再一次强调，虽然上面讨论的启发式方法很可能会减少用因果图产生的测试用例的数量，但也可能会摒弃一些有用的测试用例。当然，在通常情况下或在开始执行测试之前，几乎不可能知道摒弃掉的测试用例哪些是有用的、哪些是真正没用的。

2.6.5 从判定表生成测试用例

从判定表生成测试用例就比较简单了。判定表的每一列至少生成一个测试用例。注意，当因果图中的一个条件可以用多个值满足时，一个组合就能生成多个测试用例。例如，考虑下面的原因：

$$C: x < 99$$

上面的条件 C 可以被多个值满足，如 $x = 1$，$x = 49$；同样，也有多个值不满足条件 C，如 $x = 100$，$x = 999$。这样，在从判定表的列生成测试用例时，测试人员可以选择输入变量的不同值。

虽然在确定输入变量值时有多种选择，只要它们满足判定表中的要求就行，但还是建议，所做的选择要使新生成的测试用例尽量与用别的方法（如边界值分析）生成的测试用例不一样。练习 2.25 要求用因果图方法设计出例 2.19 中基于 GUI 的计算机营销系统的一个测试集。

2.7 基于谓词的测试生成

本节将介绍测试集的一些生成技术，它们保证能检测出涉及规则编程的某些错误。作为测试集设计依据的那些规则，可能存在于软件需求里，也可能嵌入在被测软件中。

规则可以形式化地表示为谓词。例如，考虑软件需求“若打印机处于 ON 状态且具备打印纸，则发送要打印的文件”。这句话包含一个条件和一个动作。下面的谓词，记为 p_r，表示这句话的条件部分：

$$p_r: (\texttt{printer\_status=ON}) \wedge (\texttt{printer\_tray} = \neg \texttt{empty})$$

谓词 p_r 包含两个由布尔运算符“∧”连接的关系表达式。两个关系表达式都使用了等于符号（=）。编程人员可能正确地为 p_r 编码，也可能没有正确编码，导致程序中存在缺陷。我们的目标是，根据谓词产生测试用例，从而可以确保在测试中发现某种类型的所有缺陷。这种用于验证谓词实现是否正确的测试称之为*谓词测试*。

在介绍谓词测试方法之前，先定义几个与谓词和布尔表达式相关的术语；然后讨论谓词测试方法所能发现的故障类型；接着介绍谓词约束；最后是谓词测试用例生成算法。

2.7.1 谓词和布尔表达式

设 *relop* 表示集合 $\{<, >, \leqslant, \geqslant, =, \neq\}$ 中的一个关系运算符。设 *bop* 表示集合 $\{\wedge, \vee, \forall, \neg\}$ 中的一个布尔运算符，其中 $\wedge$，$\vee$，$\forall$ 是二元布尔运算符，$\neg$ 是一元布尔运算符。布尔变量的取值集合为 {true, false}，对于给定的布尔变量 a，$\neg a$ 和 $\bar{a}$ 都表示 a 的补。

*关系表达式*是指形如 e_1 *relop* e_2 的表达式，其中 e_1 和 e_2 取值为有限或无限集合 S。可将 S 中的元素进行排序，从而可以使用任意关系运算符对 e_2 和 e_1 进行比较。

一个条件可以表示成*简单谓词*或*复合谓词*。简单谓词就是一个布尔变量或关系表达式，其中变量可以取非。复合谓词可以是一简单谓词，或是由若干简单谓词或其补通过二元布尔运算符连接起来的一个表达式。谓词当中的圆括号表示布尔变量、关系表达式的组合。表 2-5 给出