AND } (\text{NOT } W))$$

或

B

写下这个公式后，店员就可以进行布尔测试的工作了。别这么大惊小怪的，这里已经悄悄转移到另一种不同形式的布尔代数中去了。在这种形式的布尔代数中，字母不再只表示集合，字母还可以被赋予数字，但需要注意的是它们只能被赋予 0 或者 1。数字 1 表示“是的”、“正确”，本例中的意思是“这只猫符合我的要求”；数字 0 表示“否定”、“错误”、本例中即“这只猫不符合我的要求”。

首先，店员拿出一只未阉过的黄褐色的公猫。下面是满足条件的猫的集合：

$$(M \times N \times (W+T)) + (F \times N \times (1-W)) + B$$

当用 0 和 1 代替字母后就变成了下面的样子：

$$(1 \times 0 \times (0+1)) + (0 \times 0 \times (1-0)) + 0$$

注意被赋予了 1 的字母只有 M 和 T，因为拿来的这只猫是公的，黄褐色的。

现在必须要做的是简化这个表达式。如果简化后表达式的结果是 1，这只猫就满足了你的要求，否则就不是你想要的猫。当简化表达式时，千万记住我们并不是在真正地做加法和乘法。当 + 表示 OR，× 表示 AND 时，大部分规则是相同的。(现代课本中有时用 和 分别表示 AND 和 OR，而不用 × 和 +；但这里用 + 和 × 这两个符号却是恰到好处的。)

当用 × 表示 AND 时，可能的结果是：

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

换句话说，只有当 × 的左、右两个操作数均为 1 时，结果才为 1。这个过程和普通乘法一模一样。若用一张小表总结一下，你会发现它们和第 8 章的加法表和乘法表的形式相似：

AND	0	1
0	0	0
1	0	1

当用 + 表示 OR 时，可能的结果是：

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

$$1+1=1$$

当+的左、右操作数中有一个为1时，结果就是1。除了1+1=1这种情况，这种计算和普通加法产生的结果是一致的。可用另一张小表来总结：

OR	0	1
0	0	1
1	1	1

现在可以用这些表来计算前面那个表达式的结果了：

$$(1 \times 0 \times 1) + (0 \times 0 \times 1) + 0 = 0 + 0 + 0 = 0$$

结果是0，表示“否定”、“错误”，即这只小猫不满足客户需求。

接下来，店员拿来一只无生育能力的白色的小母猫。原始表达式是：

$$(M \times N \times (W+T)) + (F \times N \times (1-W)) + B$$

把0和1代入上式：

$$(0 \times 1 \times (1+0)) + (1 \times 1 \times (1-1)) + 0$$

并且把它简化一下：

$$(0 \times 1 \times 1) + (1 \times 1 \times 0) + 0 = 0 + 0 + 0 = 0$$

看来，这只可怜的小猫还是不符合要求。

然后，店员又拿来一只无生育能力的灰色的小母猫。（灰色是非白色、黑色或黄褐色的一种其他颜色。）下面是表达式：

$$(0 \times 1 \times (0+0)) + (1 \times 1 \times (1-0)) + 0$$

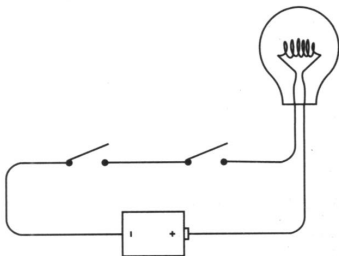
现在把它简化为：

$$(0 \times 1 \times 0) + (1 \times 1 \times 1) + 0 = 0 + 1 + 0 = 1$$

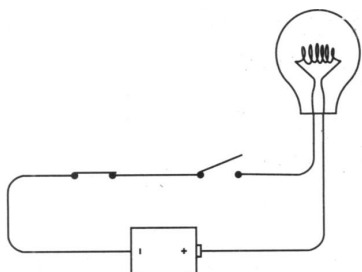
最后的结果1表示“是的”、“正确”，这只小猫总算找到新家了！

在你买到小猫的那天晚上，当小猫蜷身睡在你的腿上时，你开始考虑是否能够通过电线连接一些开关和灯泡来决定哪些小猫满足你的要求。（你真是一个奇怪的家伙。）你丝毫没有意识到你将要实现一个关键概念上的突破。你要做的是一些试验，这些试验把布尔代数和电路结合在一起，从而使使用二进制数字工作的计算机的设计和制造成为可能。（可别让这些话吓着你。）

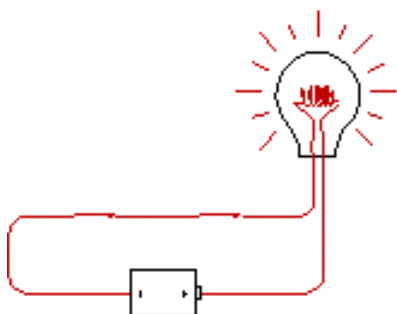
下面就开始了。你像往常一样把灯泡和电池连接在一起，这一回你用了两个开关：



开关这种方式的连接——一个在另一个的右边——称为串联的。如果你闭合了左边的开关，什么也不会发生：



同样，如果你让左边的开关断开而闭合右边的开关，结果还是一样。只有当左右两个开关都闭合时，灯泡才会发光，如下所示：



这里的关键是“都”。只有左边和右边的开关都闭合时，电流才能流过回路。

这个电路执行了一个逻辑运算。事实上，灯泡回答了这个问题：“两个开关都处于闭合状态吗？”可以把电路的工作总结为下面这张表：

左开关状态	右开关状态	灯泡状态
断开	断开	不亮
断开	闭合	不亮
闭合	断开	不亮
闭合	闭合	亮

在前一章中，我们已知道二进制数字（或“位”）是如何表示信息的：它可以表示从最普通的数字到Roger Ebert的拇指方向等的一切事情。可以说“0”代表“Ebert拇指向下的方向”，而“1”表示“Ebert拇指向上的方向”。一个开关有两个位置，所以它可以代表一个位。“0”表示“开关是断开的”，而“1”表示“开关是闭合的”。一个灯泡有两种状态，所以它也可以表示一个二进制位。“0”表示“灯泡不亮”而“1”表示“灯泡亮”。现在可以把上面的表简化一下：

左开关状态	右开关状态	灯泡状态
0	0	0
0	1	0
1	0	0
1	1	1

注意，如果交换左、右开关，结果是一样的，所以没必要指明哪个开关是左开关或右开关。因此这张表可以重画成类似于前面“AND”表和“OR”表的样子：

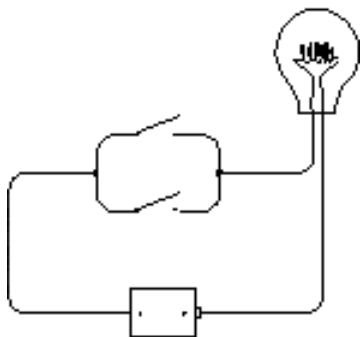
开关串联	0	1
0	0	0
1	0	1

事实上，这和“AND”表是一样的。让我们检查一下：

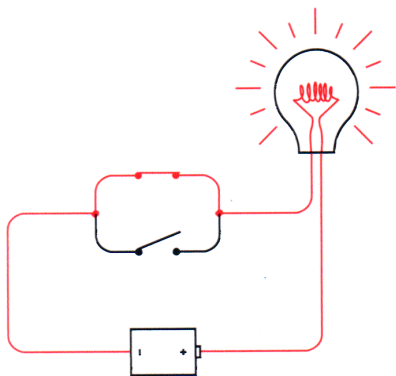
AND	0	1
0	0	0
1	0	1

这个简单的电路实际上执行了布尔代数的“AND”操作。

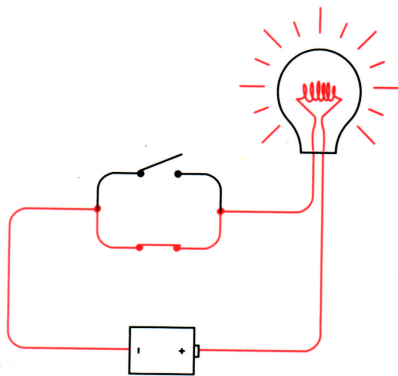
现在试着用另一种方式连接电路：



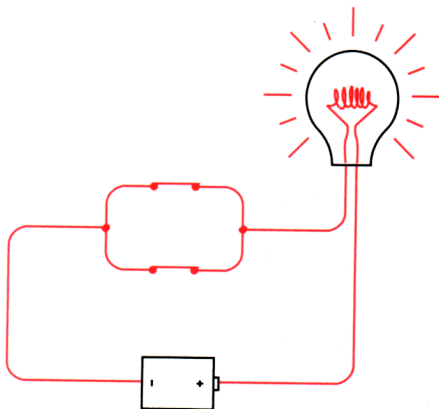
这些开关称为并行连接。它和前一种连接方式的区别是，如果闭合了上面的开关，灯泡会亮：



如果闭合了下面的开关，灯泡会亮：



如果同时闭合上、下两个开关，灯泡还是会亮：



可见，当上面或下面的开关有一个闭合时，灯泡就会亮。这里的关键字是“或”。

这个电路也执行了一个逻辑运算，灯泡回答了这样一个问题：“是否有开关闭合？”下面的表总结了电路是如何工作的：

上开关状态	下开关状态	灯泡状态
打开	打开	不亮
打开	闭合	亮
闭合	打开	亮
闭合	闭合	亮

仍然用“0”表示开关断开或灯泡不亮，用“1”表示开关闭合或灯泡亮。这张表可以这样：

上开关状态	下开关状态	灯泡状态
0	0	0
0	1	1
1	0	1
1	1	1

同样，这两个开关交换位置也没关系，所以这张表可以重写成如下的样子：

开关并联	0	1
0	0	1
1	1	1

你可能已经猜到了这和布尔代数中的“OR”表是一样的：

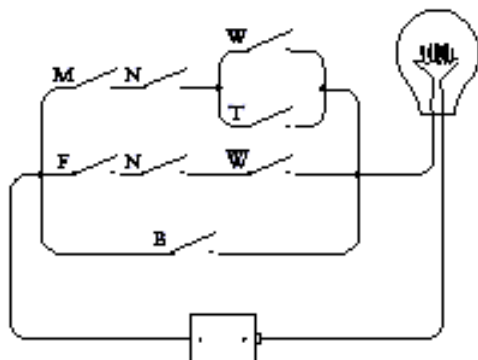
OR	0	1
0	0	1
1	1	1

这意味着两个并联的开关执行的是和布尔一样的操作。

当你再进入宠物店时，你告诉店员：“我想要一只阉过的公猫，白的或黄褐色的均可；或者要一只没生育能力的母猫，除了白色，其他任何颜色均可；或者只要是只黑猫，我也要。”店员便得到了如下的表达式：

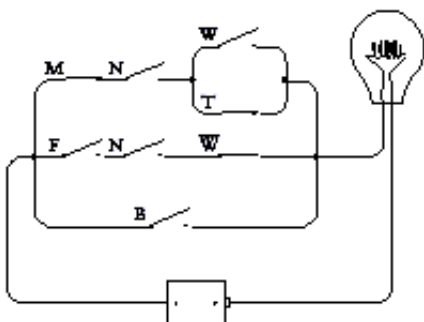
$$(M \times N \times (W+T)) + (F \times N \times (1 - W)) + B$$

现在你知道两个串联开关执行的是逻辑与 (AND, 由符号 \times 来表示), 两个并联开关执行的是逻辑或 (OR, 由符号 $+$ 来表示), 你可以按如下方法连接 8 个开关:

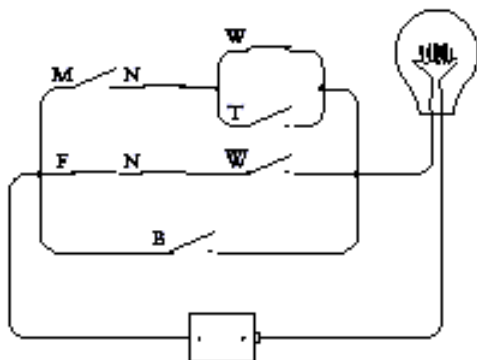


这个电路中的每一个开关都被标上了一个字母 (与布尔表达式中所用字母相同)。 \bar{W} 表示非 W, 是 $1 - W$ 的另一种写法。事实上, 如果按从左至右, 从上至下的顺序来阅读这个电路图, 你遇到的字母的顺序和它们在布尔表达式中出现的次序是一样的。表达式中的乘号 (\times) 都对应角是电路图中串联的两个或两组开关的位置; 表达式中的加号 ($+$) 号对应的是电路图中并联的两个或两组开关的位置。

你应该记得, 店员最先挑出的是只未阉过的褐色的公猫。闭合相应的开关:

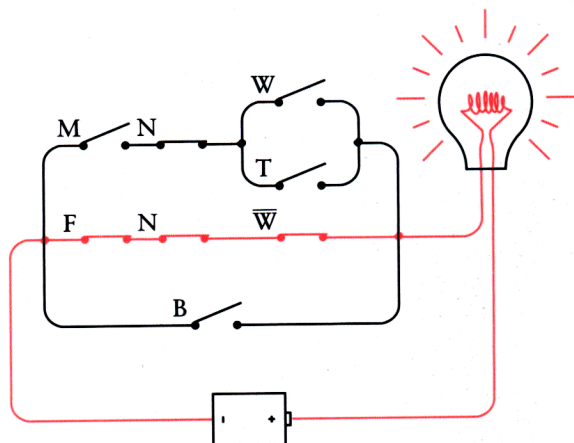


尽管 M、T 和非 W 这三个开关都闭合了, 但没有构造出一个完整的电路来点亮灯泡。接着, 店员拿出一只无生育能力的白色的母猫:



这次, 由于右边开关未闭合也无法构成一个完整的电路。但最后, 店员拿出一只无生育

能力的灰色的母猫：



这样就可以构成一个完整的电路，灯泡被点亮并表示小猫符合你的要求。

乔治·布尔从来没有连接过这样一个电路，他也没能看到用开关、电线和灯泡来实现一个布尔表达式。当然，其中的一个原因是直到布尔死后 15 年，白炽灯泡才被发明。但摩尔斯在 1844 年展示了他的电报机，比布尔的《The Laws of Thought》的发表早 10 年，而用一个电报发声器来代替所示电路中的灯泡是十分简单的。

可惜 19 世纪没有人把布尔代数中的与、或和串联、并联一些简单的开关联系起来。数学家没有、电工没有、电报操作员也没有，没有人想到过这种联系，甚至连计算机革命的创始人查尔斯·巴贝芝（1792—1871）也没有。他曾和布尔联系过，并了解过他的工作，他一生中大部分时间致力于设计第一台差分机及接下来的解析机。一个世纪之后，这些机器被认为是现代计算机的雏型。我们现在知道，帮助巴贝芝的是他认识到计算机应产生于电报继电器中，而非那些齿轮和控制杆。

是的，问题的答案正是电报继电器。