

| 第一部分 |

The Craft of Model-Based Testing

基于模型测试的模型理论



第1章 |

The Craft of Model-Based Testing

基于模型测试概述

无论是软件还是硬件，甚至是日常生活中，所有的测试都可以视为系统因某个激励产生响应，然后对其进行检查的过程。事实上，早期的需求方法也主要是针对激励 – 响应进行研究的。在基于模型的测试（MBT）中，我们认为模型在某种程度上就是激励 – 响应的一种表达方式。其中有一些说明性的词汇有助于我们对这个问题的理解。

在软件测试领域，有3种被广泛接受的测试级别：单元测试、集成测试和系统测试。每个级别的测试都有明确的测试目标和测试方法。单元测试主要针对类或者过程进行测试，集成测试主要针对单元之间的交互进行测试，而系统测试则更加关注被测系统（SUT）的对外端口（也称为端口边界）。每个级别的测试都有一个测试覆盖矩阵（更详细的讨论请见[Jorgensen 2013]），同时每个级别的测试用例都包含相似的要素：用例名称和标识（ID）、前置条件、输入序列（也可能是交互输入）及预期输出、记录的实际输出、后置条件，以及通过准则。

1.1 基本术语

定义：被测系统通常缩写为 SUT，即将要被测试的系统。

SUT 可能是：一个由软件控制的硬件系统，一个仅有硬件的系统或者仅有软件的系统，甚至可能是由若干个 SUT 组成的大系统。SUT 也可以是一个单独的软件单元或者是由多个单元组成的集合。

定义：被测系统的端口边界，是被测系统中所有能够施加输入激励以及接收输出响应的端口集合。

无论是硬件、软件还是固件抑或是3种的组合，每一类系统都有端口边界。识别出 SUT “真正的” 端口边界对于基于模型的测试（MBT）过程而言是至关重要的。为什么说是“真正的”？因为在测试过程中我们很容易将用户引起的物理事件与由此产生的电子信号（激励）相混淆。在基于 Web 的应用中，用户接口很可能就是系统级的输入和输出所在位置。在汽车刮水器控制器中，端口边界通常包括控制杆、决定刮水器速度的拨盘以及驱动刮水器叶片的电机。本书中还用到了车库门控系统（后面会详述），该系统的端口边界包括发送控制信号的设备、安全设备、末端传感器，以及一个驱动电机。一个单元的端口边界则是激活该单元的某种机制（可能是面向对象软件中的消息，也可能是传统软件中的某个过程调用）。

定义：端口输入事件是针对给定 SUT 端口边界的一个激励。同样，端口输出事件是发生在 SUT 端口边界的一个输出响应。

在看待系统的端口输入事件上，开发人员和测试人员在观念上有明显的不同。测试人员的想法是针对 SUT 如何产生或引发一个输入激励，而开发人员考虑更多的是输入激励会导致软件产生什么样的行为和动作。这种区别也同样明显地反映在输出事件上，测试人员需要知道如何观察或者探知到系统的输出响应，而开发人员则关心如何生成或者引发输出响应。这些观念上的不同，有一部分是由软件开发团队构建的设计和开发模型所造成的。设计和开

发模型是从开发者的视角进行构建的，而测试用例的产生却是从测试人员的视角构建的。这个不同之处可能会在基于模型的测试中产生一定的影响。

1.2 事件

下面的术语基本上都是同义词，但略有不同：端口输入事件、激励、输入。同样，下面这些词也基本上算是同义词：端口输出事件、响应、输出。事件是一级一级发生的，当然，或许使用“按顺序发生”会更好些。我们来看车库门控系统（见1.8.2节详述），重点是包含光束传感器的安全设备。当车库门正在向下关闭的时候，如果任何事物阻断了光束（在靠近地板部分），那么电机会立刻停止并反向打开车库门。这个事件序列就是从某一个物理事件开始的，这个事件有可能是向下关门过程中有一个小动物恰巧穿过了光束。当光束传感器探测到中断时，它会给控制器发送一个信号。这就是端口输入事件，而且是一个真实的电信号。内部的控制器软件就会将此视为一个逻辑事件。

端口输入事件可能发生在不同的逻辑场景中。“一只猫穿过了光束”是一个物理事件，而这个物理事件有可能发生在好几种场景中，例如车库门已经打开的时候，正在打开车库门的时候，或者正在关闭车库门的时候。我们所关心的逻辑事件却只是发生在“正在关门”的时候。通常，事件发生环境在某些有限状态机（FSM）中表示为状态。在评估不同类型的模型对MBT的支持能力的时候，有一个很重要的指标就是该类型的模型能否表达和识别出对环境敏感的输入事件。同样，这也要求我们关注端口输入设备本身。假设一个测试人员想要测试光束传感器，尤其是其失效模式。通常的设备失效包括“在位置1失效”（SA-1）和“在位置0失效”（SA-0）。对于SA-1失效，光束传感器在发送信号位置失效，即保持一直发送信号的状态，无论物理输入事件有没有发生。请注意，在这种失效情况下是没法关上车库门的（请见下表中的EECU-SA-1用例）。而SA-0失效更加隐蔽一些，它表示光束传感器始终保持不发送信号的状态，这将导致即使物理中断发生之后，门也不会反向打开。我相信，如果律师得知在安全设备中会产生SA-0失效之后，那么他一定会非常郁闷。在第8章我们会对此进行建模。

用例名称	光束传感器在位置1失效	
用例ID	EEUC-SA-1	
描述	用户尝试使用设备控制信号关闭一个已经打开的门。光束传感器发生SA-1失效	
前置条件	1. 车库门开 2. 光束传感器发生SA-1失效	
事件序列		
输入事件	输出事件	
1. 设备控制信号	2. 起动电机向下	
3. 光束SA-1失效	4. 停止并反向运转电机	
5. 达到上行轨道终点	6. 停止电机	
后置条件	1. 车库门打开 2. 光束传感器产生SA-1失效	

“在某个位置失效”这类故障以及其他失效模式是很难被定义的。它们可能不会出现在需求文档中。就算是在需求文档中有所说明，在很多基于模型测试的过程中也很难对其进行

建模。我们会在第 8 章详述这一点。用户有可能会提供类似 EEUC-SA-1 的用例吗？基于以往的经验，这是很有可能的。但在敏捷开发中，这就是个挑战了。

1.3 测试用例

对于一个测试用例而言，它有两种基本的形式——抽象的和真实的，有些 MBT 团队将后者称为具体测试用例。抽象测试用例通常是从一个形式化模型中派生出来的。何为抽象？意指输入通常是以变量方式来表达的。真实的（具体的）测试用例包含输入变量的真实数值，以及预期的输出数值。这两种形式都包括前置和后置条件。

1.4 测试用例的执行框架

图 1-1 是一个通用的自动化测试用例执行框架。它的基础是我的团队在 20 世纪 80 年代早期开发的一个针对电话交换机系统的回归测试项目。图中的计算机包括所有测试用例的处理器，这些处理器控制并观察测试用例的执行过程。测试用例使用简单的语言来描述，这个语言可以解释执行。语言中包括 CAUSE 和 VERIFY 语句，它们指代 SUT 的端口边界。CAUSE 语句一般都带有参数，它们指代端口输入事件和发生这些事件的设备（它们还可能有附加参数）。同样，VERIFY 语句指代预期的端口输出事件。在电话交换机的 SUT 中，一个测试用例可能有以下两种语句：

```
CAUSE InputDigit(9) on Line12  
VERIFY DigitEcho(9) on Line12
```

在这些语句中，InputDigit 指代 Line12 设备上发生的带有参数的端口输入事件以及该设备上发生的端口输出事件。能够实现这个框架的关键是开发一个能够连接 SUT 与测试用例处理器的“套件”。套件主要完成的工作包括以下内容。在 CAUSE 语句中，将端口输入事件从逻辑形式（抽象形式的用例）转换为物理形式；在 VERIFY 语句中，将端口输出事件从物理形式转换为逻辑形式。

以上内容主要针对系统级测试。在单元测试中，类似 nUnit family 的自动测试工具很常见，其中 CAUSE 和 VERIFY 语句被 ASSERT 语句替代。ASSERT 语句中包括被测单元的输入和输出，由此取代了系统测试中的测试套件。本书中我们忽略了集成测试，因为几乎没有 MBT 工具能够支持集成测试。本章结尾部分给出的 MBT 例子也只是包括了单元测试和系统测试。大家一定要记住，基于模型的测试要想成功，使用的模型必须能够提供激励和响应，不管它是单元级别还是系统级别。

1.5 MBT 中的模型

软件和系统设计模型通常包含两种类型——针对结构的模型和针对行为的模型。在通用建模语言（UML）中（UML 已经成为一种事实上的标准），针对结构的模型集中于类、类的属性、方法和类之间的连接（继承、聚合、相关）。另外，主要有两种针对行为的模型——状态图和活动（或者顺序）图。本书第一部分呈现了 9 种针对行为的模型：流程图、决策表、

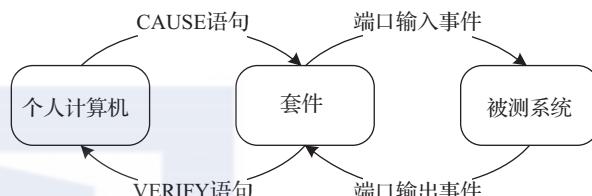


图 1-1 通用的测试执行框架

有限状态机、Petri网、事件驱动的Petri网、状态图、泳道型事件驱动的Petri网、UML（用例和活动图）、业务流程建模和标识（BPMN）。在这些模型中，除了泳道型事件驱动的Petri网[Jorgensen 2015]和BPMN以外，都在[Jorgensen 2008]中有非常详细的解释。本书的重点是扩展这些模型，以支持基于模型的测试。在MBT中有一个不可避免的局限性，只有原生模型（软件和系统设计模型）比较好，派生出来的测试用例才能够同样好。因此在第一部分我们需要特别关注的就是不同模型的局限性以及它的表达能力。

1.6 ISTQB中的MBT扩展

本书意在与ISTQB基础级大纲中关于MBT的内容保持一致，该大纲于2015年10月发布。ISTQB（国际软件测试评定委员会）是一个非营利组织，截止到2013年，已经有超过100个国家的336 000名测试人员取得认证。本书的第二部分介绍了6个测试工具，以及使用这些工具对书中的例子进行测试的结果。

1.7 MBT的形式

基于模型测试的过程有3种基本形式：手动测试、半自动测试和全自动测试[Utting 2010]。在手动MBT中，构建并分析SUT的模型是为了设计测试用例。例如，在基于有限状态机的MBT过程中，首先需要使用有限状态机对SUT进行建模，这样一条从初始状态到最终状态的路径就可以被形象地标识出来并转化成测试用例。接着需要确定一个可用准则，用来选择哪些测试用例将予以执行。这些选择测试用例的准则实质上就是某些覆盖矩阵。然后我们还需要对这些用抽象术语描述的测试用例进行“具体化”（这是一个在MBT领域常用的术语）。比如，将“个人ID号码”这种抽象表达具体化为一个真实的数值“1234”。最后一步是在被测系统上执行具体的测试用例。这部分内容可以参考Craig Larman[Larman 2001]，资料中提到了用例的4个级别（第9章会详述）。其中第三级称为扩展的关键用例，包含了抽象的变量名；而Larman的第四级用例就是真实使用的用例，将抽象的术语和变量名称替换为要测试的真实数值。这就是具体化过程。半自动MBT和手动MBT的区别是在测试过程的早期是否使用了工具。工具是指能够运行某个合适模型并且生成抽象测试用例的引擎。下一步，从生成的测试用例集中挑选予以执行的测试用例，这个过程可以自动进行也可以手动进行。在有限状态机例子中，挑选测试用例的准则可能是以下几种：

- 1) 覆盖所有状态。
- 2) 覆盖所有变迁。
- 3) 覆盖所有路径。

这个挑选过程也可以自动进行。另外，全自动MBT与半自动MBT的区别是测试用例是否自动执行，我们在第二部分会详述这一过程。

1.8 案例集

1.8.1 单元级问题：保费计算

政策规定的汽车保费是根据考虑了成本的基本利率计算而成的。该计算的输入如下所示：

6 第一部分 基于模型测试的模型理论

- 1) 基本利率是 600 美元。
- 2) 保险持有者的年龄 ($16 \leq \text{年龄} < 25$; $25 \leq \text{年龄} < 65$; $65 \leq \text{年龄} < 90$)。
- 3) 小于 16 岁或者大于 90 岁的，不予保险。
- 4) 过去 5 年中出险次数 (0、1 ~ 3 和 3 ~ 10)。
- 5) 过去 5 年出险次数超过 10 次的，不予保险。
- 6) 好学生减免 50 美元。
- 7) 非饮酒者减免 75 美元。

具体数值的计算见表 1-1 ~ 表 1-3。

表 1-1 不同年龄段的保费系数

年龄区间	年龄系数
$16 \leq \text{年龄} < 25$	$x = 1.5$
$25 \leq \text{年龄} < 65$	$x = 1.0$
$65 \leq \text{年龄} < 90$	$x = 1.2$

表 1-2 出险次数中的惩罚金额

过去 5 年的出险次数	惩罚金额
0	0 美元
1 ~ 3	100 美元
4 ~ 10	300 美元

表 1-3 针对好学生和非饮酒者减免费用的决策表

c1: 好学生	T	T	F	F
c2: 非饮酒者	T	F	T	F
a1: 减免 50 美元	X	X	—	—
a2: 减免 75 美元	X	—	X	—
a3: 什么都不做	—	—	—	—

1.8.2 系统级问题：车库门控系统

车库门控系统包括驱动电机、能够感知开 / 关状态的车库门轮传感器以及控制设备。另外，还有两个安全设备：地板附近的光束传感器和障碍物传感器。只有车库门在关闭过程中，后面两个安全设备才会运行。正在关门的时候，如果光束被打断（可能家中的宠物穿过）或者车库门碰到了一个障碍，那么门会立即停止动作，然后向反方向运行。为减少后续章节中的模型数目，本书中我们只考虑光束传感器。关于障碍物传感器的分析，与之基本相同，不再赘述。一旦门处于运行状态（要么正在打开，要么正在关闭），控制设备发出信号，门就会停止运行。后续的控制信号会根据门停下来时的运动方向来起动门的运行。最后，有些传感器会检测到门已经运行到了某个极限位置，即要么是全开，要么是全关。一旦这种情况发生，门会停止动作。图 1-2 是车库门控系统的 SysML 上下文图。

在绝大多数的车库门控系统中，有如下几个控制设备：安装在门外的数字键盘、车库内独立供电的按钮、车内的信号设备。为简单起见，我们将这些冗余的信号源合并为一个设备。同样，既然两个安全设备产生同样的响应，我们只考虑光束设备，忽略障碍物传感器。

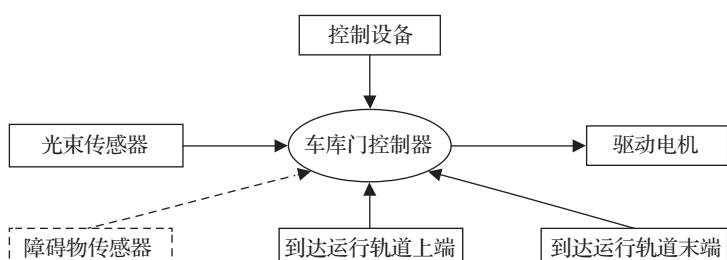


图 1-2 针对车库门控系统的 SysML 图

1.8.3 其他案例

这里给出其他几个案例以说明和比较建模理论和技术。表 1-4 描述了表 1-5 中的示例使用的不同的建模方式。

表 1-4 表 1-5 中示例的建模选择

	流程图	决策表	有限状态机	Petri 网	事件驱动的 Petri 网	状态图		
不错的选择	WCF	ND	EVM	EVM	EVM	EVM	WCF	风寒指数计算
	IP	WW	RRX	RRX	RRX	RRX	IP	保费计算
				WW	WW	WW	ND	日期计算
				PCP	PCP	PCP	EVM	咖啡自动售卖机
			GDC	GDC	GDC	GDC	RRX	铁路道口门控制器
可以工作，但是……	ND	IP	WW				WW	刮水器控制器
	EVM	EVM					GDC	车库门控系统
	RRX	RRX					PCP	生产者 - 消费者问题
	WW	GDC						
糟糕的选择	PCP	WCF	WCF	WCF	WCF	WCF		
	GDC	PCP	IP	IP	IP	IP		
			ND	ND	ND	ND		
			PCP					

表 1-5 第一部分各章使用的补充示例

章	模型	章节特定的补充示例
2	流程图	咖啡自动售卖机 (EVM), 日期计算 (ND), 风寒指数计算 (WCF)
3	决策表	日期计算 (ND)
4	有限状态机	铁路道口门控制器 (RRX), 刮水器控制器 (WW)
5	Petri 网	生产者 - 消费者问题 (PCP)
6	事件驱动的 Petri 网	刮水器控制器 (WW)
7	状态图	刮水器控制器 (WW)
8	泳道型事件驱动的 Petri 网	
9	面向对象的模型 (统一建模语言)	

1.9 MBT 的技术现状

从 20 世纪 80 年代起，MBT 就以手动的方式开始为人所用。此后许多学术研究团队开发的开源 MBT 工具，让更多的人开始关注 MBT。最近，商用 MBT 工具的出现使得这项技术进入到工业领域。从 Robert V. Binder 所做的两份调查中可以看出使用 MBT 的一些原因。最早针对 MBT 用户的调查是在 2011 年开展的，随后在 2014 年又做过一次。这些调查总结了早期 MBT 使用者的期待与顾虑。

Binder 在 2011 年的调查 [Binder 2012] 中特别强调了以下几点：

- MBT 的应用有了长足发展，包括软件过程、应用领域和开发团队方面。
- MBT 不仅可用而且可行。一半的被访者反馈说，只需要 80 小时或者更少的时间就可以基本熟练掌握 MBT 技术，80% 的受访者需要最多 100 小时。
- 平均来说，受访者反馈 MBT 技术能够减少 59% 的错误逃逸率。
- 平均来说，受访者反馈 MBT 技术能够减少 17% 的测试开销。
- 平均来说，受访者反馈 MBT 技术能够减少 25% 的测试时间。

该调查项目在 2014 年再次开展，同时增加了两个 MBT 的实践者：Anne Kramer（见第 18 章）和 Bruno Legeard（见第 14 章）[Binder 2014]。此次一共收到 100 份反馈。下述内容显示了此次调查的亮点（内容为直接引用或者部分引用）。

测试级别：

- 77.4% 的人使用 MBT 进行系统测试
- 49.5% 的人使用 MBT 进行集成测试
- 40.9% 的人使用 MBT 进行验收测试
- 31.2% 的人使用 MBT 进行部件测试

生成的产品：

- 84.2% 的产品是自动的测试脚本
- 56.6% 的产品是手动测试用例
- 39.5% 的产品是测试数据
- 28.9% 的产品是其他文档

最大的益处：

- 测试覆盖率
- 处理复杂度
- 自动测试用例的生成
- 重用模型和模型元素

最大的局限性：

- 工具支持
- 需要专门的 MBT 技能
- 不愿意改变

通用观察：

- 96% 的人在功能测试中使用 MBT
- 81% 的人使用图形化模型
- 59% 的人针对行为特性建模

- 大概需要 80 小时成为熟练用户
- 72% 的参与者非常愿意继续使用 MBT

MBT 用户的期待：

- 73.4% 的人期待更高效的测试设计
- 86.2% 的人期待更有效的测试用例
- 73.4% 的人期待更好地管理系统测试的复杂性
- 44.7% 的人期待改进交流
- 59.6% 的人期待尽早开始测试设计

MBT 的整体效率：

- 23.6% 非常有效
- 40.3% 中等有效
- 23.6% 稍微有效
- 5.6% 几乎无效
- 1.4% 效率轻微下降
- 2.8% 效率中等下降
- 2.8% 效率极度下降

参考文献

[Binder 2012]

Binder, Robert V., Real Users of Model-Based Testing, blog, <http://robertvbinder.com/real-users-of-model-based-testing/>, January 16, 2012.

[Binder 2014]

Binder, Robert V., Anne Kramer, and Bruno Legeard, *2014 Model-Based Testing User Survey: Results*, 2014.

[Jorgensen 2008]

Jorgensen, Paul C., *Modeling Software Behavior—A Craftsman's Approach*. CRC Press, Boca Raton, FL, 2008.

[Jorgensen 2013]

Jorgensen, Paul C., *Software Testing—A Craftsman's Approach*, 4th ed. CRC Press, Boca Raton, FL, 2013.

[Jorgensen 2015]

Jorgensen, Paul C., A Visual Formalism for Interacting Systems. In Petrenko, Schlingloff, Pakulin (Eds.): *Tenth Workshop on Model-Based Testing (MBT-2015), Proceedings MBT 2015*, arXiv:1504.01928, EPTCS 180, 2015, pp. 41–55. DOI:10.4204/EPTCS.180.3.

[Larman 2001]

Larman, Craig, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, 2nd ed. Prentice-Hall, Upper Saddle River, NJ, 2001.

[Uutting 2010]

Uutting, Mark, Pretschner, Alexander, and Legeard, Bruno, A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 2012;22:297–312. Published online in Wiley InterScience (www.interscience.wiley.com). DOI:10.1002/stvr.456.

第 2 章 |

The Craft of Model-Based Testing

流程图

计算机领域很早就开始使用流程图 (flowchart) 了, 这可能是使用最早的一类行为模型。在 20 世纪 60 年代, 工具供应商通常会提供一些具有可塑性的流程图模板, 程序员可以据此绘制出更整齐的流程图。IBM 公司甚至提供了一个带有基本标记的不同规格标准的流程图模板。前辈们开玩笑说, 这是该领域的第一个 CASE (计算机辅助软件工程) 工具。

2.1 定义与表示法

通常有两种不同风格的流程图标记形式。图 2-1 所示为第一种, 这是一种极简风格的表示方式, 只有表示动作、决策和两种页连接符的符号。分页连接符一般用于不能在一页上显示完全的大型系统中。传统的分页连接符使用大写字母 ABC…, 第一个分页连接符是 A, 与之对应的连接点也用 A 表示。

在流程图使用的早期, 因为比较关心系统的输入 / 输出设备, 所以开发出很多扩展的流程图符号, 如图 2-2 所示, 这里面甚至有表示卡片式 I/O 的符号。

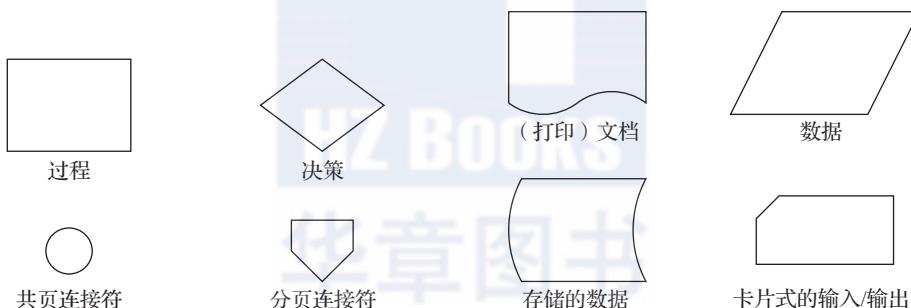


图 2-1 极简风格流程图符号

图 2-2 I/O 设备流程图符号

在这种流程图标记方式中, “流”的部分使用带箭头的线段, 表示从某个流程图符号开始, 在某个流程图符号处结束。图 2-3 是特浓咖啡自动售卖机的流程图示例, 利用这个咖啡机, 可以用 1 欧元买到一小杯意大利特浓咖啡。后续我们还会使用这个案例讨论流程图技术。

2.2 技术详解

图 2-3 中的流程图能够处理自动售卖机的销售过程。接收欧元硬币, 给出特浓咖啡。如果仔细观察流程图, 可能会注意到, 所有的箭头都会在符号的顶部结束。这不是强制要求, 但这样做有助于理解。决策框只能有一个入口, 既然是决策过程, 那么至少应有两个出口箭头。在早先的 Fortran 年代, “Arithmetic IF” 语句带有 3 个输出。这个钻石形状的框 (决策符号框) 的边与底部顶点可以分别表示 3 种选择 (<、=、>)。在图 2-3 中很容易看到这 3 个

选项。如果多于 3 个选项（比如 Case/Switch 语句），每个选项对应一个连接箭头即可。（标识符是为使用者服务的，不必非要遵守条条框框）。过程框则最多只有一个输出箭头。但它们可能有多个输入箭头。箭头上的标签通常表明决策框有几个可能的出口，Yes/No、True/False 或者数值显示决策结果。从过程框发出的箭头则没有标签。这样表示过程框的过程是完整的，流程走向下一个框。框之间不会表达信息或内容。使用者需要自己确定过程框的结果对于后续框是否可用（流程图具有“记忆性”）。

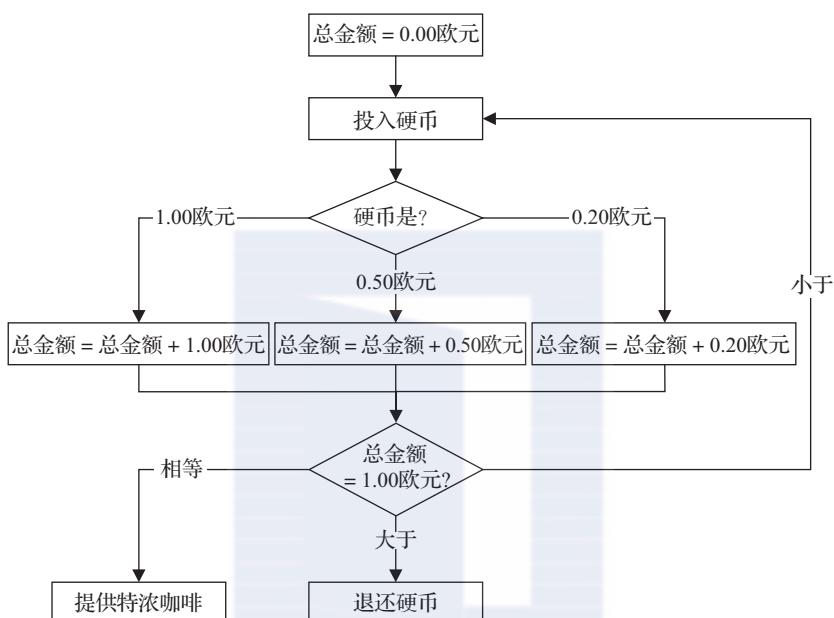


图 2-3 特浓咖啡自动售卖机流程图

对于过程框和决策框来说，框内文本内容几乎没有限制，可以有不同的形式，这些形式的抽象程度也可以不同。框内文本可以简明扼要，如图 2-3 所示。或者也可以非常具体，甚至可直接是某种编程语言。例如“硬币是”决策框的输出结果可以是 1.00 欧元、0.50 欧元或 0.20 欧元三种形式。同样，决策条件也可以表示为针对每一种硬币类型的二进制条件，如“是”或“否”。不管使用哪种方式，最好要保持一致，如果不同抽象级别的文本表达混合在一起，那将是很让人困惑的。流程图的符号集可以支持结构化编程的 3 种基本结构：顺序、选择、循环。例如图 2-3 所示的决策框表示了“选择”结构，而图 2-3 底部还有一处表示了“循环”结构：如果变量“总金额”与 1 欧元的比较结果是“小于”，就会返回到投入硬币过程框。再看一下 1 欧元那个分支，它结束“提供特浓咖啡”这个过程框，这正是“顺序”结构的例子。在结构化编程中，通常有“单一入口，单一出口”的设计惯例，但是这个惯例并不是强制要求，图 2-3 所示例子是符合这个惯例的。

流程图采用分层策略来处理不同抽象级别的过程。也就是说，一个高抽象级别的过程框可以扩展成一系列更详细、单独的流程图。如果这样做，那么每一个低抽象级别的流程图都应该被命名，以此来清晰地表明它是从某个高抽象级别的过程框扩展而来的，或者干脆使用分页连接符（只是分页连接符的方式显得有些笨重）。本章结束部分的表 2-4，总结了在流程图中可以表达的控制事件。

2.3 案例分析

2.3.1 日期计算函数

NextDate 函数是测试圈里非常流行的一个例子，它非常简单，很容易找到测试用例的预期输出。(给定某个日期，NextDate 函数返回下一个日期。) 图 2-4 和图 2-5 采用分层流程图方式将功能进行了分解。变量 lastDay 的数值通过其他方式计算出来，然后与决策框中的日期进行比较。如果某个变量可以在多处赋值，那么必须保证最终计算出来的数值在运行过程中不会相互改写。在 NextDate 流程图中，有 3 个值被分配给 nextDay、nextMonth 和 nextYear。决策框要求这 3 个赋值必须是互斥的。在这些计算过程中使用到了 GoTo 语句，当使用流程图来建立主要的行为模型时，使用 GoTo 语句是一种常用作法。在流程图中可以清晰地看到，NextDate 函数的逻辑是很严密的。

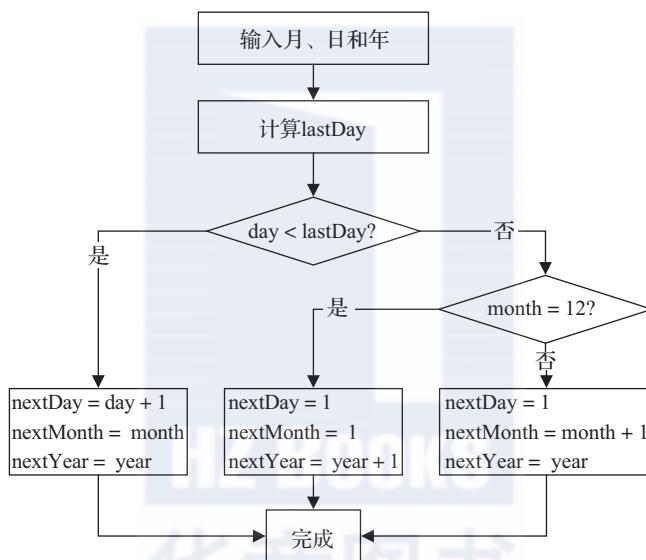


图 2-4 NextDate 函数

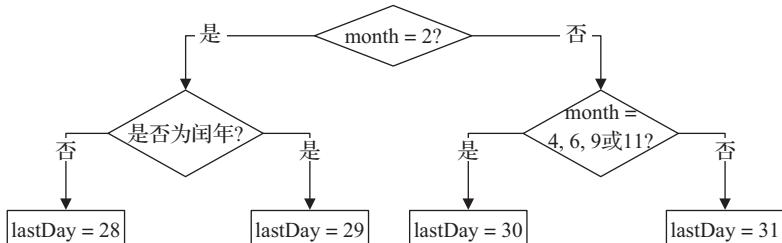


图 2-5 lastDay 计算的详细信息

2.3.2 风寒指数表

著名的风寒指数（密歇根州或者其他寒冷地区）是由两个变量构成的函数：每小时风速 V 和摄氏温度 T 。其公式如下所示：

$$W = 35.74 + 0.6215 \times T - 35.75 \times (V^{0.16}) + 0.4275 \times T \times (V^{0.16})$$

其中, W 是人脸的表面温度, 以华氏度为单位; T 是空气温度, 以华氏度为单位, $-20 \leq T \leq 50$; V 是风速, 以 mile/h ($1\text{mile} = 1609.34\text{m}$) 为单位, $3 \leq V \leq 73$ 。

基于图 2-6 所示的流程图可以完成一个类似表 2-1 的表格, 请注意其中的循环嵌套。

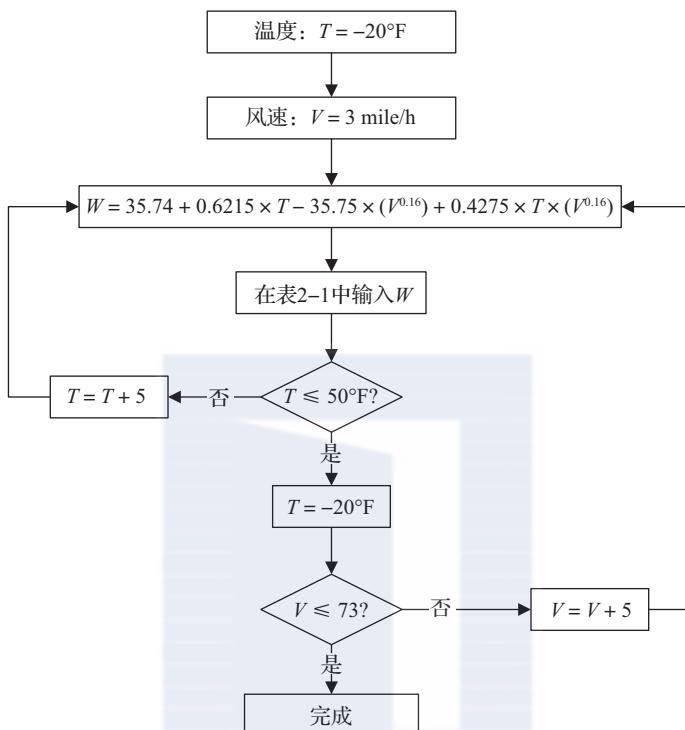


图 2-6 风寒指数计算流程图

表 2-1 以风速和空气温度表示的风寒指数函数

温度 / 风速	-20 °F	-15 °F	...	0 °F	...	45 °F	50 °F
3 mile/h							
8 mile/h							
...							
70 mile/h							
73 mile/h							

(例子中的数值范围是随意制定的, 在密歇根州, 这是真实情况!) 风寒表格中温度的范围是 $-20^\circ\text{F} \leq T \leq 50^\circ\text{F}$, 每次递增 5°F ; 风速范围是 $3 \leq V \leq 73$, 每次递增 5mile/h 。

2.3.3 保费计算流程图

图 2-7 是 1.8.1 节中定义的保费计算问题的流程图模型。

2.3.4 车库门控系统流程图

图 2-8 是 1.8.2 节中定义问题的流程图模型。

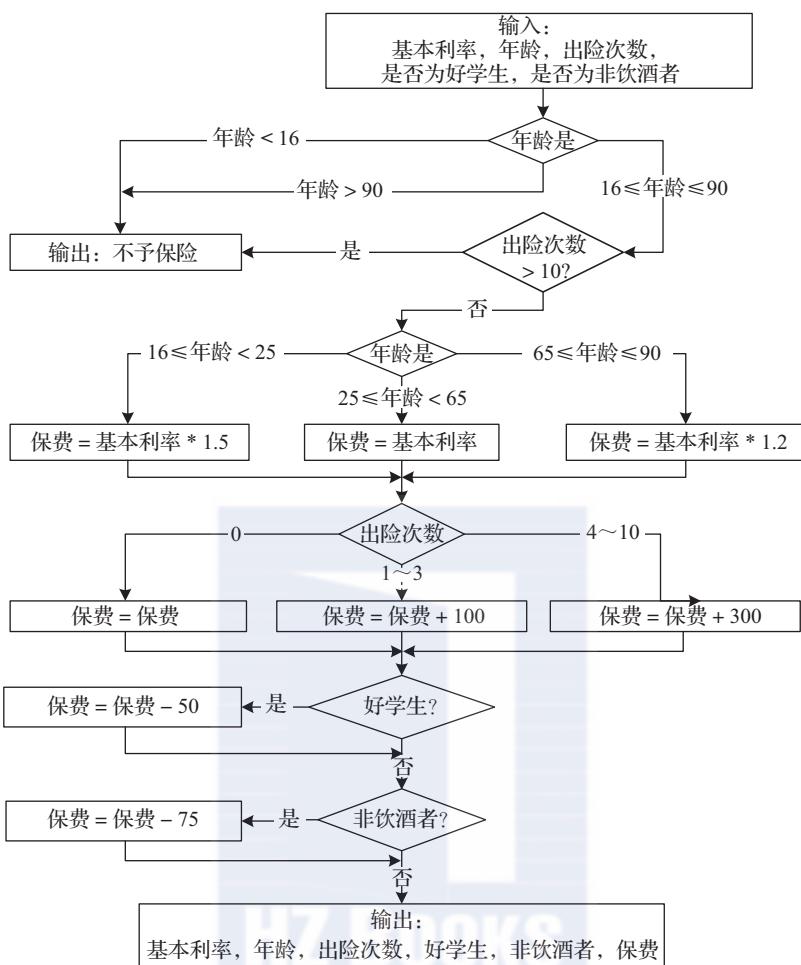


图 2-7 保费计算问题的流程图

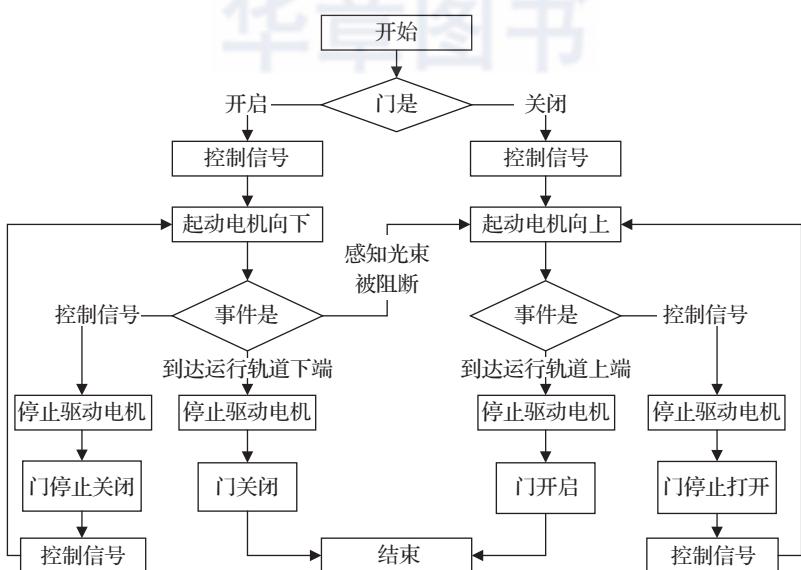


图 2-8 车库门控系统的流程图

2.4 基于流程图派生的测试用例

流程图中的路径可以直接推导出抽象的测试用例。由于流程图可以显示并发的路径，因此很容易手工设计出相关的抽象测试用例。从图 2-7 所示的保费计算流程图中可以看到关于年龄和保费变量的并发路径。毫无疑问，并发路径应该是互斥的。（测试用例也可以利用某个指定流程图的语义内容派生出来，但这反过来又要求手动推导测试用例生成。）通常来说，从流程图中不可能直接设计出具体的测试用例，因为流程图只显示了需要完成的过程，并不实际执行这些过程。

（对于类似 Fortran 这样的程序来说，流程图很好用。）而对于事件驱动型应用来说，流程图的形式就显得不太适合了。从接下来给出的事件驱动示例，可以清晰看出以下几个问题。

- 事件可以表示为决策的输出或者过程（过程框）。
- 由于没有专门的事件标识符，所以事件只能表示为流程图中有明确语义的内容。
- 需要具有洞察力和领域经验才能识别与上下文相关的输入事件。

2.4.1 保费计算问题的测试用例

图 2-7 所示的流程图中一共有 40 条不同的路径，其中 36 条对应着表 2-2 列出的使用等价类测试方法生成的测试用例。年龄和出险次数变量由变量的取值范围来定义，因此从表面上看，只需要测试某种形式的边界值就足够充分了。将每个等价类中的边界值映射到相同的乘法和加法函数集合中，它们会在结果中产生大量的冗余，因而这并无太大价值。图 2-9 显示了表 2-2 中测试用例 1 的路径。

表 2-2 显示了保费计算问题的抽象测试用例和具体测试用例。抽象测试用例可以从流程图中直接设计出来（MBT 工具可以完成这个过程）。实际数值则需要从需求中获取。从图 2-7 中给定的文本可以看出，对于 MBT 工具来说，直接生成测试用例需要的实际数值还是很困难的。我主要使用电子表格中的替换功能来完成这部分工作，这样还不算太麻烦。通常来说，电子表格是 MBT 工具很顺手的一个补充。

下面是图 2-8 所示路径对应的测试用例 1 中的抽象和具体测试用例。

保费计算抽象测试用例 1	
前置条件	基本费用、年龄、出险次数、是否为好学生、是否为非饮酒者都是已知的
输入	过程
1. 年龄, $16 \leqslant \text{年龄} < 25$	2. 基本利率乘以 1.5
3. 出险次数 = 0	4. 基本利率乘以 1
5. 是好学生	6. 从基本费用中减去 50 美元
7. 非饮酒者	8. 从基本费用中减去 75 美元
后置条件	基本费用包含保费

保费计算具体测试用例 1	
前置条件	基本费用 = 600 美元
输入	过程（输出）
1. 年龄 = 20	2. $900 = 600 \times 1.5$
3. 出险次数 = 0	4. $900 = 900 - 0$
5. 是好学生	6. $850 = 900 - 50$
7. 非饮酒者	8. $775 = 850 - 75$
后置条件	保费 = 775 美元

表 2-2 保费计算问题的抽象和具体测试用例

测试用例	输入数据				数值(基本利率)				
	年龄	出险次数	好学生	非饮酒者	年龄系数	出险次数(美元)	好学生(美元)	非饮酒者(美元)	保费(美元)
1	20	0	是	是	1.5	0	-50	-75	775
2	20	0	是	否	1.5	0	-50	0	850
3	20	0	否	是	1.5	0	0	-75	825
4	20	0	否	否	1.5	0	0	0	900
5	20	2	是	是	1.5	100	-50	-75	875
6	20	2	是	否	1.5	100	-50	0	950
7	20	2	否	是	1.5	100	0	-75	925
8	20	2	否	否	1.5	100	0	0	1000
9	20	6	是	是	1.5	300	-50	-75	1075
10	20	6	是	否	1.5	300	-50	0	1150
11	20	6	否	是	1.5	300	0	-75	1125
12	20	6	否	否	1.5	300	0	0	1200
13	45	0	是	是	1	0	-50	-75	475
14	45	0	是	否	1	0	-50	0	550
15	45	0	否	是	1	0	0	-75	525
16	45	0	否	否	1	0	0	0	600
17	45	2	是	是	1	100	-50	-75	575
18	45	2	是	否	1	100	-50	0	650
19	45	2	否	是	1	100	0	-75	625
20	45	2	否	否	1	100	0	0	700
21	45	6	是	是	1	300	-50	-75	775
22	45	6	是	否	1	300	-50	0	850
23	45	6	否	是	1	300	0	-75	825
24	45	6	否	否	1	300	0	0	900
25	75	0	是	是	1.2	0	-50	-75	595
26	75	0	是	否	1.2	0	-50	0	670
27	75	0	否	是	1.2	0	0	-75	645
28	75	0	否	否	1.2	0	0	0	720
29	75	2	是	是	1.2	100	-50	-75	695
30	75	2	是	否	1.2	100	-50	0	770
31	75	2	否	是	1.2	100	0	-75	745
32	75	2	否	否	1.2	100	0	0	820
33	75	6	是	是	1.2	300	-50	-75	895
34	75	6	是	否	1.2	300	-50	0	970
35	75	6	否	是	1.2	300	0	-75	945
36	75	6	否	否	1.2	300	0	0	1020
37	15 或 91	<10	(任何)	(任何)	(不允许)				(无保费)
38	20	11	(任何)	(任何)	(不允许)	11			(无保费)
39	45	11	(任何)	(任何)	(不允许)	11			(无保费)
40	75	11	(任何)	(任何)	(不允许)	11			(无保费)

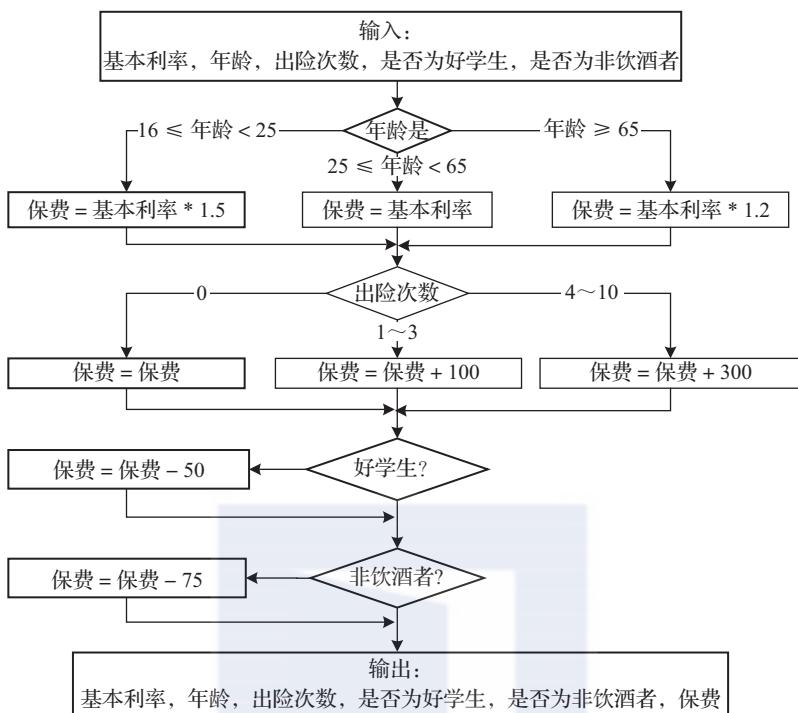


图 2-9 表 2-2 中测试用例 1 的路径

2.4.2 车库门控系统的测试用例

在车库门控系统的流程图中有两个循环：一个是停下并重启正在关闭的门，另一个是停下并重启正在打开的门。（一个是在关门过程中停下或者重启，一个是在开门过程中停下或者重启。）如果假设输入事件和输出过程是同时发生的，那么数学家会说，在图 2-8 所示的流程图中，存在一个含有不同路径的可数无穷路径集合。在实际生活中，我家的车库门需要大概 13s 关闭或者开启，而停止 / 重启序列需要大概 1s，因此，实际上车库门流程图只有一个有限数目的可能路径。图 2-10 显示了其中一条路径，该路径由下面的测试用例来表示。图 2-10 所示的流程图符号是有编号的，以表示路径的追踪和命名。

流程图符号序列为 1、2、3、4、5、12、13、14、15、20 的车库门测试用例（见图 2-10）	
前置条件	车库门开启
输入事件	输出过程
1. 发出控制信号	2. 起动电机向下
3. 传感光束被阻断	4. 起动电机向上
5. 运行到轨道上端	6. 停止电机
后置条件	车库门开启

表 2-3 包含车库门流程图中的 5 条不同路径。我们先简要描述一下每条路径（也可以视之为用户场景），然后再使用按系列编号的流程图符号来详细描述它们。5 条路径涵盖了每个流程图符号和流程图中的每条边。另外还有些路径是针对光束被打断之后的过程的，这些路径也包括每个上下文的控制设备的输入事件。对于工具来说，很难从流程图中设计出详细的、类似使用用例（use case，从用户角度考虑的使用场景）的测试用例。

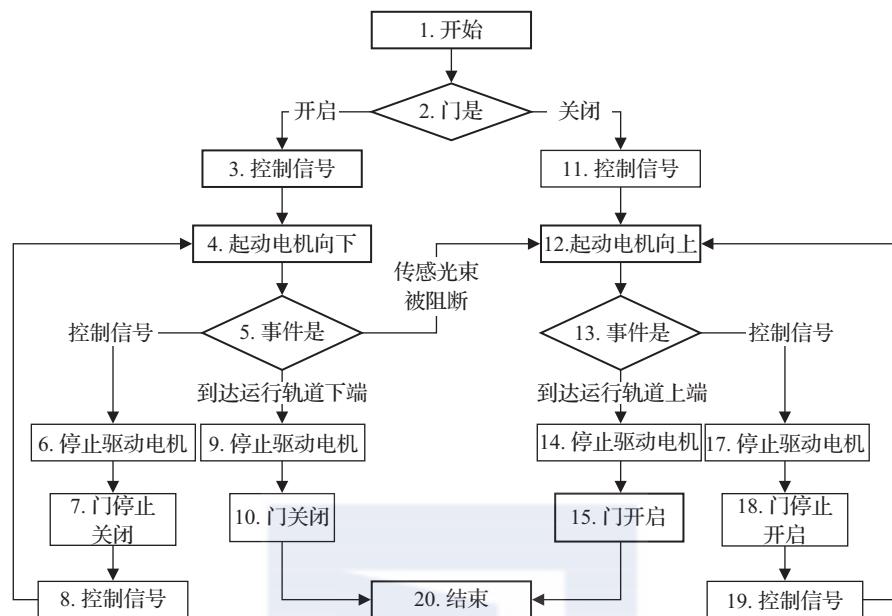


图 2-10 车库门流程图的一条路径

表 2-3 车库门流程图中的示例路径

车库门流程图中的路径		
路径	描述	流程图符号序列
1	车库门正常关闭	1、2、3、4、5、9、10、20
2	门中间停止一次后正常关闭	1、2、3、4、5、6、7、8、4、5、9、10、20
3	门正常开启	1、2、11、12、13、14、15、20
4	门中间停止一次后正常开启	1、2、11、12、13、17、18、19、12、13、14、15、20
5	门正在关闭时，光束传感器被阻断	1、2、3、4、5、12、13、14、15、20

表 2-4 包括从图 2-8 所示流程图中设计出的手工测试用例信息。从中我们可以看出，流程图的定义存在一些“小问题”，原因如下：

- 1) 输入事件看上去既可以是决策框的输出，也可以是过程框中表示的一个过程；
 - 2) 事件上下文（状态）看上去是与过程框对应的过程，也可以说是决策框的输出；
 - 3) 有两个潜在的死循环：正在开门或者关门时停止。
- 除非具有领域经验，否则，利用输入事件 / 输出过程对是没有办法定位下一步操作的。

2.5 优势与局限

流程图有很多优势。如果任何一种表达方式已经使用了数十年，那么它肯定还是有可取之处的。对于流程图来说，这个优势就很容易理解。由于过程框和决策框里面的文本可以使用自然语言，因此流程图使客户和开发者之间具有更好的交互性。就连美国的 IRS 都是用流程图来解释复杂的税务代码的。如我们之前所说，流程图能够表达基本的结构化编程架构。其中，还有没有明确说明的地方，这就是对于流程图中“存储器”功能的使用。如果一个变量在过程框中被赋予一个值，那么该变量在后续需要保持这个数值。在图 2-3 中可以清楚地看到，变量“总金额”被定义之后，在之后的循环中可以再次定义它。任何“形式化好的”流程图都可以

使用命令式和结构化的编程语言进行编码。如示例中所示，流程图支持几种级别的抽象，因此它们具有可扩展性，能够描述大型而且复杂的应用。另一个优势是，它们可以用来描述复杂的计算和算法。最后一个优势是有些控制过程或者行为可以通过流程图中不同的路径来表达。

表 2-4 车库门测试用例信息

门的上下文（状态）	输入事件	输出过程	下一个上下文（状态）
开启	发出设备控制信号	起动电机向下	正在关闭
正在关闭	运行到轨道下端	停止电机	已关闭
正在关闭	发出设备控制信号	停止电机	停止关闭
正在关闭	光束被阻断	停止电机向下并反转	正在开启
停止关闭	发出设备控制信号	起动电机向下	正在关闭
已关闭	发出设备控制信号	起动电机向上	正在开启
正在开启	发出设备控制信号	停止电机	停止开启
正在开启	运行到轨道上端	停止电机	开启
停止开启	发出设备控制信号	起动电机向上	正在开启

流程图也有些限制。由于流程图的本质是将过程序列化，所以很难表达事件驱动的系统，因为在事件驱动的系统里面，独立事件可能以任何顺序发生。此外，流程图很难描述被描述系统需要操作的外部设备的上下文。虽然面向设备的 I/O 符号可以完成这个工作，但是需要很多扩展。流程图几乎没有办法表达数据，除非是在过程框或者 I/O 框里面。过程框和决策框里面的文本可以包含变量名，但这是很粗浅的表达。数据表达都如此困难，表达数据结构以及数据之间的关系就更难了。同样，对于描述事件，它也有很多潜在的困难。从图 2-8 中可以看到，设备控制信号有时表示为过程框，有时又显示为决策框的输出。表 2-5 将流程图的描述能力与第 1 章定义的标准列表进行了对比。

表 2-5 用流程图表示行为事件

事件	表现	建议
顺序事件	好	流程图的要点
选择事件	好	流程图的要点
循环事件	好	流程图的要点
可用事件	不好	当文本在过程框中时，必须描述
不可用事件	不好	当文本在过程框中时，必须描述
触发事件	不好	当文本在过程框中时，必须描述
激活事件	不好	当文本在过程框中时，必须描述
挂起事件	不好	当文本在过程框中时，必须描述
恢复事件	不好	当文本在过程框中时，必须描述
暂停事件	不好	当文本在过程框中时，必须描述
冲突事件	不好	
优先级事件	不好	当文本在过程框中时，必须描述
互斥事件	不好	决策后的并行路径
同步执行事件	不好	当文本在过程框中时，必须描述
死锁事件	不好	
上下文敏感事件	不好	必须仔细检查决策后的序列来推断
多原因输出事件	不直接	必须仔细检查决策后的序列来推断

(续)

事件	表现	建议
异步事件	不好	
事件静默	不好	
存在记忆?	好	参照先前的决策、输入、过程框
分层?	好	过程框根据需要扩展更详细的内容

2.6 经验教训

在 20 世纪 60 年代晚期，电话交换机系统开发实验室需要将所有的交互系统源代码的流程图文档提供给运营公司。当时，源代码大概是 30 万行的汇编程序。在这个过程中，软件工程师需要向图案部提交手绘流程图，六周以后，软件工程师就可以得到非常完美的流程图。在这六周内，如果设计师想要对流程图进行修改，那么他们可以将原始草图替换成修改之后的草图，以免在图案部里重新排队。

同时，我的管理团队中的一位数学家参加了一次研讨会，会上他看到一个程序，这个程序能够在 CalComp 绘图机上画出离散部件电路图。我们研究了技术资料后，决定将电路图符号替换成流程图符号，结果就产生了 AELFlow 系统 [Jorgensen 和 Papendick 1970]。这是我们学到的第一个经验：为了使绘图机能够得到认可，我们向每个部门展示了他们如何从绘图机上获益。推销了几周之后，我们最终得到许可购买了最小的绘图机。6 个月之后，我们有了最大的绘图机。AELflow 系统非常有效，不仅节约了返工的时间，而且提高了设计文档的整体可用性。相比汇编代码，流程图更容易从技术角度进行描述。从各方面来看，AELflow 系统都是真正的 CASE 工具，这比术语 CASE 的使用要早得多。

这里关键的教训是：变革是很难引入的，它需要时间、耐心和对企业的认知，同时也需要培训。尽管对于 AELFlow 系统来说，培训是很少的一部分。Gartner Hype 的周期是相当精确的，尽管持续的间隔可能有变化（如图 2-11 所示）。准备引入 MBT 的组织一定要经历宣传周期。我的观点是，期望膨胀的峰值期源于 MBT 产品的销售团队，泡沫破裂的幻灭期则源于不同模式的不充分的培训和教育。稳步爬升的复苏期开始于合适的工具和良好的模式教育，实质生产的成熟期则随着市场的关注度和占有率而保持。

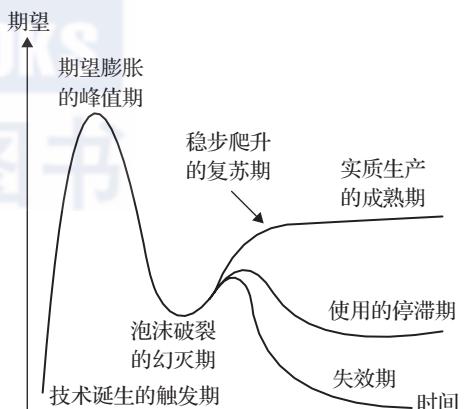


图 2-11 Gartner 技术成熟度曲线

参考文献

- [Jorgensen and Papendick 1970]
Jorgensen, Paul C. and David L. Papendick, AELFLOW—An automated drafting system.
Automatic Electric Technical Journal 1970;12(4):172–180.

第3章

The Craft of Model-Based Testing

决策表

几十年来，人们一直使用决策表技术表达和分析复杂的逻辑关系。决策表具有严格的形式化，善于分析事件的完整性与判断事件的冗余性和一致性。此外，它还支持特定情况下的编译 [CODASYL 1978]。决策表非常适合分析不同条件集情况下的行为组合。决策表也可以提供一个框架，以指导客户和开发者清晰地描述需求。表 3-1 描述了一些基本的决策表术语。

3.1 定义与表示法

决策表分为 4 个部分：粗竖线的左侧是桩，右侧是入口，粗横线的上面是条件，下面是行为。因此，我们可以将其分别称为条件桩、条件入口、行为桩、行为入口。入口部分的每一列是一条规则。如果某个行为与某个规则相对应，那就是说，在那种情况下应该采取这种行为。在表 3-1 所示的决策表中，如果条件 c1、c2、c3 都为真，那么行为 a1、a2 就都发生（规则 1）。如果 c1 和 c2 都为真而 c3 为假，那么行为 a1 和 a3 发生（规则 2）。

表 3-1 决策表示例

桩	规则 1	规则 2	规则 3	规则 4	规则 5	规则 6	规则 7	规则 8
c1	T	T	T	T	F	F	F	F
c2	T	T	F	F	T	T	F	F
c3	T	F	T	F	T	F	T	F
a1	X	X			X			
a2	X							
a3		X				X	X	X
a4			X	X	X		X	X

如果使用二进制条件（真 / 假、是 / 否、0/1），则决策表的条件部分就是一个旋转 90° 的根据命题逻辑得到的真值表。这种结构确保能够考虑到每个条件值的可能组合。在对软件的描述过程中，决策表并不是一种必不可少的表达形式，它只是一种说明性的表达。因为决策表中的条件并没有指定特殊的顺序，所以选中的行为也不会以特定的顺序发生，而且规则也可以用任何顺序来编写。

如果使用表 3-1 所示形式描述软件行为，我们通常会将正常情况放在前几条规则中来表示，而将异常情况放在后几条规则中来表示，这样会使决策表比较易懂。

定义：如果决策表中所有的条件都是二进制格式的，那么我们称之为有限入口决策表，表示为 LEDT。

带有 n 个条件的有限入口决策表，它带有 2^n 个不同的规则。

定义：如果某个决策表中所有的条件都具有可选的有限数值 (>2)，那么我们称之为扩展入口决策表，表示为 EEDT。

定义：如果决策表中有部分条件带有可选的有限数值，其他都是严格的二进制格式，我们称之为混合入口决策表，表示为 MEDT。

决策表假设所有用于评估条件的数值，对于表格的规则执行都是可用的。决策表中的行为可以改变变量值，这可能使决策表发生“循环表格”的行为。这就带来一个层次调用的问题：决策表中的行为可能会引用其他决策表。

3.2 技术详解

决策表严格的结构能支持某些代数运算。

3.2.1 决策表的精简

如果两个或多个规则有相同的行为入口，那么必有某个条件能在一个规则里面为真，而在另一个规则中为假。很明显，该条件在这些行为中没有起到任何作用，而这个行为会在其他规则中执行。因此表 3-2 可以将规则 3 和规则 4 精简，同理，也精简了规则 7 和规则 8。

表 3-2 精简决策表

桩	规则 1	规则 2	规则 3 和 4	规则 5	规则 6	规则 7 和 8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X					
a3		X			X	X
a4			X	X		X
规则数目	1	1	2	1	1	2

定义：如果某个条件对于由两个规则执行的行为没有影响，那么该条件的规则入口就是一个无关入口，通常使用长横线（—）标识。

规则 3 和规则 4 中，c3 的入口就是一个无关入口。无关入口有两个主要的解释：该条件是无关的，或者该条件不能实施。有时候人们对后一种情况，使用“n/a”符号标识该条件。由于在表 3-1 中，同样的行为发生在规则 3 和规则 4 中，条件 c3 对于行为集就没有影响，所以其被无关入口的长横线所取代。同理，请见规则 7 和规则 8。

借助布尔代数的知识可以更加理论化地解释这个问题，在一个良好形式化的决策表中规则应该是互斥的，所以第一个精简可以简单地理解为如下过程。

$$\text{规则 3: } (c1 \wedge (\neg c2) \wedge c3) \rightarrow a4$$

$$\text{规则 4: } (c1 \wedge (\neg c2) \wedge (\neg c3)) \rightarrow a4$$

所以两条规则的互斥就是: $((c1 \wedge (\neg c2) \wedge c3) \rightarrow a4) \oplus ((c1 \wedge (\neg c2) \wedge (\neg c3)) \rightarrow a4)$ 。

这样可以得出:

$$((c1 \wedge (\neg c2)) \wedge (c3 \oplus (\neg c3))) \rightarrow a4$$

由于 $(c3 \oplus (\neg c3))$ 永远为真，所以 $(c1 \wedge (\neg c2)) \rightarrow a4$ 。

3.2.2 有互斥条件的决策表

如果条件与等价类相对应，那么决策表就具有了比较明显的特性。表 3-3 所示决策表中

的条件是部分日历问题，它们对应的是月份变量的互斥等价类。因为它们是互斥的，所以不可能在某个规则中看到两个入口都为真。此处仍然使用无关入口长横线（—）。在这种情况下，它实际的意思是“一定是错的”。有些资深的决策表使用者会使用 F! 来强调这一点。

表 3-3 带有互斥规则和规则数目的决策表

条件	规则 1	规则 2	规则 3
c1: 30 天的月份?	T	—	—
c2: 31 天的月份?	—	T	—
c3: 二月?	—	—	T
a1			
规则数目	4	4	4

使用无关入口有个小问题，那就是如何确定完整的决策表。对于有限入口决策表来说，如果存在 n 个条件，就必须存在 2^n 个不同的规则。如果无关入口的确表明某个条件是无关紧要的，那么我们可以通过如下方法计算规则的数目：不包含无关入口的规则视为一条规则，规则里面只要有无关入口，就将无关入口的数目乘以 2 加入该规则的数目。表 3-2 中最下面一行就是压缩规则的数目，规则数目之和为 8 (2^3)。表 3-3 中决策表的规则数目也显示在表格的最底下一行。注意，其规则数之和是 12 (理应如此)。

如果我们将这种简单的算法用在表 3-3 的决策表中，我们就可以获得表 3-4 所示的规则数目。但是，我们应该只有 8 (2^3) 条规则，所以肯定什么地方出了问题。为了找到问题所在，展开这三条规则，将无关入口替换成本表 3-5 所示的 T 和 F。

表 3-4 表 3-3 中带有规则数目的扩展决策表

条件	规则 1	规则 2	规则 3	规则 4	规则 5	规则 6	规则 7	规则 8	规则 9	规则 10	规则 11	规则 12
c1: 30 天的月份?	T	T	T	T	T	T	F	F	T	T	F	F
c2: 31 天的月份?	T	T	F	F	T	T	T	T	T	F	T	F
c3: 二月?	T	F	T	F	T	F	T	F	T	T	T	T
a1												
规则数目	1	1	1	1	1	1	1	1	1	1	1	1

注意，在规则 1、5、9 里面，所有入口都是 T，在规则 2、3、6、7、10 和 11 中，其中两个条件都是真。如果将这些不可能的规则都删除，我们就只剩下 3 条规则。这个过程的结果如表 3-6 所示，其中删除了不可能的规则，使用 F! (必定为假) 来强调互斥关系。

表 3-5 表 3-3 中带有不可能规则的扩展决策表

条件	规则 1	规则 2	规则 3	规则 4	规则 5	规则 6	规则 7	规则 8	规则 9	规则 10	规则 11	规则 12
c1: 30 天的月份?	T	T	T	T	T	T	F	F	T	T	F	F
c2: 31 天月的份?	T	T	F	F	T	T	T	T	T	F	T	F
c3: 二月?	T	F	T	F	T	F	T	F	T	T	T	T
a1												
不可能?	Y	Y	Y	N	Y	Y	Y	N	Y	Y	Y	N

表 3-6 删除不可能规则

条件	规则 4	规则 8	规则 12
c1: 30 天的月份?	T	F!	F!
c2: 31 天的月份?	F!	T	F!
c3: 二月?	F!	F!	T
a1			

3.2.3 冗余和不一致的决策表

在识别、设计并开发一个完整的决策表过程中，需要对它的冗余和不一致性进行充分的分析。表 3-7 所示的决策表就是冗余的，即 3 个条件对应 9 条规则（规则 9 与规则 4 是完全一致的）。为什么会这样呢？这很可能就是由一个能力不足的设计者所实现的决策表。

表 3-7 冗余决策表

桩	规则 1 ~ 4	规则 5	规则 6	规则 7	规则 8	规则 9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X

注意，规则 9 的行为入口与规则 1 ~ 4 的是一致的。只要冗余规则里面的行为与决策表中对应部分是一致的，就没什么大问题。如果行为入口不同，如表 3-8 所示，我们就有大问题了。

如果在表 3-8 所示的决策表中，我们要处理一件事务，其中 c1 为真、c2 和 c3 都为假，那么规则 4 和规则 9 都适用，我们可以得出以下两个结论：

- 1) 规则 4 和规则 9 是不一致的。
- 2) 决策表不具有确定性。

表 3-8 不一致的决策表

桩	规则 1 ~ 4	规则 5	规则 6	规则 7	规则 8	规则 9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

规则 4 和规则 9 是不一致的，因为行为集不同。整个表格是不确定的，因为没法判断应该使用规则 4 还是规则 9。

3.2.4 决策表引擎

尽管决策表是一种陈述性的表达方式，但我们还是希望尽量将其结构严格化，以支持

决策表引擎的执行。决策表执行引擎的输入应该是一个能够完成某个规则条件入口的所有信息，而输出动作应该是该规则对应的行为。但是这里有个问题，我们很难找到一个简单的方法来控制输出行为的顺序。不过可以使用整数入口来表示与某个规则对应的行为（而不是简单的字母形式 Xs，例如 a1）来表示输出动作的执行顺序。相应的，决策表引擎也可以是交互式的，用户可以针对决策表中的每一个规则提供一组数值表示。

3.3 案例分析

3.3.1 日期计算函数

因为 `NextDate` 函数的输入变量之间有很有趣的逻辑关系，所以它在软件测试领域非常有名。`NextDate` 是一个带有 3 个变量的函数：年、月、日。它在执行的时候，以年月日的方式返回下一天的日期。所有数值都有边界，是正整数，年的边界是随机的。

1 ≤ 月 ≤ 12

1 ≤ \square ≤ 31

1801 ≤ 年 ≤ 2100

将 NextDate 制作成一个决策表，我们需要使用精心选择的等价类作为条件，请参考 [Jorgensen 2009] 来获取更完整的讨论。

$M1 = \{月 : 月有 30 天\}$

$M2 = \{月: 月有 31 天除了十二月\}$

M3 = { 月 : 月是十二月 }

$$M4 = \{ \text{月} : \text{月是二月} \}$$

$$D1 = \{ \text{日} : 1 \leq \text{日} \leq 27 \}$$

$$D2 = \{ \text{日} : \text{日} = 28 \}$$

$$D3 = \{ \text{日} : \text{日} = 29 \}$$

$$D4 = \{ \text{日} : \text{日} \equiv 30 \}$$

$$D5 = \{ \text{日} : \text{日} \equiv 31 \}$$

$\text{Y1} = \{\text{年}, \text{年是闰年}\}$

$\text{Y2} = \{\text{年}, \text{年是平年}\}$

由于这些类的笛卡儿乘积包括 40 个元素，因此，我们需要考虑一个带有 40 个规则的决策表。如表 3-9 和表 3-10 所示，其中很多规则可以被压缩。

表 3-9 全 NextDate 决策表（第一部分）

表 3-9 全 NextDate 决策表 (第二部分)

条件	11	12	13	14	15	16	17	18	19	20	21	22
c1: 月在?	M3	M3	M3	M3	M3	M4						
c2: 日在?	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: 闰年?	—	—	—	—	—	—	Y	N	Y	N	—	—
a1: 不可能?	—	—	—	—	—	—	—	—	—	X	X	X
a2: 增加日	X	X	X	X		X	X	—	—	—	—	—
a3: 重置日	—	—	—	—	X	—	—	X	X	—	—	—
a4: 增加月	—	—	—	—	—	—	—	X	X	—	—	—
a5: 重置月	—	—	—	—	X	—	—	—	—	—	—	—
a6: 增加年	—	—	—	—	X	—	—	—	—	—	—	—

表 3-10 压缩后的 NextDate 决策表

条件	1-3	4	5	6-9	10	11-14	15	16	17	18	19	20	21、22
c1: 月在?	M1	M1	M1	M2	M2	M3	M3	M4	M4	M4	M4	M4	M4
c2: 日在?	D1, D2, D3	D4	D5	D1, D2, D3, D4	D5	D1, D2, D3, D4	D5	D1	D2	D2	D3	D3	D4, D5
c3: 闰年?	—	—	—	—	—	—	—	—	Y	N	Y	N	—
a1: 不可能?	—	—	X	—	—	—	—	—	—	—	X	X	—
a2: 增加日	X	—	—	X		X		X	X	—	—	—	—
a3: 重置日	—	X	—	—	X	—	X	—	—	X	X	—	—
a4: 增加月	—	X	—	—	X	—	—	—	—	X	X	—	—
a5: 重置月	—	—	—	—	—	—	X	—	—	—	—	—	—
a6: 增加年	—	—	—	—	—	—	X	—	—	—	—	—	—

3.3.2 汽车刮水器控制器

刮水器是由一个末端带有拨盘的控制杆控制的。控制杆有 4 个位置：关 (OFF)、Int (间歇)、低 (LOW) 和高 (HIGH)。拨盘有 3 个位置，分别为 1、2、3。拨盘的位置表明 3 种速度。只有当控制杆位于 Int 的时候，拨盘的位置才起作用。下表所示为相对于控制杆和拨盘位置的刮水器速度 (次 /min)。

c1: 控制杆	OFF	Int	Int	Int	LOW	HIGH
c2: 拨盘	n/a	1	2	3	n/a	n/a
a1: 次 /min	0	6	12	20	30	60

如上所述，我们几乎可以将其视为一个扩展入口决策表，只有一点区别，就是表 3-11 所示的刮水器速度都是独立的行为 (而不是条件有多个数值的情况)。

条件 c1 和 c2 只能表明拨盘和控制杆的状态，并不能显示控制杆和拨盘的动作事件，因此这个决策表不能表示出每个状态的前置状态，也就是无法表示某种状态对前置状态的敏感

情况。一般来说，如果控制杆在 Int 位置，其对应的规则就可以执行到拨盘位置，至少，拨盘位置此时是有意义的。第 10 章会更加详细地分析这个模型。

表 3-11 刮水器控制器的决策表

c1: 控制杆位置	关闭	间歇			低	高
c2: 拨盘位置	—	1	2	3	—	—
a1: 0 次 /min	X	—	—	—	—	—
a2: 6 次 /min	—	X	—	—	—	—
a3: 12 次 /min	—	—	X	—	—	—
a4: 20 次 /min	—	—	—	X	—	—
a5: 30 次 /min	—	—	—	—	X	—
a6: 60 次 /min	—	—	—	—	—	X

3.3.3 铁路道口门控制器

在伊利诺伊州北部，有一个铁路道口，杨树大道和芝加哥西北铁路在此交叉。在这个道口有 3 个不同的岔路，每个岔路都有传感器以感知火车接近道口或者离开道口。如果没有火车在道口或者接近道口，道口的门就是打开的。第一列火车过来的时候，门开始降低，当最后一列火车离开的时候，门开始抬升。如果已经有一列火车在道口内，第二列或者第三列火车已到达，那么门不执行动作，因为门此时已经是放下的状态。

端口输入事件	端口输出事件	数据
p1: 火车到达	p3: 降低道口门	d1: 道口内火车的数量
p2: 火车离开	p4: 抬升道口门	

表 3-12 是关于混合入口决策表 (MEDT) 极好的例子。条件 c1 类似内存，可以通过行为 a4 和 a5 予以更新。“不可能规则”在行为 a6 的入口处显示。规则 1 和规则 3 都不可能，因为如果道口没有火车，那么怎么可能有一个离开的呢？规则 13 和规则 14 也是不可能的，因为 3 个车道都已经被占用了。如果 c2 和 c3 都为真（规则 5 和规则 9），那么“什么也不做”行为就可以被递增和递减的行为取代。但是“什么也不做”这个行为显示了这些输出如何互相抵消。同样，决策表不表示时间，所以对于“什么也不做”入口来说，也可能有其他理由。

表 3-12 铁路道口门控制器的混合入口决策表

规则	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c1: 火车数	0			1			2			3						
c2: 火车到达	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
c3: 火车离开	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
a1: 下闸		X														
a2: 升闸							X									
a3: 什么也不做				X	X			X	X			X				X
a4: 增加火车数量		X				X				X						
a5: 减少火车数量							X				X					X
a6: 不可能的情况	X		X									X	X			

3.4 基于决策表派生的测试用例

基于决策表派生测试用例是可能的，但是过程会比较麻烦。最大的问题是，决策表是陈述性的，因此不能表示其顺序。如果决策表已经被优化过，则情况会更糟糕，因为优化过程可能会改变所有或者部分条件、行为和规则。而我们也因此丧失了测试用例中输入的顺序特性，而且决策表的优化过程也可能会隐藏某些有价值的测试用例。

当然，如果设计和开发决策表的过程足够谨慎，我们还是可以获得比较完整的信息的。同样，也可以辨识出其冗余性和不确定性，相应的测试用例就能避免这些问题。然而，因为我们永远不能自动检测出缺失的条件或缺失的行为，所以我们只能获得有限的完整性。这时候就需要领域经验了，这也是基于决策表派生测试用例的过程中最好由手工完成的原因。决策表对于计算型应用来说是非常适合的，但是对于事件型驱动应用就差一点。此外，如果事件和数据的上下文与条件桩能够仔细地分开，那么我们就能比较容易地识别出上下文敏感的输入事件。在计算型和决策敏感型应用中，规则是测试用例很好的来源。在事件驱动型应用中，测试用例可能要对应一系列的规则。

3.4.1 保费计算问题的决策表

第1章中提到的保费计算问题几乎可以直接开发设计为混合入口决策表（MEDT）。年龄和“出险次数”变量已经定义了范围，可以直接引出扩展入口的条件 c1 和 c2，如表 3-13 所示。优先减免是两个布尔量，因此条件 c3 和 c4 是有限入口条件。出于节省空间的考虑，表 3-13 分成 4 个部分。此处没有不可能的规则，也没有“什么也不做”的行为。拒保条件在表 3-13 的第四部分。

对于一个应用来说，开发出一个完整的决策表本身就有极大的好处，因为在整个过程中，我们可以确保没有任何缺失的条件。当然决策表的描述性特质在这里是一个隐含的问题。基于年龄的罚款系数应该加在因为有错被罚款之前，否则罚款金额可能不正确。如果使用决策表的绝对描述性特性来生成测试用例，那么这种事情就没法避免，此时就需要领域经验了。

表 3-13 保费计算问题决策表（第一部分）

c1: 年龄	16 ≤ 年龄 < 25											
c2: 出险次数	0				1 ~ 3				4 ~ 10			
c3: 好学生?	T	T	F	F	T	T	F	F	T	T	F	F
c4: 非饮酒者?	T	F	T	F	T	F	T	F	T	F	T	F
a1: 基本利率乘以为 1.5	X	X	X	X	X	X	X	X	X	X	X	X
a2: 基本利率乘以为 1												
a3: 基本利率乘以 1.2												
a4: 基本费用增加 0 美元	X	X	X	X								
a5: 基本费用增加 100 美元					X	X	X	X				
a6: 基本费用增加 300 美元									X	X	X	X
a7: 基本费用减免 50 美元	X	X			X	X			X	X		
a8: 基本费用减免 75 美元	X		X		X		X		X		X	
规则	1	2	3	4	5	6	7	8	9	10	11	12

表 3-13 保费计算问题决策表（第二部分）

c1: 年龄	25 ≤ 年龄 < 65											
c2: 出险次数	0				1 ~ 3				4 ~ 10			
c3: 好学生?	T	T	F	F	T	T	F	F	T	T	F	F
c4: 非饮酒者?	T	F	T	F	T	F	T	F	T	F	T	F
a1: 基本利率乘以 1.5												
a2: 基本利率乘以 1	X	X	X	X	X	X	X	X	X	X	X	X
a3: 基本利率乘以 1.2												
a4: 基本费用增加 0 美元	X	X	X	X								
a5: 基本费用增加 100 美元					X	X	X	X				
a6: 基本费用增加 300 美元									X	X	X	X
a7: 基本费用减免 50 美元	X	X			X	X			X	X		
a8: 基本费用减免 75 美元	X		X		X		X		X		X	
规则	13	14	15	16	17	18	19	20	21	22	23	24

表 3-13 保费计算问题决策表（第三部分）

c1: 年龄	65 ≤ 年龄 ≤ 90											
c2: 出险次数	0				1 ~ 3				4 ~ 10			
c3: 好学生?	T	T	F	F	T	T	F	F	T	T	F	F
c4: 非饮酒者?	T	F	T	F	T	F	T	F	T	F	T	F
a1: 基本利率乘以 1.5												
a2: 基本利率乘以 1												
a3: 基本利率乘以 1.2	X	X	X	X	X	X	X	X	X	X	X	X
a4: 基本费用增加 0 美元	X	X	X	X								
a5: 基本费用增加 100 美元					X	X	X	X				
a6: 基本费用增加 300 美元									X	X	X	X
a7: 基本费用减免 50 美元	X	X			X	X			X	X		
a8: 基本费用减免 75 美元	X		X		X		X		X		X	
规则	25	26	27	28	29	30	31	32	33	34	35	36

表 3-13 保费计算问题决策表（第四部分）

c1: 年龄	< 16	> 90	—
c2: 出险次数	—	—	> 10
c3: 好学生?	—	—	—
c4: 非饮酒者?	—	—	—
a9: 不能投保	X	X	X
规则	37	38	39

在保费计算问题中，MEDT 里面有 39 个不同的规则。每个规则都定义了一个抽象的测试用例，它们也对应第 2 章保费计算问题流程图中的路径。减少这些逻辑用例，生成具体测试用例，这是不能在决策表中完成的。除了一些复合条件中年龄小于 16 岁或大于 90 岁的，这 39 个用例与第 2 章流程图中派生出来的用例紧密相关。表 3-14 可以从表 3-13 的第一部分派生出来。

表 3-14 保费计算问题抽象测试用例

保费计算问题抽象测试用例 (规则 1)	
前置条件	基本费用、年龄、出险次数、是否为好学生、是否为非饮酒者都是已知的
输入	动作
1. 年龄 $16 \leqslant \text{年龄} < 25$	2. 基本利率乘以 1.5
3. 出险次数 = 0	4. 基本利率乘以 1
5. 是好学生	6. 从基本费用中减去 50 美元
7. 非饮酒者	8. 从基本费用中减去 75 美元
后置条件	基本费用包含保费

表 3-15 不能从表 3-13 中派生出来，但可以从表 3-13 中的规则里面手工开发出来。

表 3-15 保费计算问题具体测试用例

保费计算问题具体测试用例 (规则 1)	
前置条件	基本费用 = 600 美元
输入	动作
1. 年龄 = 20	2. $900 = 600 \times 1.5$
3. 出险次数 = 0	4. $900 = 900 - 0$
5. 是好学生	6. $850 = 900 - 50$
7. 非饮酒者	8. $775 = 850 - 75$
后置条件	保费 = 775 美元

3.4.2 车库门控系统的决策表

表 3-16 是针对车库门控系统的决策表模型。从表中可以看到，在车库门某个给定的上下文内应该发生的事件（条件 c1 的入口）。出于空间的考虑，该决策表分成两部分，第一个部分对应关车库门的操作，第二部分对应开车库门的操作。我们也可以看到上下文相关的输入事件，例如在规则 1 中，控制信号的响应是，向下起动驱动电机；而在规则 5 中，对于同样的输入事件，是停止电机。表 3-16 使用了 F!（必定为假）标识符，显示在每个上下文（有些由不可能的规则所指示）中被禁止的事件。这里面也有个约定俗成的建模传统：禁止同时发生的事情。在这个表中，由于允许每个上下文的每个事件发生，因此我们有若干个不可能的规则。

表 3-16 车库门控系统问题决策表（第一部分）

规则	1	2	3	4	5	6	7	8	9	10	11	12
c1: 门的状态	开启				正在关闭				停止关闭			
c2: 发出控制信号	T	F!	F!	F!	T	F!	F!	F!	T	F!	F!	F!
c3: 到达轨道下端	F!	T	F!	F!	F!	T	F!	F!	F!	T	F!	F!
c4: 到达轨道上端	F!	F!	T	F!	F!	F!	T	F!	F!	F!	T	F!
c5: 光束被阻碍	F!	F!	F!	T	F!	F!	F!	T	F!	F!	F!	T
a1: 起动电机向下	X								X			
a2: 起动电机向上												
a3: 停止电机					X	X						
a4: 停止向下并反转									X			
a5: 什么也不做				X								X
a6: 不可能		X	X				X			X	X	
a7: 循环表格	X			X	X	X		X	X			X

表 3-16 车库门控系统问题决策表（第二部分）

规则	13	14	15	16	17	18	19	20	21	22	23	24
c1: 门的状态	关闭				正在开启				停止开启			
c2: 发出控制信号	T	F!	F!	F!	T	F!	F!	F!	T	F!	F!	F!
c3: 到达轨道下端	F!	T	F!	F!	F!	T	F!	F!	F!	T	F!	F!
c4: 到达轨道上端	F!	F!	T	F!	F!	F!	T	F!	F!	F!	T	F!
c5: 光束被阻碍	F!	F!	F!	T	F!	F!	F!	T	F!	F!	F!	T
a1: 起动电机向下												
a2: 起动电机向上	X								X			
a3: 停止电机					X		X					
a4: 停止向下并反转												
a5: 什么也不做				X				X				X
a6: 不可能		X	X			X				X	X	
a7: 循环表格	X			X	X		X	X	X			X

一个简单的派生测试用例的原则是，一个规则应该对应一个测试用例。但问题是规则都很短小，只涉及一个输入事件。因此，将其理解为激励 / 响应对会更好些。在第 4 章，我们会看到这些基于激励 / 响应对的规则与有限状态机（针对车库门控系统）描述的事件可以很好地关联。利用表 3-16 派生出全部的测试用例是可能的，但只能手工完成。此时需要领域知识来减少敏感的规则序列，我们使用行为 a7（循环表格）来表述这些规则。决策表的陈述特性是另一个复杂的因素，规则的选择和执行会有一些“自然”顺序，但这些都没有表现出来。还有一点，决策表是一次性描述。除非选择循环表格动作，否则没法显示出关门和重新开门等循环。

3.4.3 车库门控系统的测试用例

从表面上看，车库门控制器有 24 个规则，MEDT 可以生成 24 个测试用例。5 个“什么也不做”行为（规则 4、12、16、20 和 24）都很奇怪，你怎么去测试“什么也不做”这样的事情呢？这些规则中的每一条都涉及光束被打断这个事件，但这个事件只有当门关上的时候（规则 8）才会激活。这 5 个“什么也不做”的规则，就是应该被忽略的物理事件。（在第 8 章，我们会更进一步讨论如何使能和禁用光束传感器。）有 10 个不可能的用例（规则 2、3、7、10、11、14、15、18、22 和 23）都是物理上不可能的。它们指的是到达轨道顶端这个事件，而这是不可能发生的。现在就剩下 9 个真实的测试用例（规则 1、5、6、8、9、13、17、19、21）。这些行为都是合法的端口输出事件，这 9 个都是具体测试用例。如前所述，它们都是短小的激励 / 响应对。我们可以将这些基于规则的、短小的测试用例与规则序列相关联，从而使其更符合端对端的测试用例要求。表 2-3 中的 5 个测试用例在表 3-17 中以规则序列的方式呈现。

表 3-17 车库门控系统测试用例的规则序列

车库门控系统决策表规则序列		
路径	描述	规则序列
1	门正常关闭	1, 6
2	门中间停止一次后正常关闭	1, 5, 9, 6
3	门正常开启	13, 17
4	门中间停止一次后正常开启	13, 17, 21, 19
5	门正在关闭时，光束传感器被阻断	1, 8, 19

3.5 优势与局限

对于逻辑敏感型应用来说，决策表很明显是一种建模选择。如果能将条件表示为带有依赖关系的等价类（比如 NextDate 应用），那么就更适用了。同样，利用代数简化过的决策表可以让我们用更优雅的方式最小化决策表（精简决策表）。使用决策表的最大问题就是应用中的计算只能表示成行为，而且无法对行为的顺序关系进行表达。最后，输入事件必须表示为条件，输出事件表示为行为。有必要再强调一下，不能表示顺序关系这一点使得决策表这种方法对于事件驱动的应用显得不太够用。表 3-18 总结了决策表中行为事件的各种表达方式。

3.6 经验教训

在 20 世纪 70 年代，我是 CODASYL 决策表任务小组的成员，我们当时需要创建一个针对决策表的“定义性的”描述，包括推荐最终的最佳方案。最终产品由 ACM 公司出版，是一个加大的纸质文档的版本。在这个过程中，我们经历了几个很有趣的阶段。其中之一是，团队中有一名成员来自比利时，他提议我们应当针对当时正在实施的租赁控制法案制作一个决策表，因为这个法案非常让人困惑。在后续的一个季度会议中，我们将各自的工作成果汇聚成一个固定的决策表，搞清了这个法案中的很多事情，甚至包括一组不一致性。

另一个小组成员 Lewis Reinwald 有一个基于决策表的程序，它可以清除被过度修改过的 Fortran 程序。通常来说，这些程序是被一群既不懂原始程序也不懂太多之前修改原因的开发人员开发的。在一个测试用例中，Reinwald 先生处理了一个非常庞大且维护过度的（简直就是噩梦）的 Fortran 程序，该程序有 2000 多行源代码。他从源程序中派生出一个简化的决策表，并手工“清理”了决策表，然后生成了原始程序的改进版本。改进后的版本只有 800 行语句。

从 1978 ~ 1981 年，我为一个意大利公司工作。在需求规范化过程中，我们使用有限状态机的组合，继而细化成决策表。我们发现，决策表的结构能够不断逼迫我们考虑各种可能的场景，而这些场景如果不是因为决策表的活动，那么可能一直到开发后期才被察觉。

在大学教授的课程里，我使用了已经通过州立法的关于教师退休计划奖励作为例子。在这个例子中我们发现一对会导致退休福利计算结果混乱的互相矛盾的条件。上述 3 个例子都说明，决策表对于理清复杂的业务规则是非常有效率的。

表 3-18 决策表行为事件的表达

事件	表现	建议
顺序	不好	决策表是描述性的
选择	好	决策表价值所在
循环	好	必须使用循环表格行为
可用	非直接	与其他可用条件结合形成可用条件
不可用	非直接	与其他可用条件结合形成不可用条件
触发	不好	
激活	非直接	只有在顺序可用和不可用规则中才有意义，但它们没有顺序关系，所以实现起来非常复杂
挂起	非直接	真实的挂起条件需要与已有的“无关入口”或者“F！”条件入口结合使用
恢复	不好	

(续)

事件	表现	建议
暂停	不好	
冲突	不好	
优先级	不好	
互斥	好	规则是互斥的
同步	不好	
死锁	不好	
上下文敏感输入事件	好	事件是一个条件，上下文是分开的条件
多原因输出事件	好	不同规则的相同行为
异步事件	不好	
事件静默	不好	

参考文献

[CODASYL 1978]

CODASYL Systems Group, *DETAB-X, Preliminary Specification for a Decision Table Structured Language*, Association for Computing Machinery, New York, 1978.

[Jorgensen 2009]

Jorgensen, Paul C., *Modeling Software Behavior—A Craftsman's Approach*. CRC Press, Boca Raton, FL, 2009.