第3章

# 枚　举

专业的测试者都有一个根深蒂固的观念：枚举或穷尽测试是不可能的。原因在第1章讲解测试选择问题时已经分析过。这种观念致使测试人员在面对实际测试问题时，总是不假思索地放弃枚举的尝试，直接开始进行测试选择，却不可避免地在测试完整性方面遇到更大的挑战。实际上，比较合理的做法是采取“先扩张再压缩”的策略，也就是在测试设计时先考虑测试完整性问题，努力拓展测试输入空间的疆域，保证设计层面的测试尽可能充分；继而考虑正确性判定问题，看是否有可能通过测试得到被测对象正确的结论；最后才考虑如何解决测试选择问题，保证实施层面的测试可行性。

枚举是人类进行测试活动时最朴素和自然的思想。当测试人员想要得到“被测对象正确”的结论时，首先想到的办法就是枚举所有可能的使用场景。即便在今天的规模化研发生产活动中，测试人员很少有机会对完整的被测对象进行枚举测试，但是确实有很多测试设计方法中体现了枚举的思想，有很多测试者在用枚举的方式完成测试活动的某些环节或某些部分，并由此得到“被测对象在某些方面正确”的结论。

当然，测试人员也必须正视枚举带来的测试规模失控风险。实用的枚举测试方法，必然伴随着测试选择方面的考量。

## 3.1 组合测试

如 2.2.1 节所述，在被测对象所处的环境中，很多因素会影响期望的实现。这些影响因素之间可能存在相互作用的关系。被测对象的某些缺陷，只会在特定的影响因素组合条件下才会被激活。组合测试关注的正是这些影响因素之间的相互作用关系。

举例说明。假设被测对象是某电信系统交换机，被测对象期望是“具备正常的通话功能”。有四个因素可能影响该期望的实现，分别为呼叫种类、资费方式、接入方式和状态。每个因素均有三种不同的可能取值，如表 3-1 所示。

**表 3-1 影响某电信系统交换机通话功能的因素及其取值**

| 呼叫种类 | 资费方式 | 接入方式 | 状　态 |
|---|---|---|---|
| 市话 | 被叫方 | 环路 | 成功 |
| 长途 | 集中 | 综合业务 | 忙 |
| 国际 | 免费 | 专用分组 | 阻塞 |

通过对被测对象工作机制的分析，测试人员了解到被测对象对这 4 个因素的处理存在耦合——在通话功能相关的一系列逻辑决策中，大多包含两个及以上因素作为条件。为了验证通话功能是否正确，需要测试各种因素不同取值组合的情况。任一个形如(“市话”“集中”“综合业务”“阻塞”)的四元组可以构成一个测试用例。每个测试用例都可以覆盖一些特定的因素取值组合。组合测试的目的正是通过对这类用例的设计，验证被测对象在因素的耦合处理方面是“正确”的。

### 3.1.1 组合测试的概念

设与被测对象期望有关的影响因素共有 $n$ 个，记为 $C=\{c_1, c_2, \cdots, c_n\}$，这些影响因素可被视为被测对象的测试输入变量。在实践中，一般假定每个测试输入变量的水平取值范围是一个有限的离散点集 $V_i$，$V_i$ 中有 $a_i$ 个元素，即 $a_i=|V_i|(1\leqslant i\leqslant n)$。不失一般性，

设 $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n$。用 $R$ 表示各个测试输入变量之间可能存在的相互作用关系。例如，如果 $\{c_1, c_2\} \in R$，说明测试输入变量 $c_1$ 和 $c_2$ 之间存在相互作用，$c_1$ 与 $c_2$ 的某些取值组合可能激发被测对象的缺陷。

**定义：两两组合覆盖表**

设 $\boldsymbol{A}$ 是一个 $m \times n$ 的矩阵，即 $\boldsymbol{A} = (a_{ij})_{m \times n}$，其第 $j$ 列表示被测对象的测试输入变量 $c_j$，元素取自 $c_j$ 的水平取值范围 $V_j$。如果 $\boldsymbol{A}$ 的任意第 $i$ 列和第 $j$ 列都满足：$V_i$ 中所有元素和 $V_j$ 中所有元素的全部两两组合，都在 $\boldsymbol{A}$ 的第 $i$ 列和第 $j$ 列形成的二元有序对中至少出现一次，那么称 $\boldsymbol{A}$ 为被测对象期望的一个两两组合覆盖表。$\boldsymbol{A}$ 的每一行代表一个两两组合测试用例。

仍然以电信系统交换机通话功能的测试为例。如果令 $R = \{\{c_1, c_2\}, \{c_1, c_3\}, \{c_1, c_4\}, \{c_2, c_3\}, \{c_2, c_4\}, \{c_3, c_4\}\}$，即 4 个因素之间存在两两交互作用，则该通话功能的一个两两组合覆盖表如表 3-2 所示。

**表 3-2 某电信系统交换机通话功能的一个两两组合覆盖表**

| 呼叫种类 | 资费方式 | 接入方式 | 状 态 |
|---|---|---|---|
| 市话 | 集中 | 专用分组 | 忙 |
| 长途 | 免费 | 环路 | 忙 |
| 国际 | 被叫方 | 综合业务 | 忙 |
| 市话 | 免费 | 综合业务 | 阻塞 |
| 长途 | 被叫方 | 专用分组 | 阻塞 |
| 国际 | 集中 | 环路 | 阻塞 |
| 市话 | 被叫方 | 环路 | 成功 |
| 长途 | 集中 | 综合业务 | 成功 |
| 国际 | 免费 | 专用分组 | 成功 |

可以看出，该表中包含任意两个因素的全部 9 个取值组合。该表的每一行就是一个两两组合测试用例。

类似地，可以定义三三组合覆盖表、四四组合覆盖表等。特别地，如果部分测试输入变量之间存在两两相互作用，另外部分的测试

输入变量之间存在三三相互作用，还可以定义可变力度的组合覆盖表。利用组合覆盖表进行测试设计的方法被称为组合测试。

### 3.1.2 组合测试的枚举本质

对于一个有 10 个测试输入变量、每个变量有 4 种可能取值的被测对象，如果进行组合全覆盖测试，变量间所有可能组合的数量是 $4^{10}=1048576$ 个，而基于两两组合覆盖表进行组合测试所需的用例数只有 31 个。一些测试者由此认为，组合测试是一种用于用例数量约简的测试设计方法，根本目的是缓解测试选择问题。实际上这种看法并不全面。

关于测试输入变量之间耦合作用的缺陷，有测试者针对 Mozilla 浏览器和 Apache 服务器进行了研究。结果显示，变量两两组合可以覆盖这类缺陷的 70%，三三组合可以覆盖 90%，六六组合则可以覆盖几乎全部。这一结论与测试人员的经验直觉相吻合，即变量组合相关的缺陷大多发生在较低的组合维数上。因此，如果测试人员的目的是检出与变量之间相互作用有关的缺陷，那么使用两两组合这样的小力度组合测试方法就可以检出大部分此类缺陷，并且使测试用例数量得到有效的约简。

以上是从“检出缺陷”角度出发得到的认识。接下来尝试切换到“评估质量”的角度。针对期望 $A$：“被测对象在所有测试输入变量的**所有可能组合**情况下功能都正常”，采用组合全覆盖测试方法，枚举所有可能的变量取值组合，在执行 $4^{10}=1048576$ 个测试用例之后，可以得到“被测对象在**所有可能组合**情况下正确”的结论；针对期望 $B$：“被测对象在所有测试输入变量的**两两组合**情况下功能都正常”，采用组合测试方法，枚举所有可能的变量取值并两两组合，在执行 31 个测试用例之后，可以得到“被测对象在**两两组合**情况下正确”的结论。确实，相对于组合全覆盖测试方法，组合测试的用例规模要小得多。但其主要原因是二者针对的期望不同，期望 $B$ 的测试输入空间规模要远小于期望 $A$。相应地，二者质量评估的结论也不同。然而站在测试设计思想的角度来看，二者本质上并无二致，都是“枚举”所

有可能的具体事件。

### 3.1.3 贪心法

前面已经提到过，以枚举方式进行的测试，容易造成测试规模的失控。仍然考虑 3.1.2 节中的例子。为了枚举所有测试输入变量的两两组合事件，如果在每个测试用例中只覆盖其中一个事件，那么需要的测试用例数为 $4^2 \cdot C_{10}^2 = 720$ 个。在很多情况下，这样的测试规模仍然是不可接受的。

事实上，以这种方式生成的用例集中存在大量冗余，因为每个测试用例本可以同时覆盖多个两两组合事件。组合测试用例约简的出发点就是在一个用例里覆盖尽可能多的两两组合事件，从而最大程度减少用例数量。

进行组合测试用例约简的方法很多，包括贪心法、数学方法、启发式搜索方法以及混合算法等，其中最经典的是贪心法。下面以一个简单的例子介绍贪心法的主要思路。

设被测对象有 3 个测试输入变量，变量 $A$ 有两个可能取值 $A_1$ 和 $A_2$，变量 $B$ 有两个值 $B_1$ 和 $B_2$，变量 $C$ 有 3 个值 $C_1$、$C_2$ 和 $C_3$。测试人员希望以尽可能少的用例覆盖所有变量的两两组合情况。

对于变量 $A$ 和 $B$，测试用例集$\{(A_1,B_1),(A_1,B_2),(A_2,B_1),(A_2,B_2)\}$覆盖了所有两两组合情况。进而考虑变量 $C$。首先将已有的这组测试用例进行水平扩充。由于 $C$ 有 3 个取值 $C_1$、$C_2$ 和 $C_3$，将原测试用例集中前 3 个测试用例分别扩充为$\{(A_1,B_1,C_1),(A_1,B_2,C_2),(A_2,B_1,C_3)\}$。此时未被覆盖的两两组合遗漏项有$\{(A_2,C_1),(B_2,C_1),(A_2,C_2),(B_1,C_2),(A_1,C_3),(B_2,C_3)\}$。现在要从 $C_1$、$C_2$ 和 $C_3$ 中选一个加到$(A_2,B_2)$上，如果选 $C_1$，即扩充为$(A_2,B_2,C_1)$，可以覆盖遗漏项$(A_2,C_1)$和$(B_2,C_1)$；而选择 $C_2$、$C_3$，都只能使扩充后的测试用例覆盖一个遗漏项，因此决定选择 $C_1$，并将其扩充为$(A_2,B_2,C_1)$。

通过上一步水平扩充后，还有遗漏项$\{(A_2,C_2),(B_1,C_2),(A_1,C_3),(B_2,C_3)\}$未被覆盖，为了覆盖遗漏项$(A_2,C_2)$，产生测试用例

$(A_2,—,C_2)$，为了覆盖$(B_1,C_2)$，将$(A_2,—,C_2)$修改为$(A_2,B_1,C_2)$；为了覆盖$(A_1,C_3)$，产生测试用例$(A_1,—,C_3)$，为了覆盖$(B_2,C_3)$，将$(A_1,—,C_3)$修改为$(A_1,B_2,C_3)$。

至此，被测对象三个变量的所有两两组合情况都得到了覆盖，共需 6 个测试用例。

可见，贪心法的原则就是“每一步都谋取最大的收益”：在扩充测试用例集的过程中，随时保持对“未覆盖组合”的关注，每一次扩充都尽可能覆盖最多的“未覆盖组合”，以期用最少的测试用例实现组合的全覆盖。当然，贪心法的局限性也很明显，因为在每次做决策时，贪心法只考虑当前的局面，而不考虑长远的利益，所以得到的可能并非最优解。

### 3.1.4 排除法

在一些实际情况下，可能无法获知测试输入变量之间有怎样的相互作用关系，但是可以确定某些变量之间没有相互作用关系。这时就可以利用排除法开展组合测试。下面以电路领域的实践为例进行说明。

电路的测试输入变量是给到输入引脚的电平信号，有“0”和“1”两种取值。测试设计的目的，是通过尽量少的测试用例，来验证在各种输入组合情况下，电路的输出都正确。假设已知电路的输出函数，即输出电平信号与各个测试输入变量之间的关系。这时可以遵循如下原则来约简组合测试用例：如果两个测试输入变量从来没有在同一个输出函数中出现，那么就可对这两个变量施加相同的测试输入信号。换言之，如果两个测试输入变量不会对任何输出产生耦合作用，那么就可以排除这两个变量的组合测试用例。例如，某电路系统的输入/输出情况如图 3-1 所示。

其中 $a\sim g$ 是测试输入变量，$f_1\sim f_5$ 是输出，并已知每个输出与输入变量的关联关系，例如 $f_1$ 只与 $a$、$b$、$e$ 有关，$f_5$ 只与 $e$、$f$ 有关，等等。如果进行组合全覆盖测试，所需组合测试用例数是 $2^7=128$ 个。由于某些测试输入变量之间并不存在耦合关系，这些组合

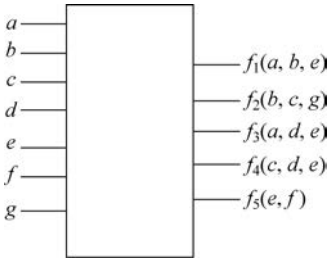


**图 3-1 某电路系统的输入/输出情况**

用例中有相当一部分是冗余的。遵循上述测试用例约简原则，可以按如下步骤排除掉这些冗余用例。

(1) 首先，构造 $m$ 行 $n$ 列的输入/输出关联矩阵，$m$ 是原始输出数，$n$ 是电路的原始输入数。矩阵中每一行对应一个原始输出，每一列对应一个原始输入。如果原始输出与某个原始输入相关，则矩阵中对应的单元值为 1，否则为 0。本例中电路系统的输入/输出关联矩阵如图 3-2 所示。

(2) 其次，对关联矩阵的各列重新排序，并按列进行分块。排序和分块的目标，是使所有输出函数在每个分块中不受两个或两个以上原始输入的影响。也就是说，要使每一个矩阵分块的每一行中 1 的个数小于或等于 1。对于本例中的关联矩阵，一种可行的排序和分块方式如图 3-3 所示。

| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | $f_1$ |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | $f_2$ |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | $f_3$ |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | $f_4$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | $f_5$ |

**图 3-2 输入/输出关联矩阵**

| Ⅰ | | | Ⅱ | | Ⅲ | | |
|---|---|---|---|---|---|---|---|
| $a$ | $c$ | $f$ | $b$ | $d$ | $e$ | $g$ | |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | $f_1$ |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | $f_2$ |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | $f_3$ |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | $f_4$ |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | $f_5$ |

**图 3-3 对输入/输出关联矩阵各列的重新排序和分块**

(3) 最后，在分块后的关联矩阵的基础上，构造特征关联矩阵。具体方法是：将关联矩阵每一行中每个分块的多个列合并成一个

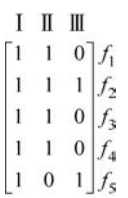


**图 3-4　特征关联矩阵**

列，并填入该行该分块中 1 的个数。本例中得到的特征关联矩阵如图 3-4 所示。

根据测试用例约简原则，第 $I$ 分块中的变量 $a$、$c$、$f$ 可以施加相同的测试输入信号，第Ⅱ分块中的变量 $b$ 和 $d$、第Ⅲ分块中的变量 $e$ 和 $g$ 也是如此。显然，$2^3=8$ 个组合测试用例即可枚举测试此电路。

## 3.2　分割测试

当测试人员想要枚举测试输入空间内的所有具体事件时，往往会遇到测试规模失控的问题。分割测试的思路是，如果可以把整个测试输入空间分割成一些子空间，任一子空间中所有的测试输入点对被测对象期望的影响方式都是相同的，那么就可以在每个子空间中只选择一个点作为测试用例，从而大大缓解测试选择问题。可见，分割测试的本质是借由对所有子空间的枚举，间接实现对整个测试输入空间内所有具体事件的枚举。

### 3.2.1　测试输入空间的分割

实施分割测试的关键问题是如何对测试输入空间进行分割。理想的分割结果是：在每一个分割出的子空间中，所有测试输入点以相同的方式影响期望的实现。这样测试人员才能任选一个点代表所有点，以该点的测试结果代表子空间所有点的测试结果，称这样的子空间为同质子空间。

> **定义：同质子空间**
>
> 如果一个子空间中存在能激发某个缺陷的测试输入点，当且仅当该子空间中所有测试输入点都能激发该缺陷，称该子空间为同质子空间。

同质子空间的定义有两层内涵：

(1) 被测对象以同样的方式响应同质子空间中所有测试输入点。

(2) 响应方式要么都正确，要么都错误。

被测对象如何响应测试输入，取决于被测对象的具体实现；而响应方式正确与否，取决于被测对象期望。也就是说，测试人员必须综合分析来自现实和理想的信息，才有可能从测试输入空间划分出真正的同质子空间。下面以软件领域的分割测试为例进一步解释。

对于软件而言，被测程序以特定的路径来响应某一测试输入。路径相同，则意味着程序对输入的响应方式相同。而路径是否正确，需要根据软件规约来判断。软件规约中包含了程序的期望信息——对于不同的输入，定义了不同的期望处理方式。如果程序的实现与预期相符，程序会为每一种期望处理方式设定一条正确的路径。如果程序的实现与预期不符，那么路径与期望处理方式就无法对齐。

因此，可以按以下三个步骤进行同质子空间的分割：

(1) 将整个测试输入空间按程序流程图中的路径划分子空间，每个子空间中的测试输入点触发相同的路径。

(2) 将整个测试输入空间按软件规约中的期望信息划分为子空间，每个子空间中的测试输入点对应相同的期望处理方式。

(3) 将以上两步得到的子空间进行交叉。

例如，某程序只有一个整形的输入变量，原始的测试输入空间包含所有整数。从程序流程图来看，所有正整数走一条路径，所有负整数走一条路径，零走一条路径；从该程序的规约来看，奇数和偶数有着不同的具体处理要求。按上述步骤对测试输入空间进行分割，最终得到五个子空间：正奇数、负奇数、正偶数、负偶数、零。这样划分出的子空间就是同质子空间。测试人员可以从这些子空间中随意选取一个输入数据进行测试，测试结果足以代表该子空间内的所有测试输入点。

如果程序正确，那么步骤(1)与步骤(2)得到的分割结果应该是相同的。分割结果存在差异的地方，往往是检出缺陷的“甜区”。当

然在实际工程中，程序结构通常建立在规约的基础上，因此分割结果的差异一般不会特别突出。

再举个更实际的例子。某个折后价计算程序的规约描述为："一家公司生产X和Y两种产品，X单价为5元，Y单价为10元。订单中包含对X和Y的订购数量。折扣方式是：如果总价超过200元，则折扣为5%；如果总价超过1000元，则折扣为20%；如果订购X的数量超过30件，则额外再折扣10%。程序输入为X和Y两种产品分别的订购数量，输出为这两种产品的折后价之和（向上取整）。"

该程序的实现代码如下：

```
public double discount_invoice(int x, int y) {
    int discount1 = 0;
    int discount2 = 0;
    if (x <= 30)
        discount2 = 100;
    else
        discount2 = 90;
    int sum = 5 * x + 10 * y;
    if (sum <= 200)
        discount1 = 100;
    else if (sum <= 1000)
        discount1 = 95;
    else
        discount1 = 80;
    return (Math.ceil(sum * discount1 * discount2 / 10000));
}
```

仍然按照上述过程，对测试输入空间进行分割。首先进行步骤(1)，按程序路径划分子空间。该程序中有六条路径，如图3-5所示。

每条路径都有一个与测试输入数据相关的路径条件，当且仅当测试输入数据满足该条件时，这条路径才会被执行。换言之，满足同一个路径条件的测试输入数据，触发的程序路径是相同的。因此在步骤(1)中，可以按路径条件对测试输入空间进行分割。本例中各执行路径的路径条件如表3-3所示。

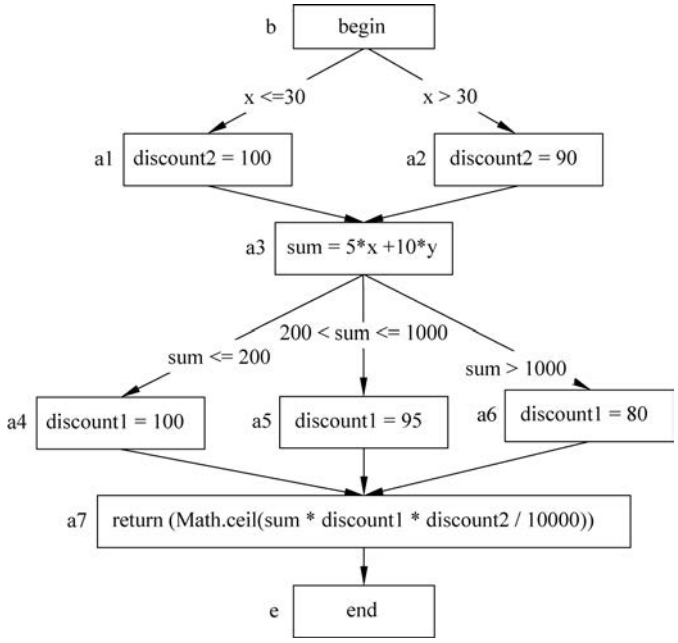


**图 3-5 示例程序的执行路径**

**表 3-3 各执行路径的路径条件**

| 路 径 | 路 径 条 件 |
|---|---|
| (b,a1,a3,a4,a7,e) | x≤30,(5 * x+10 * y)≤200 |
| (b,a1,a3,a5,a7,e) | x≤30,200<(5 * x+10 * y)≤1000 |
| (b,a1,a3,a6,a7,e) | x≤30,1000<(5 * x+10 * y) |
| (b,a2,a3,a4,a7,e) | 30<x,(5 * x+10 * y)≤200 |
| (b,a2,a3,a5,a7,e) | 30<x,200<(5 * x+10 * y)≤1000 |
| (b,a2,a3,a6,a7,e) | 30<x,1000<(5 * x+10 * y) |

接下来进行步骤(2),按规约划分子空间。对规约中的期望信息进行梳理,提取出如下的期望处理方式:

(1) 若 x≥0 且 y≥0,则 sum=5 * x+10 * y。

(2) 若 sum≤200,则 discount1=100。

(3) 若 sum>200 且 sum≤1000,则 discount1=95。

(4) 若 sum＞1000，则 discount1＝95。

(5) 若 0＜x≤30，则 discount2＝100。

(6) 若 x＞30，则 discount2＝90。

(7) 最终输出的折扣价之和为 ceil(sum * discount1 * discount2/10000)。

其中，x 是顾客订购 X 产品的数量；y 是顾客订购 Y 产品的数量；sum 是订单折前价；discount1 和 discount2 是两类折扣的百分比；ceil()是向上取整的函数。

根据规约描述的期望处理方式，当 x 和 y 满足 x≤30 且 5 * x＋10 * y≤200 时，输出应该是 5 * x＋10 * y。也就是说，对于输入数据子集{(x,y)|x≤30,5 * x＋10 * y≤200}中的所有数据，期望的处理方式是相同的，因此这个数据子集就构成了一个子空间。通过类似的分析，我们将测试输入空间划分为 A、B、C、D、E、F 六个子空间，如图 3-6 所示。

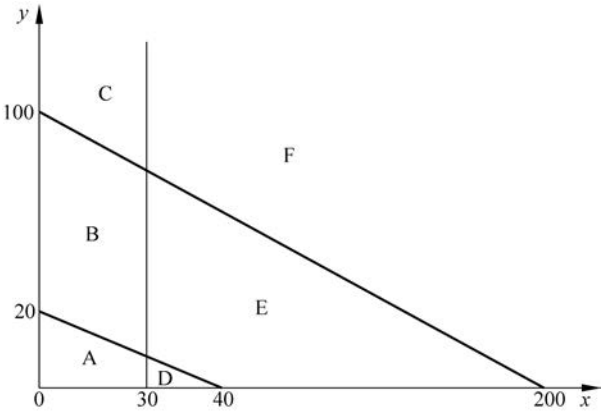


**图 3-6　测试输入空间划分结果**

最后进行步骤(3)，将步骤(1)和(2)得到的子空间进行交叉。这时发现，步骤(1)和(2)得到的子空间分割结果大体相同，差异仅仅在于步骤(1)的分割结果丢掉了 $x$ 轴和 $y$ 轴两个边界。而正是这微小的差异揭示了程序中隐藏的缺陷：遗漏了对输入变量非负合法性的

判断。

在真实的工程实践中，这种分割方法往往会遇到一个棘手的问题，就是程序路径的数量过于庞大，每条路径的构成也过于复杂，以至于步骤(1)无法施行。这解释了为什么对于大型程序进行分割测试时，测试者往往只基于期望信息来分割测试输入空间，也就是采用所谓的“等价类划分法”。这种做法对很多测试者来说习以为常，并深以为然。但测试人员需要清醒地认识到，“等价类划分”方法有一个重要的假设，即如果按处理方式的不同对输入数据进行分类，那么程序最终实现的分类结果完全等同于规约所期望的分类结果。显然，这个假设并非一定成立。仅仅根据规约信息划分的子空间并不是同质子空间。所谓的“等价类”，其中的测试输入并非真正等价。没有对被测程序路径结构的考察，纯黑盒的分割测试充其量只是一个阉割版。

相对来说更好的做法是，在基于规约的分割基础上引入一部分程序结构信息，通过交叉的方式得到更接近“同质”的子空间。

### 3.2.2 基于缺陷的分割测试

同质子空间的分割要求过于严苛，使得理想的分割测试难以实现。通过枚举子空间来证明被测对象的正确性，似乎只具有方法论层面的意义。不仅在软件领域，其他领域亦是如此。因此，可以考虑借助分割测试来实现另一个目标，也就是检出缺陷。以检出缺陷为目的的分割测试，称为基于缺陷的分割测试。

理想的分割测试，每个子空间中只需要选取一个测试输入点；而基于缺陷的分割测试，一般无法分割出同质子空间，因此需要从每个子空间中多采样一些点，来提高检出缺陷的概率。假设整个测试输入空间 $D$ 被分割为 $k$ 个子空间 $D_1, D_2, \cdots, D_k$，各子空间的规模分别是 $d_1, d_2, \cdots, d_k$，包含失效测试输入点的数量分别是 $m_1, m_2, \cdots, m_k$，失效测试输入点的概率密度分布分别是 $\theta_1, \theta_2, \cdots, \theta_k$。

尽管基于缺陷的分割测试并不苛求子空间的“同质”，但测试人员仍然希望子空间中的所有测试输入点对被测对象期望的影响方式

应该尽量相似。从检出缺陷的目的出发，如果一些测试输入点揭示某类缺陷的能力类似，宜将其划入同一个子空间。具体的策略可能有很多种，其差别在于关注的缺陷类型不同。在此基础上，可以假定子空间中失效测试输入点是均匀分布的，即 $\theta_i = m_i / d_i$。也就是说，从子空间中任意选取一个测试输入点作为测试用例，检出缺陷的概率是 $\theta_i = m_i / d_i$。

假设测试者从子空间 $D_i$ 中按均匀分布随机选取的测试用例数为 $n_i$。记分割测试在整个测试输入空间中能检出至少一个缺陷的概率为 $P_p$，则有 $P_p = 1 - \pi_{i=1}^{k} (1 - \theta_i)^{n_i}$。既然以检出缺陷为目的，自然希望 $P_p$ 尽可能高。

在软件领域，基于缺陷的分割测试根据容易出错的逻辑、相对复杂的数据结构划分子空间。譬如，当测试人员认为程序分支逻辑结构复杂易错时，可以把所有能覆盖某个分支的测试输入数据归拢到一个子空间中。这就是所谓的分支测试策略。如果所有分支子空间的缺陷密度 $\theta_i$ 都很低(比如整个子空间中只有边界上的点会触发缺陷)，$P_p$ 就很低。这种情况下，分支测试就是一种很差的策略。但是如果某分支子空间的多数输入数据都会触发缺陷，那么 $P_p$ 就会比较高。因此对基于缺陷的分割测试来说，判断分割策略优劣的核心指标是子空间中失效测试输入点的密度。

考虑以下程序：

```
public boolean branchTest(int x){
   if( - 10 <= x && x <= 9){
        if(x >= 0){
            //P1
        }
        else{
            //P2
        }
    }
    return false;
}
```

变量 x 的取值范围为[－10,9]。P1 和 P2 都是线性代码片段。使用分支测试策略,划分出的两个子空间分别是 $D_1$＝[－10,－1]和 $D_2$＝[0,9]。假设该被测程序唯一的缺陷隐藏在两个分支子空间的边界上,即 $x=0$ 可以触发该缺陷。那么两个子空间中失效测试输入点的概率密度分别为 $\theta_1=0,\theta_2=0.1$。另外,假设总的用例数为 6,在 $D_1$ 和 $D_2$ 中分别按均匀分布随机选取 3 个用例。此时 $P_p=1-1*0.9^3=0.271$。

如果测试人员能意识到被测程序中存在边界值缺陷的可能性较高,可以对以上分割策略进行优化,将分支的边界值纳入一个单独的子空间中,分割的结果为 $D_1$＝[－10,－2],$D_2$＝[－1,0],$D_3$＝[1,9],各子空间中失效测试输入点的概率密度分别为 $\theta_1=0,\theta_2=0.5,\theta_3=0$,如果总的用例数仍然为 6,在 $D_1$、$D_2$ 和 $D_3$ 中分别按随机分布随机选取 2 个用例,此时 $P_p=1-1*0.5*1=0.5$,相比优化前有了大幅提升。这种分割策略也就是通常所说的边界值测试策略。

当测试人员在实践中应用基于缺陷的分割测试时,如果检出缺陷的效果很差,原因往往是只简单套用该方法的“形”,比如仅仅根据程序控制流或数据流划分子空间;而没有抓住该方法的“神”,即从“被测对象容易包含哪些缺陷”入手来分割子空间。

### 3.2.3 等比例采样策略

对基于缺陷的分割测试来说,既然需要在子空间中按均匀分布随机选取测试输入点,那么这种方法与一般的随机测试有何区别呢?所谓“随机测试”,指的是直接在整个测试输入空间上按均匀分布随机选取测试输入点,相比分割测试来说更加简单。有研究表明,在测试用例数一定的前提下,分割测试检出缺陷的能力与随机测试差别不大。如果考虑到分割子空间的成本,分割测试的投入产出比甚至不如随机测试。而进一步的研究发现,如果采用等比例采样策略,也就是在分割测试中按相同比例对每个子空间进行随机采样,则对于某个失效测试输入点而言,分割测试选中该点的概率要高于采样比例相同的随机测试。

也就是说，将测试输入空间划分为多个子空间后，采样比例不变，但是检出某个缺陷的概率变高了。这是个有趣的结论，因为乍看之下似乎违反直觉。下面给出一个简单的证明。

设整个测试输入空间的规模为 $d$，随机测试从整个测试输入空间中选取的测试用例总数为 $n$。随机测试在每次选择上测试用例时，所有测试输入点被选中的概率是相同的，均为$\frac{1}{d}$。因此，任意失效测试输入点被测试集选中至少一次的概率是 $P_r=1-\left(1-\frac{1}{d}\right)^n$。

设分割测试得到的各子空间的规模分别是 $d_1, d_2, \cdots, d_k$，有 $d=\sum_{i=1}^{k} d_i$。分割测试从子空间 $D_i$ 中选取的测试用例数为 $n_i$，且 $n=\sum_{i=1}^{k} n_i$。如果使用“等比例采样策略”，每个子空间中选取测试用例的比例与整体比例一致，即$\frac{n_i}{d_i}=\frac{n}{d}$。另外，已知 $f(x)=\left(1-\frac{1}{x}\right)^x$ 在 $x \geqslant 1$ 范围内为单调增函数。那么在子空间 $D_i$ 中，任意失效测试输入点被测试集选中至少一次的概率为

$$
\begin{aligned}
P_s &= 1-\left(1-\frac{1}{d_i}\right)^{n_i} \\
&= 1-\left[\left(1-\frac{1}{d_i}\right)^{d_i}\right]^{\frac{n_i}{d_i}} \\
&= 1-\left[\left(1-\frac{1}{d_i}\right)^{d_i}\right]^{\frac{n}{d}}
\end{aligned}
$$

我们知道，函数 $f(x)=\left(1-\frac{1}{x}\right)^x$ 的曲线如图 3-7 所示。

可见 $f(x)=\left(1-\frac{1}{x}\right)^x$ 在 $x \geqslant 1$ 范围内为单调增函数。因此：

$$
\begin{aligned}
P_s &= 1-\left[\left(1-\frac{1}{d_i}\right)^{d_i}\right]^{\frac{n}{d}} \\
&> 1-\left[\left(1-\frac{1}{d}\right)^{d}\right]^{\frac{n}{d}}
\end{aligned}
$$

$$=1-\left(1-\frac{1}{d}\right)^{n}$$

$$=P_r$$

由此得证，对于某个缺陷而言，基于等比例采样策略的分割测试将其检出的概率确实比随机测试高。

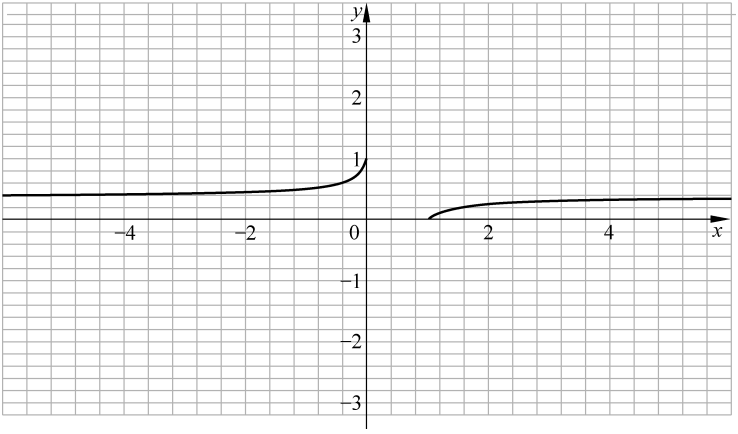


**图 3-7 函数 $f(x)=\left(1-\frac{1}{x}\right)^{x}$ 的曲线图**

下面用一个简单的例子来解释等比例采样策略的效果。假设测试输入空间的规模为 $d=4$，随机测试选取的用例总数为 $n=2$，采样比例为$\frac{1}{2}$。测试输入空间中有一个失效测试输入点，则随机测试选中该点至少一次的概率是 $1-\left(1-\frac{1}{4}\right)^{2}=\frac{7}{16}$；而在基于等比例采样策略的分割测试中，假设 $d_1=d_2=2$，则 $n_1=n_2=2\times\frac{1}{2}=1$，子空间分割结果如图 3-8 所示。

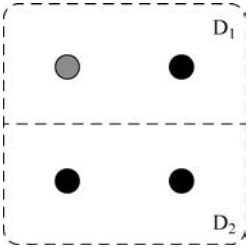


**图 3-8 子空间分割结果**

易知，该失效测试输入点被选中至少一次的概率是$\frac{1}{2}$，相对随机测试有所提高。

## 3.3 模型检验

一旦试图用枚举的方法来判定被测对象的正确性，测试人员势必要面对大规模测试集带来的测试成本问题。这时，一些测试者想到的办法是利用计算机仿真技术。只要能让计算机理解被测对象的理想与现实，就有可能利用日益充沛且廉价的算力，在一些场景中实现自动化的枚举测试。

在芯片领域，这是一种普遍的做法，其过程大致如图 3-9 所示。


**图 3-9　芯片的枚举测试过程**

将寄存器传输级的芯片设计视为被测对象，由形式验证驱动器对其施加所有可能的测试输入，由性质检查器观察其输出，一旦与期望的性质不符，则将 fails()信号置 1。如果在整个枚举测试过程中，fails()信号始终为 0，则说明芯片逻辑设计是正确的。

模型检验是另一种借助计算机仿真手段的枚举测试方法，也是主流的形式化验证手段之一。这种方法通过穷尽搜索被测对象模型所有可能的状态，检验被测对象的行为是否满足质量特性期望。对模型检验方法来说，“理想”是以形式化手段描述期望的规约，“现实”是以形式化手段描述被测对象行为的模型。如果在形式化模型的所有状态下，被测对象的行为都满足形式化规约，就证明了被测对象的形式化模型是正确的；否则，模型检验方法会给出至少一个反例来说明形式化模型为何不正确，以便后续修正。值得一提的是，如果在被测对象模型中植入一系列模拟缺陷，并利用模型检验方法给出反例的能力，就可以生成具有缺陷预防能力的测试集。

早期的模型检验主要用于硬件系统的测试。随着技术的发展，模型检验的应用范围逐步扩大，涵盖了安全关键的软硬件系统测试、协议测试等。譬如，通过模型检验可以证明某个通信协议的实现不会发生死锁。

模型检验的目的是解决正确性判定问题，需要逐一测试模型的所有状态，本质上体现的是枚举的思想。当然，为了缓解状态空间爆炸问题，也有一些妥协的变种方法，如界限模型检验、面向覆盖率的随机模型检验等。

模型检验的基本方法如图 3-10 所示。

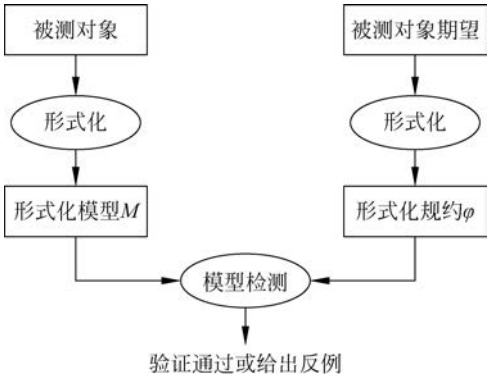


**图 3-10 模型检验的基本方法**

其中包括三个主要步骤：

(1) 建立被测对象的形式化模型 $M$，通常采用迁移系统、有穷自动机、Petri 网等形式。

(2) 建立被测对象期望的形式化规约 $\varphi$，通常采用时序逻辑公式的形式。

(3) 枚举验证，也就是基于状态空间搜索的方法遍历 $M$ 的所有可能状态，检查是否与 $\varphi$ 相符。

### 3.3.1 形式化模型

被测对象的形式化模型并不等于被测对象本身。对形式化模型进行测试，实质上是一种妥协，目的是借助模型检验方法来实现枚举测试。为了保证测试结论能够最大程度适用于被测对象本身，模型应该尽可能忠实地反映被测对象的行为特征或质量特性。同时，从模型检验的可行性及执行效率考虑，模型还应该具备足够

的抽象程度，屏蔽与目标特性关系不大的实现细节。可见，建模是一项需要测试者充分权衡的工作：建模的抽象程度过高，可能会造成缺陷遗漏；抽象程度过低，模型就可能变得过于复杂，影响模型检验的效率。

模型检验主要用于状态转换系统，常用的形式化模型种类包括迁移系统、有穷自动机、Petri 网等。

#### 3.3.1.1 迁移系统

迁移系统模型用于表征被测对象状态及状态之间的迁移。

迁移系统的状态指的是被测对象在某个时刻的行为特征或属性特征。例如，交通灯的状态指的是灯当前的颜色；计算机程序的状态指的是所有程序变量当前的值，以及程序计数器当前的值（该值指定了下一个将被执行的程序语句）；时序电路的状态指的是寄存器当前的值。

迁移系统的迁移表示被测对象怎样从一个状态转换为另一个状态。交通灯的状态迁移指的是交通灯从一种颜色转换到另一种颜色；计算机程序的状态迁移指的是一些程序变量值的改变，或者某一行语句的执行；时序电路的状态迁移指的是寄存器值的改变。

在每种状态下，迁移系统模型可以执行一系列迁移中的某一个，这一系列迁移就是该状态可行的迁移，其他迁移是不可行的。每一个可行的迁移可以使系统达到一个新的状态。只要存在可行迁移，这个过程就会不断继续。

更具体地说，一个迁移系统的直观行为描述如下：如果 $s$ 是被测对象的当前状态，通过迁移关系→发生状态转变，迁移到新的状态 $s'$，这一过程用 $s \xrightarrow{\tau} s'$ 表示，其中 $\tau$ 表示迁移动作，$\tau$ 对 $s$ 来说是出迁移，对 $s'$ 来说是入迁移。状态迁移会在状态 $s'$ 再次发生，直到终止于一个没有出迁移的状态。当一个状态有多个出迁移时，下一个迁移的选择完全是非确定的，并且也无法知道某个迁移被选择的可能性有多大。类似地，当初始状态集由多个状态组成时，开始状态的选择也是不确定的。

迁移系统模型常以图的形式来描述。例如，饮料自动售货机的迁移系统模型如图 3-11 所示。

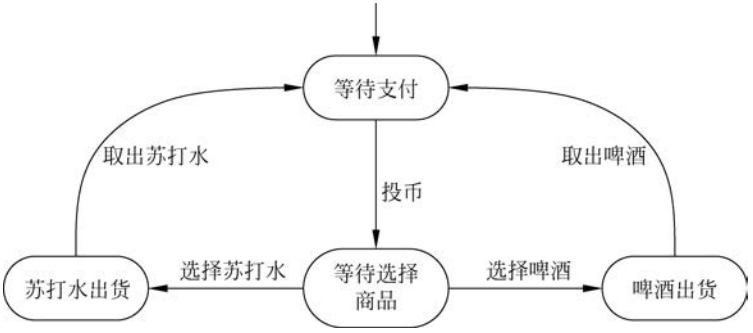


**图 3-11 饮料自动售货机的迁移系统模型**

自动售货机可以卖啤酒或苏打水。状态用圆角矩形表示，迁移用带标记的有向边表示，状态的名称写在圆角矩形里面，可能的状态包括{等待支付，等待选择商品，苏打水出货，啤酒出货}。初始状态用一个没有来源的进入箭头表示，即“等待支付”。可能的迁移动作包括{投币，选择苏打水，选择啤酒，取出苏打水，取出啤酒}。可能的状态迁移包括：

(1) 等待支付$\xrightarrow{\text{投币}}$等待选择商品。

(2) 等待选择商品$\xrightarrow{\text{选择苏打水}}$苏打水出货。

(3) 等待选择商品$\xrightarrow{\text{选择啤酒}}$啤酒出货。

(4) 苏打水出货$\xrightarrow{\text{取出苏打水}}$等待支付。

(5) 啤酒出货$\xrightarrow{\text{取出啤酒}}$等待支付。

#### 3.3.1.2 有穷自动机

相比迁移系统，有穷自动机模型更注重描述测试输入序列对被测对象状态的影响。

有穷自动机内部具有有限个状态，状态概括被测对象对历史输入序列的处理结果。被测对象只需根据当前所处的状态和当前的输

入，就可以决定后继行为，同时状态也将发生变化。开关网络、程控电话交换机、电梯控制装置、文本编辑程序、编译技术中的词法分析程序都可以建模为有穷自动机。

举一个简单的例子。两相开关装置的有穷自动机模型如图 3-12 所示。

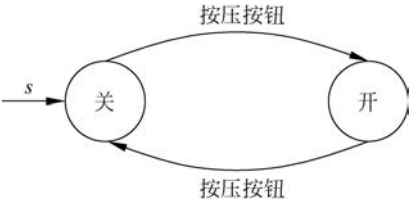


**图 3-12　两相开关装置的有穷自动机模型**

这个装置处在“开”或“关”状态时，用户都可按压按钮，该按钮根据开关状态起不同的作用。如果开关处于“关”的状态，按下按钮就会变为“开”的状态；如果开关处于“开”的状态，按下按钮就会变为“关”的状态。在有穷自动机模型中，状态用圆圈表示，测试输入用状态之间的箭头表示。图 3-12 中状态之间的两个箭头的含义是：无论被测对象处于什么状态，当收到输入“按压按钮”时，就进入另一个状态。标记为 *s* 的箭头指向的是初始状态。在这个例子中，初始状态是“关”。模型中还可以包含终止状态，用双边圆圈表示。

有穷自动机模型具有以下几个主要特点：

(1) 模型具有有限个状态，不同的状态代表不同的意义。按照实际的需要，模型可以在不同的状态下完成规定的任务。

(2) 将测试输入中出现的信息汇集在一起，构成一个字母表。模型处理的测试输入序列，可视为由字母表上的字符组成的输入字符串。

(3) 模型在任何一个状态下，从输入字符串中读入一个字符，根据当前状态和这个字符变迁到新的状态。

有穷自动机的工作机制如图 3-13 所示。

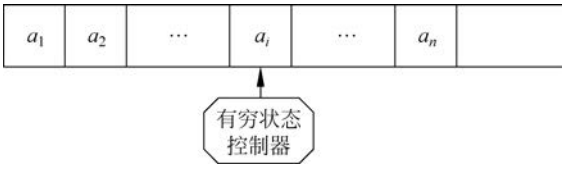


**图 3-13 有穷自动机的工作机制**

输入带上有一系列的方格，每个方格可以存放一个字符。输入字符串从输入带左端点开始存放。有穷状态控制器有一个读头，用来从输入带上读入字符。每读入一个字符，就根据当前状态和读入的字符改变有穷状态控制器的状态，并将读头向右移动一格，指向下一个待读入的字符。

如两相开关的例子所示，有穷自动机模型通常用状态迁移图表示。状态迁移图的顶点对应有穷自动机的状态，若在测试输入 $a$ 下模型从状态 $\alpha$ 迁移到状态 $\beta$，那么在状态迁移图中就有一条标以 $a$ 的弧线从状态 $\alpha$ 指向状态 $\beta$。若模型在输入字符串 $s$ 的驱动下，成功地从初始状态迁移到终止状态，则认为模型可以成功处理输入字符串 $s$。

再举一个相对复杂的例子，对于图 3-14 所示的有穷自动机模型，标记为 $S$ 的箭头指向其初始状态 $q_0$，该状态同时也是终止状态。根据状态迁移图，可知该有穷自动机可以成功处理所有由 0 和 1 组成的、0 的个数和 1 的个数都是偶数的输入字符串。

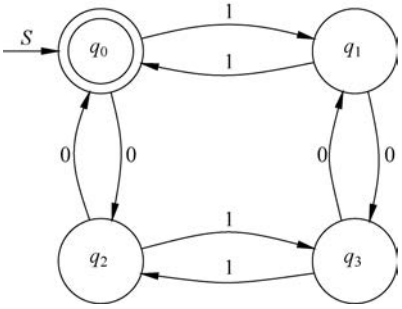


**图 3-14 某有穷自动机的状态迁移图**

#### 3.3.1.3 Petri 网

Petri 网是一种关注资源流动的形式化模型。资源可以指物质资源，也可以指信息资源。Petri 网的形式要素包括：

(1) 库所：代表资源所处的位置，用圆形节点表示。Petri 网中所有库所的令牌储量构成的向量，可以表示模型的当前状态。

(2) 令牌：代表资源，用黑点表示。令牌可以从一个库所移动到另一个库所。

(3) 弧：库所和变迁之间的有向连接，可以赋予权重。

(4) 变迁：代表资源的流动，用方形节点表示。当一个变迁的所有输入库所都拥有令牌时，这个变迁才会发生，使得输入库所的令牌被消耗，输出库所中产生令牌。变迁是一种原子操作，消耗输入库所的令牌、在输出库所产生令牌是同时发生的，不可分割。

举一个简单的例子。图 3-15 是用 Petri 网模型表示的四季流转过程，其中库所 $P_1$、$P_2$、$P_3$、$P_4$ 分别表示春季、夏季、秋季、冬季。当库所中存在令牌时，表示当前正处于该库所对应的季节。例如初始标识为 $M_0=(1,0,0,0)$，表示当前处于春季。变迁 $t_1$、$t_2$、$t_3$、$t_4$ 则表示四季的流转，如变迁 $t_1$ 的发生表示由春天到夏天的季节变换。

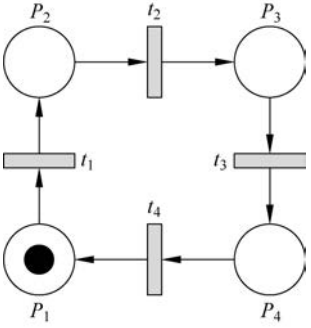


**图 3-15 四季流转过程的 Petri 网模型**

再举一个稍复杂的例子，用 Petri 网为消防员救火过程建立模型。假设在救火过程中，消防员除人手一只水桶外，无其他设备可

用。消防员使用桶从水源往火场运水，再从火场往水源运桶。可见，该过程中流动的资源有两类：桶和水。同时消防员有两种状态：运水或者运桶。

图 3-16 所示的 Petri 网模型可以描述单个消防员在救火过程中的状态变化。

图 3-16 中有两个库所 $P_1$ 及 $P_2$，库所 $P_1$ 存在令牌表示消防员正处于运水状态，库所 $P_2$ 存在令牌表示消防员正处于运桶状态。变迁 $t_1$ 和 $t_2$ 表示消防员从一种状态到另一种状态的转变。

在实际救火过程中，消防员一般都是团队作战。如果火场远、人员少，那么就需要将一组消防员排成一队，接力运水，以提高救火效率。这一过程的 Petri 网模型如图 3-17 所示。

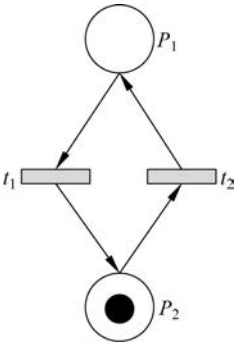


**图 3-16 单个消防员救火过程的 Petri 网模型**

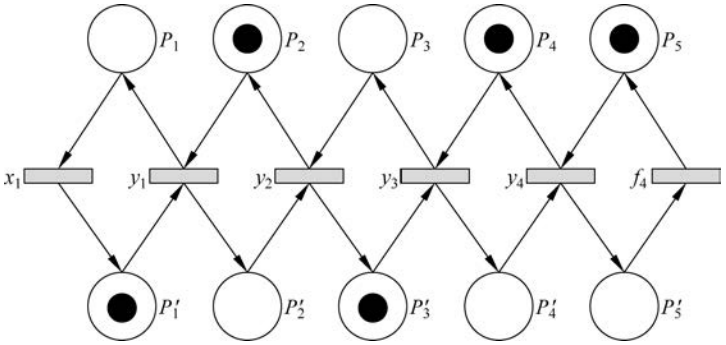


**图 3-17 消防队救火过程的 Petri 网模型**

库所 $P_i$ 存在令牌表示第 $i$ 名消防员正处于运水状态，库所 $P_i'$ 存在令牌表示消防员正处于运桶状态。变迁 $x_1$ 表示第 1 名消防员将水泼向火场；变迁 $y_i$ 表示运桶的第 $i$ 名消防员和运水的第 $i+1$ 名消防员相遇，他们交换水桶，调转方向，同时实现状态改变；变迁 $f_4$ 表示第 5 名消防员从水源处将空桶接满水。

可见，该模型中所有变迁都是在其发生条件具备时自然发生的，没有任何形式的控制，无须现场指挥，也没有时间限制。体力强的消防员可以多走点，体弱的消防员可以少走点，何时相遇就交换水桶，改变状态。

此外，该模型还可以描述并发。在第 1、2 名消防员交换水桶时，第 5 名消防员可能正在从水源地取水，这些动作是不相关的异步行为。在 Petri 网中不存在统一的时钟，如果两个变迁的输入库所集和输出库所集两两不相交，则这两个变迁所对应的事件之间没有因果关系，它们发生的先后在时间上无法区分。因此，Petri 网模型强调的是与资源流动有关的因果关系，通过这种因果关系来体现被测对象的行为特征。

### 3.3.2 形式化规约

模型检验通常用于验证与时序有关的被测对象期望。一个典型的例子是数据总线上的资源管理：总线仲裁器收到一个合理的总线占用请求信号后，经过特定周期，应正常返回总线授予应答信号。这类期望都是先声明一系列事件作为前置条件，然后再假定一系列稍后发生的事件作为需要验证的后置条件。

为了实现模型检验算法的自动化执行，我们需要以形式化规约的方式来描述被测对象期望。常见的面向时序的形式化规约包括线性时序逻辑和分支时序逻辑等。

#### 3.3.2.1 命题逻辑基础

首先简要介绍命题逻辑中最主要的一些定义和关系。

在命题逻辑中，唯一判断结果的简单陈述句称为原子命题，或简称命题。命题的判断结果称为命题的真值，可取“真”或“假”两种逻辑常量。真值为“真”的命题称为真命题，真值为“假”的命题称为假命题。真值可以变化的命题称为命题变项。将命题用逻辑连接词和圆括号按一定的逻辑关系联结起来的符号串称为命题公式。

设 $p$、$q$ 表示命题，$P$ 表示命题公式。常见的命题公式类型

包括：

(1) 合取式：$p$ 与 $q$，记作 $p \wedge q$。

(2) 析取式：$p$ 或 $q$，记作 $p \vee q$。

(3) 否定式：非 $p$，记作 $\neg p$。

(4) 蕴涵式：如果 $p$ 则 $q$，记作 $p \rightarrow q$。

(5) 等价式：$p$ 当且仅当 $q$，记作 $p \Leftrightarrow q$。

(6) 重言式：若 $P$ 无成假赋值，则称 $P$ 为重言式或永真式。

(7) 矛盾式：若 $P$ 无成真赋值，则称 $P$ 为矛盾式或永假式。

(8) 可满足式：若 $P$ 至少有一个成真赋值，则称 $P$ 为可满足的。

(9) 析取范式：仅由有限个简单合取式组成的析取式。

(10) 合取范式：仅由有限个简单析取式组成的合取式。

设 $A$、$B$ 是两个命题公式，若无论 $A$、$B$ 中的命题变项赋值为何，$A$ 和 $B$ 的真值都相同，则称 $A$ 和 $B$ 是等值的，记作 $A \Leftrightarrow B$。将命题公式 $A$ 推演为与之等值的命题公式 $B$ 的过程，称为等值演算。等值演算的基础是如下 16 组演算规律：

(1) 双重否定律：$\neg \neg A \Leftrightarrow A$。

(2) 幂等律：$A \Leftrightarrow A \vee A$；$A \Leftrightarrow A \wedge A$。

(3) 交换律：$A \vee B \Leftrightarrow B \vee A$；$A \wedge B \Leftrightarrow B \wedge A$。

(4) 结合律：$(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$；$(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$。

(5) 分配律：$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$；$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$。

(6) 德摩根律：$\neg (A \vee B) \Leftrightarrow \neg A \wedge \neg B$；$\neg (A \wedge B) \Leftrightarrow \neg A \vee \neg B$。

(7) 吸收律：$A \vee (A \wedge B) \Leftrightarrow A$；$A \wedge (A \vee B) \Leftrightarrow A$。

(8) 零律：$A \vee 1 \Leftrightarrow 1$；$A \wedge 0 \Leftrightarrow 0$。

(9) 同一律：$A \vee 0 \Leftrightarrow A$；$A \wedge 1 \Leftrightarrow A$。

(10) 排中律：$A \vee \neg A \Leftrightarrow 1$。

(11) 矛盾律：$A \wedge \neg A \Leftrightarrow 0$。

(12) 蕴涵等值律：$A \rightarrow B \Leftrightarrow \neg A \vee B$。

(13) 等价等值律：$A \leftrightarrow B \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$。

(14) 假言易位律：$A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A$。

(15) 等价否定律：$A \leftrightarrow B \Leftrightarrow \neg A \leftrightarrow \neg B$。

(16) 归谬律：$(A \rightarrow B) \wedge (A \rightarrow \neg B) \Leftrightarrow \neg A$。

#### 3.3.2.2 线性时序逻辑

线性时序逻辑将时间设想为一个线性序列，该序列由一系列离散的时刻组成，每个时刻对应被测对象状态序列 $\delta$：$s_0, s_1, s_2, \cdots$中的一个状态。时序逻辑以状态序列作为命题的论断对象，在状态序列上解释其真值，同一个时序逻辑公式在不同的时刻(即状态序列的不同位置)可能有不同的真值，这是时序逻辑区别于经典命题逻辑的重要特性。

与经典命题逻辑公式一样，线性时序逻辑公式中也有原子命题、逻辑常量、逻辑连接词等要素。在此基础之上，线性时序逻辑引入了时序算子，以描述与时间相关的期望。常用的时序算子包括：

(1) $\Box p$：意为“任一时刻 $p$ 为真”。

(2) $\Diamond p$：意为“某一时刻 $p$ 为真”。

(3) $\bigcirc p$：意为“下一时刻 $p$ 为真”。

(4) $p \cup q$：意为“$p$ 为真直到 $q$ 为真”。

(5) $p \omega q$：意为“$p$ 为真直到 $q$ 为真，或任一时刻 $p$ 为真”。

将被测对象期望表述为命题逻辑公式 $f$，对于被测对象状态序列 $\delta$ 中第 $j$ 个位置的状态 $s_j$，如果 $f$ 在 $s_j$ 上为真，则记作 $(\delta, j) \models f$；如果 $f$ 在 $s_j$ 上为假，则记作 $(\delta, j) !\models f$。再结合时序算子，即可表述在 $s_j$ 及其之后的状态序列上的期望，也就是基于线性时序逻辑的形式化规约。例如：

1. $(\delta, j) \models \Box f \Leftrightarrow \forall k \geqslant j, (\delta, k) \models f$

$\Box f$ 在时刻 $j$ 成立，当且仅当 $f$ 在时刻 $j$(含 $j$)之后的任一时刻均成立，如图 3-18 所示。

例如，假设在时刻 $j=4$ 之后都有 $x>3$ 成立，则线性时序逻辑公式“$\Box(x>3)$”在各时刻的真值如表 3-4 所示。

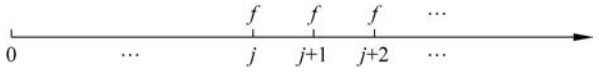


图 3-18 $(\delta, j) \models \Box f$

表 3-4 $\Box(x>3)$真值表

| $j$ | $x$ | $x>3$ | $\Box(x>3)$ |
|---|---|---|---|
| 0 | 1 | F | F |
| 1 | 3 | F | F |
| 2 | 2 | F | F |
| 3 | 4 | T | F |
| 4 | 3 | F | F |
| 5 | 5 | T | T |
| 6 | 4 | T | T |
| … | … | … | … |

显然,如果$\Box f$在时刻$j$成立,则它在$j$之后的任一时刻也成立。

2. $(\delta, j) \models \Diamond f \Leftrightarrow \exists k \geqslant j, (\delta, k) \models f$

$\Diamond f$在时刻$j$成立,当且仅当$f$在时刻$j$(含$j$)之后的某一时刻成立,如图 3-19 所示。


图 3-19 $(\delta, j) \models \Diamond f$

例如,假设在时刻$j=3$之后都有$x \neq 4$,则线性时序逻辑公式"$\Diamond(x=4)$"在各时刻的真值如表 3-5 所示。

表 3-5 $\Diamond(x=4)$真值表

| $j$ | $x$ | $x=4$ | $\Diamond(x=4)$ |
|---|---|---|---|
| 0 | 1 | F | T |
| 1 | 2 | F | T |
| 2 | 3 | F | T |

续表

| $j$ | $x$ | $x=4$ | $\Diamond(x=4)$ |
|---|---|---|---|
| 3 | 4 | T | T |
| 4 | 5 | F | F |
| 5 | 6 | F | F |
| … | … | … | … |

3. $(\delta,j)\models\bigcirc f\Leftrightarrow(\delta,j+1)\models f$

$\bigcirc f$ 在时刻 $j$ 成立，当且仅当 $f$ 在时刻 $j$ 的下一时刻 $j+1$ 成立，如图 3-20 所示。

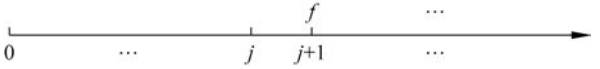


**图 3-20 $(\delta,j)\models\bigcirc f$**

例如，假设状态变量 $x$ 的取值以 0011 为周期循环往复，则线性时序逻辑公式“$(x=0)\wedge\bigcirc(x=1)$”在各时刻的真值如表 3-6 所示。

**表 3-6 $(x=0)\wedge\bigcirc(x=1)$真值表**

| $j$ | $x$ | $x=0$ | $x=1$ | $\bigcirc(x=1)$ | $(x=0)\wedge\bigcirc(x=1)$ |
|---|---|---|---|---|---|
| 0 | 0 | T | T | F | F |
| 1 | 0 | T | T | T | T |
| 2 | 1 | F | T | T | F |
| 3 | 1 | F | T | F | F |
| 4 | 0 | T | F | F | F |
| 5 | 0 | T | F | T | T |
| 6 | 1 | F | T | T | F |
| … | … | … | … | … | … |

4. $(\delta,j)\models f\cup g\Leftrightarrow\exists k\geqslant j,(\delta,k)\models g\wedge\forall i,j\leqslant i<k,(\delta,i)\models f$

$f\cup g$ 在时刻 $j$ 成立，当且仅当 $g$ 在时刻 $j$（含 $j$）之后的某一时刻 $k$ 成立，而 $f$ 在时刻 $j,\cdots,k-1$ 一直成立，如图 3-21 所示。

例如，假设状态变量 $x$ 的取值按自然数序列递增，则线性时序逻

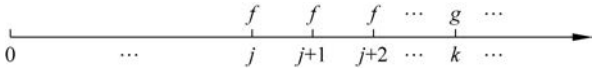


图 3-21 $(\delta, j) \models f \cup g$

辑公式“$(3 \leqslant x \leqslant 5) \cup (x = 6)$”在各时刻的真值如表 3-7 所示。

表 3-7 $(3 \leqslant x \leqslant 5) \cup (x = 6)$真值表

| $j$ | $x$ | $3 \leqslant x \leqslant 5$ | $x = 6$ | $(3 \leqslant x \leqslant 5) \cup (x = 6)$ |
|---|---|---|---|---|
| 0 | 1 | F | F | F |
| 1 | 2 | F | F | F |
| 2 | 3 | T | F | T |
| 3 | 4 | T | F | T |
| 4 | 5 | T | F | T |
| 5 | 6 | F | T | T |
| 6 | 7 | F | F | F |
| … | … | … | … | … |

5. $(\delta, j) \models f \omega g \Leftrightarrow (\delta, j) \models f \cup g \vee (\delta, j) \models \Box f$

例如，仍然假设状态变量 $x$ 的取值按自然数序列递增，则线性时序逻辑公式“$[(3 \leqslant x \leqslant 5) \vee (x \geqslant 8)] \omega (x = 6)$”在各时刻的真值如表 3-8 所示。

表 3-8 $[(3 \leqslant x \leqslant 5) \vee (x \geqslant 8)] \omega (x = 6)$真值表

| $j$ | $x$ | $(3 \leqslant x \leqslant 5) \vee (x \geqslant 8)$ | $x = 6$ | $[(3 \leqslant x \leqslant 5) \vee (x \geqslant 8)] \omega (x = 6)$ |
|---|---|---|---|---|
| 0 | 1 | F | F | F |
| 1 | 2 | F | F | F |
| 2 | 3 | T | F | T |
| 3 | 4 | T | F | T |
| 4 | 5 | T | F | T |
| 5 | 6 | F | T | T |
| 6 | 7 | F | F | F |
| 7 | 8 | T | F | T |
| 8 | 9 | T | F | T |
| … | … | … | … | … |

该公式在区间 2,…,5 成立是因为 $f \cup g$ 成立,在无穷区间 7,8,…成立是因为 $f$ 在该区间的所有时刻均成立。

下面来看一个线性时序逻辑的应用示例。对于一个用于称重的弹簧来说,其迁移系统模型如图 3-22 所示。

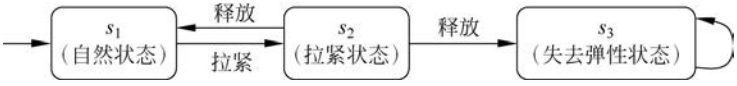


**图 3-22　弹簧的迁移系统模型**

拉紧弹簧可以使其伸长,释放拉力可以恢复其原始形状。如果拉力过大,弹簧还可能失去弹性,保持伸长的形状无法恢复。为了简化描述,用 $e$ 指代原子命题"弹簧伸长",用 $m$ 指代原子命题"弹簧失去弹性"。显然,当弹簧处于 $s_2$ 或 $s_3$ 状态时,$e$ 都成立;当弹簧处于 $s_3$ 状态时,$m$ 成立。

这个模型可能产生无穷多个状态序列,如:

$\xi_0 = s_1\ s_2\ s_1\ s_2\ s_1\ s_2\ s_1 \cdots$

$\xi_1 = s_1\ s_2\ s_3\ s_3\ s_3\ s_3\ s_3 \cdots$

$\xi_2 = s_1\ s_2\ s_1\ s_2\ s_3\ s_3\ s_3 \cdots$

以序列 $\xi_2$ 的第一个状态 $s_1$ 为例,有如下结论:

(1) $(\xi_2,1) \models \bigcirc e$。下一时刻运算符○在公式中被用来断言序列中的第二个状态,也就是 $s_2$ 满足 $e$。

(2) $(\xi_2,1) !\models \bigcirc\bigcirc e$。线性时序逻辑公式○○$e$ 表示"弹簧在下下个时刻会伸长"。然而 $s_1$ 的下下个状态还是 $s_1$,弹簧并未伸长。

(3) $(\xi_2,1) \models \Diamond e$。线性时序逻辑公式◇$e$ 表示"弹簧终将伸长",断言序列中存在某个状态满足 $e$。确实,序列中的第二个状态就满足 $e$。

(4) $(\xi_2,1) !\models \Box e$。线性时序逻辑公式□$e$ 表示"弹簧一直伸长",断言序列中的每个状态都满足 $e$。实际上序列中第三个状态就不满足 $e$。

(5) $(\xi_2,1) \models \Diamond\Box e$。线性时序逻辑公式◇□$e$ 表示"弹簧最终会一直伸长",断言序列中存在某个状态,其所有后续的状态都满足

$e$。确实，从序列的第四个状态开始就一直满足 $e$。

(6) $(\xi_2,1)!\models(\neg e)\cup m$。线性时序逻辑公式 $(\neg e)\cup m$ 表示“弹簧直到失去弹性才能伸长”。实际上对第二个状态来说，弹簧是伸长的，且并未失去弹性。

进一步，对于被测对象模型 $M$ 来说，如果 $M$ 满足某形式化规约 $\varphi$，意味着 $M$ 的任何状态序列中的任何状态都满足 $\varphi$。假设 $M$ 是上述弹簧模型，有如下结论：

(1) $M\models\Diamond e$。对 $M$ 来说，$\Diamond e$ 指任何状态序列都会到达弹簧伸长的状态。因为弹簧不会永远处在初始状态 $s_1$，所以它是成立的。注意，如果在 $s_1$ 上添加指向自身的状态迁移，这一结论就不再成立。

(2) $M\models\Box(\neg e\rightarrow\bigcirc e)$。对 $M$ 来说，$\Box(\neg e\rightarrow\bigcirc e)$ 意味着在任何状态序列的任何状态下，如果弹簧没有伸长，那么它一定会在下一状态伸长。它是成立的，在 $M$ 的任意状态序列中，每一次 $s_1$ 发生后都立刻有 $s_2$ 发生。

(3) $M!\models\Diamond\Box e$。对 $M$ 来说，$\Diamond\Box e$ 意味着任意状态序列最终将会达到弹簧永远保持伸长的状态。显然，在 $\xi_0$ 中这并不成立。

(4) $M!\models\Box(e\rightarrow\bigcirc\neg e)$。对 $M$ 来说，$\Box(e\rightarrow\bigcirc\neg e)$ 意味着在每个弹簧伸长的状态后，存在一个直接的后继状态满足 $\neg e$。虽然 $\xi_0$ 中该公式成立，但是 $\xi_1$、$\xi_2$ 中此公式均不成立，因为它们最终都一直停留在 $s_3$ 状态。

#### 3.3.2.3 分支时序逻辑

分支时序逻辑是在线性时序逻辑的基础上增加路径量词拓展而成的。将被测对象的状态序列视为一条路径，每一个状态就是这条路径上的节点。对被测对象模型 $M$ 来说，从初始状态 $s_0$ 出发可能产生多条路径分支。可以用路径量词 A 表示“对所有路径”，用路径量词 E 表示“存在一条路径”。结合路径量词，即可表述在多条路径分支上的状态转换逻辑期望，也就是基于分支时序逻辑的形式化规约。例如：

1. $(M,s_0)\models A\Box f\Leftrightarrow \forall\,\xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots,\forall\,i\geqslant 0,(\xi,i)\models f$

即从 $s_0$ 出发的所有路径上的所有状态节点都满足 $f$，如图 3-23 所示。

2. $(M,s_0)\models E\Box f\Leftrightarrow \exists\,\xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots,\forall\,i\geqslant 0,(\xi,i)\models f$

即存在一条从 $s_0$ 出发的路径，使得该路径上的所有节点都满足 $f$，如图 3-24 所示。

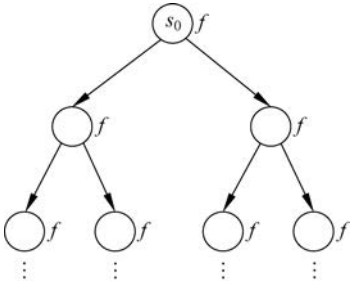


**图 3-23** $(M,s_0)\models A\Box f$

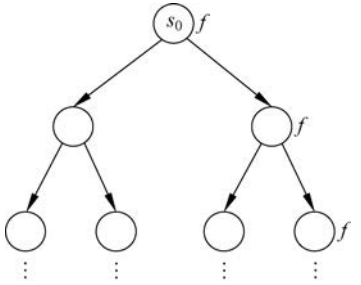


**图 3-24** $(M,s_0)\models E\Box f$

3. $(M,s_0)\models A\Diamond f\Leftrightarrow \forall\,\xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots,\exists\,i\geqslant 0,(\xi,i)\models f$

即从 $s_0$ 出发的所有路径最终都存在一个节点满足 $f$，如图 3-25 所示。

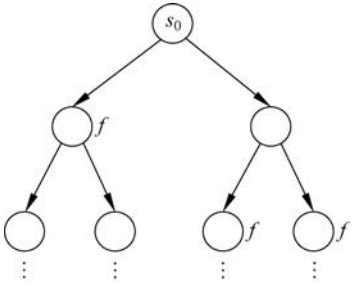


**图 3-25** $(M,s_0)\models A\Diamond f$

4. $(M,s_0)\models E\Diamond f \Leftrightarrow \exists\, \xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots, \exists\, i\geqslant 0, (\xi,i)\models f$

即存在一条从 $s_0$ 出发的路径，并且在该路径上最终存在一个节点满足 $f$，如图 3-26 所示。

5. $(M,s_0)\models A\bigcirc f \Leftrightarrow \forall\, \xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots, (\xi,1)\models f$

即从 $s_0$ 出发的所有路径上的第二个节点都满足 $f$，如图 3-27 所示。

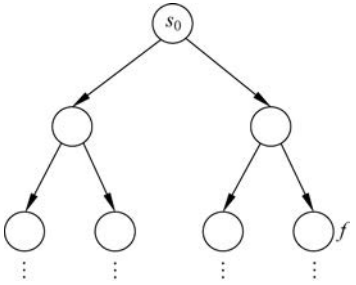


**图 3-26** $(M,s_0)\models E\Diamond f$

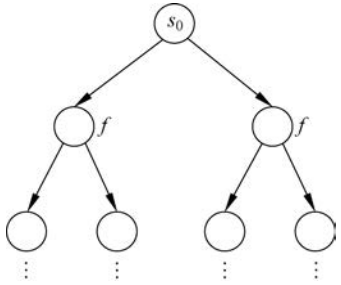


**图 3-27** $(M,s_0)\models A\bigcirc f$

6. $(M,s_0)\models E\bigcirc f \Leftrightarrow \exists\, \xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots, (\xi,1)\models f$

即存在从 $s_0$ 出发的一条路径，使得该路径上的第二个节点满足 $f$，如图 3-28 所示。

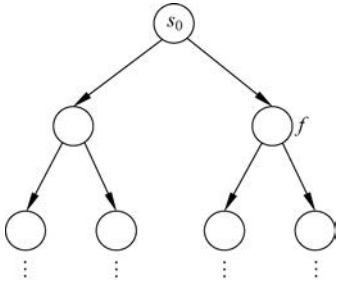


**图 3-28** $(M,s_0)\models E\bigcirc f$

7. $(M,s_0)\models A(f\cup g)\Leftrightarrow \forall \xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots,\exists k\geqslant 0,$
$(\xi,k)\models g\wedge \forall i:0\leqslant i<k,(\xi,i)\models f$

即从 $s_0$ 出发的所有路径上，都存在一个节点满足 $g$，并且在该节点之前所有的节点都满足 $f$，如图 3-29 所示。

8. $(M,s_0)\models E(f\cup g)\Leftrightarrow \exists \xi=s_0\rightarrow s_1\rightarrow s_2\rightarrow\cdots,\exists k\geqslant 0,$
$(\xi,k)\models g\wedge \forall i:0\leqslant i<k,(\xi,i)\models f$

即存在一条从 $s_0$ 出发的路径，该路径上存在一个节点满足 $g$，并且在该节点之前所有的节点都满足 $f$，如图 3-30 所示。

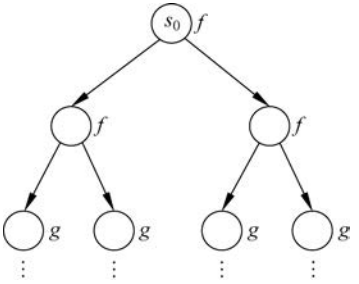


图 3-29 $(M,s_0)\models A(f\cup g)$

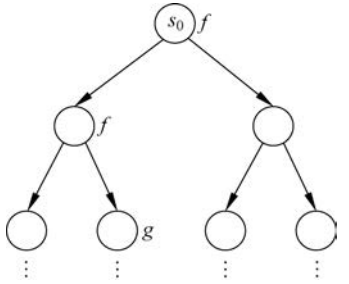


图 3-30 $(M,s_0)\models E(f\cup g)$

### 3.3.3 标记算法

标记算法是实现分支时序逻辑模型检验的一种典型方法。作为“理想”的形式化规约，有时可能包含相当复杂的逻辑，要判定某个状态是否符合这样的规约，难度比较大。针对这一问题，标记算法的基本思路是：将分支时序逻辑规约公式拆解为一层层的子公式，每一层子公式的符合性判定相对容易。由内而外对每一层子公式进行状态枚举判定，直到完成对最外层完整规约公式的判定。

具体来说，标记算法的主要过程为：对模型的每一个状态 $s$，建立标记集合 $L(s)$，以标记目前已知的、状态 $s$ 满足的所有子公式。

初始时$L(s)$只包含在状态$s$下成立的原子命题。然后将描述期望的分支时序逻辑规约公式进行拆解。例如对于公式$A\bigcirc(p\rightarrow A\Diamond q)$,可以拆解为以下四层:

(1) $p,q$。

(2) $A\Diamond q$。

(3) $p\rightarrow A\Diamond q$。

(4) $A\bigcirc(p\rightarrow A\Diamond q)$。

内层的子公式可以看成外层子公式的原子命题,由此便可将复杂的规约公式简化为一组简单的子公式。已知$\neg$、$\wedge$、$A\Diamond$、$E\bigcirc$、$EU$可以作为分支时序逻辑的一组完备算子集,其他的逻辑运算都可以转换为这一组算子的组合。因此可以将任何分支时序逻辑规约公式拆解为符合这一组算子的子公式。当然,分支时序逻辑的完备算子集并非只有这一组,标记算法也可以采用其他的完备算子集,如$\neg$、$\wedge$、$E\Box$、$E\bigcirc$、$EU$。

对拆解后的子公式从里向外逐层进行规约符合性判定。在每一层的判定过程中枚举所有状态,根据判定结果更新每个状态的标记集合$L(s)$,具体标记方法遵循如下规则:

(1) 对于公式$\neg f$,枚举所有状态,如果$f$在状态$s$不成立,则将$\neg f$加入$L(s)$中。

(2) 对于公式$f\wedge g$,枚举所有状态,如果$f$和$g$都在状态$s$中成立,则将$f\wedge g$加入$L(s)$中。

(3) 对于公式$A\Diamond f$,从路径树的叶节点出发,自底向上枚举所有状态,如果$f$在状态$s$成立,或者$A\Diamond f$在$s$的所有直接后继状态上都成立,则将$A\Diamond f$加入$L(s)$中。

(4) 对于公式$E\bigcirc f$,从路径树的根节点出发,自顶向下枚举所有状态,如果$f$在状态$s$的某个直接后继状态上成立,则将$E\bigcirc f$加入$L(s)$中。

(5) 对于公式$E(f\bigcup g)$,枚举所有状态,如果$g$在状态$s$成立,则将$E(f\bigcup g)$加入$L(s)$中。枚举完成后,再次从叶节点开始自底向上枚举所有状态,如果$f$在某个状态$s'$上成立,且$E(f\bigcup g)$在$s'$

的某个直接后继状态上成立，则将 $E(f \bigcup g)$ 加入 $L(s')$ 中。

完成了最外层完整规约公式的符合性判定后，就可以得到模型检验的结论了。如果模型的所有初始状态都满足规约，则称该模型满足规约，或称被测对象的模型是正确的。

下面举例说明模型检验标记算法的具体应用。假设被测对象的迁移系统模型如图 3-31 所示。

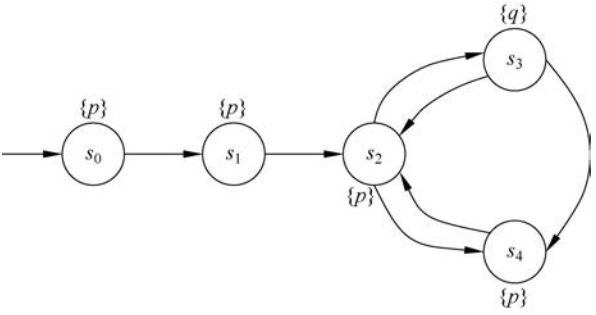


**图 3-31　某被测对象的迁移系统模型**

各状态满足的原子命题已经标注在状态节点附近。采用标记算法判定该模型是否满足规约 $A\Diamond(p \rightarrow A\Diamond q)$，实现过程如下：

(1) 以 $\neg$、$\wedge$、$A\Diamond$、$E\bigcirc$、$EU$ 为完备算子集，利用等值演算规律，对规约的分支时序逻辑公式进行重写：

$$\begin{aligned} A\Diamond(p \rightarrow A\Diamond q) &= A\Diamond(\neg p \vee A\Diamond q) \\ &= A\Diamond(\neg(p \wedge \neg(A\Diamond q))) \end{aligned}$$

(2) 进而将其拆解为一组子公式，其中 $g_7$ 即原始规约：

$$g_1 = p$$
$$g_2 = q$$
$$g_3 = A\Diamond g_2$$
$$g_4 = \neg g_3$$
$$g_5 = g_1 \wedge g_4$$
$$g_6 = \neg g_5$$
$$g_7 = A\Diamond g_6$$

(3) 根据 $g_1=p$ 进行标记：

$$L(s_0)=\{g_1\}$$
$$L(s_1)=\{g_1\}$$
$$L(s_2)=\{g_1\}$$
$$L(s_4)=\{g_1\}$$

(4) 根据 $g_2=q$ 进行标记：

$$L(s_3)=\{g_2\}$$

(5) 根据 $g_3=A\diamondsuit g_2$ 进行标记：

$$L(s_3)=\{g_2,g_3\}$$

(6) 根据 $g_4=\neg g_3$ 进行标记：

$$L(s_0)=\{g_1,g_4\}$$
$$L(s_1)=\{g_1,g_4\}$$
$$L(s_2)=\{g_1,g_4\}$$
$$L(s_4)=\{g_1,g_4\}$$

(7) 根据 $g_5=g_1\wedge g_4$ 进行标记：

$$L(s_0)=\{g_1,g_4,g_5\}$$
$$L(s_1)=\{g_1,g_4,g_5\}$$
$$L(s_2)=\{g_1,g_4,g_5\}$$
$$L(s_4)=\{g_1,g_4,g_5\}$$

(8) 根据 $g_6=\neg g_5$ 进行标记：

$$L(s_3)=\{g_2,g_3,g_6\}$$

(9) 根据 $g_7=A\diamondsuit g_6$ 进行标记：

$$L(s_3)=\{g_2,g_3,g_6,g_7\}$$

至此可得到所有状态对各层子公式的符合性判定结果，如表 3-9 所示。

**表 3-9 各层子公式的符合性判定结果**

| 子 公 式 | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|---|
| $g_1$ | √ | √ | √ | | √ |
| $g_2$ | | | | √ | |

续表

| 子　公　式 | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|---|
| $g_3$ | | | | √ | |
| $g_4$ | √ | √ | √ | | √ |
| $g_5$ | √ | √ | √ | | √ |
| $g_6$ | | | | √ | |
| $g_7$ | | | | √ | |

可见，模型仅在状态 $s_3$ 上满足原始规约 $g_7$，在初始状态 $s_0$ 上并不满足。最终结论是，被测对象的模型不满足规约。

## 3.4　本章小结

枚举是最简单直接的测试设计思想，主要用于缓解正确性判定问题。为了控制测试实施成本，基于枚举思想的测试设计方法都在“压缩”方面下足了功夫，并且为了不影响正确性的判定，枚举测试方法的压缩都是“无损压缩”。这一点不仅是此类方法的鲜明特征，也是我们在测试设计中贯彻枚举思想的基本要求。

组合测试主要用于判定被测对象在因素耦合处理方面的正确性。针对被测对象期望所关注的组合力度，一个测试用例有可能覆盖多个组合，这就为“无损压缩”创造了条件；另一种思路是，通过识别耦合关系的真空地带，排除测试用例中的冗余。

分割测试是工程实践中最常用的测试设计方法之一。如果能够分割出真正的同质子空间，分割测试中的压缩就是典型的“无损压缩”。然而同质子空间的识别需要充分综合来自现实和理想的信息，实现起来难度颇大。分割测试也常常用于检出缺陷。从效率出发，在分割子空间时应充分考虑潜在缺陷在测试输入空间中的分布情况。

当前可用的算力资源仍在快速增长，基于枚举思想的测试方法也会有越来越多的用武之地。为了借助计算机实现枚举测试，测试人员需要将被测对象的理想和现实都转化为计算机可理解的形式，

比如模型检验方法中使用的形式化模型和形式化规约。值得注意的是，这一转化过程中往往需要一定的近似和简化，有可能对枚举测试的质量评估结论产生影响。

## 本章参考文献


[1] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing[C]//27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings. IEEE, 2002: 91-95.

[2] 聂长海. 组合测试[M]. 北京：科学出版社, 2015.

[3] Weyuker E J, Ostrand T J. Theories of Program Testing and the Application of Revealing Subdomains[J]. IEEE Transactions on Software Engineering, 1980, 6(3): 236-246.

[4] Hamlet D, Taylor R. Partition testing does not inspire confidence[C]// Software Testing, Verification and Analysis, 1988. Proceedings of the Second Workshop on. IEEE, 1988.

[5] Weyuker E J, Jeng B. Analyzing partition testing strategies[J]. IEEE Transactions on Software Engineering, 1991, 17(7): 703-711.

[6] Leung H, Chen T Y. A new perspective of the proportional sampling strategy[J]. The Computer Journal, 1999, 42(8): 693-698.

[7] Chen T Y, Yu Y T. On the relationship between partition and random testing[J]. Software Engineering IEEE Transactions on, 1994, 20(12): 977-980.

[8] Chen T Y, Tse T H, Yu Y T. Proportional sampling strategy: A compendium and some insights[J]. Journal of Systems and Software, 2001, 58(1): 65-81.

[9] Duran J W, Ntafos S C. An evaluation of random testing[J]. IEEE Transactions on Software Engineering, 1984 (4): 438-444.

[10] Frankl P G, Weyuker E J. A formal analysis of the fault-detecting ability of testing methods[J]. IEEE Transactions on Software Engineering, 1993, 19(3): 202-213.

[11] Gaudel M C. Formal methods and testing: Hypotheses and correctness approximations[C]//International Symposium on Formal Methods. Springer, Berlin, Heidelberg, 2005: 2-8.

[12] 张广泉. 形式化方法导论[M]. 北京：清华大学出版社，2015.

[13] Wile B，Goss J C，Roesner W. 全面的功能验证：完整的工业流程[M]. 沈海华，乐翔，译. 北京：机械工业出版社，2010.

[14] 屈婉玲，耿素云，张立昂. 离散数学[M]. 2 版. 北京：清华大学出版社，2008.

[15] Ammann P E，Black P E，Majurski W. Using model checking to generate tests from specifications[C]//proceedings second international conference on formal engineering methods (Cat. No. 98EX241). IEEE，1998：46-54.