第2章　有限状态机及其扩展

有限状态机是有限计算的基本模型，也是许多形式化规格、验证方法的基础模型。Statecharts是通过定义递阶状态、状态的与(AND)/或(OR)分解等高级特性的有限状态机的一种扩展形式。本章对有限状态机相关的基本概念、Statecharts的基本要素及其应用进行介绍。

2.1　有限状态机

2.1.1　基本概念

在客观现实世界中，存在大量的具有有限个状态的系统。例如，钟表就是一个有限状态系统，它共有12×60×60种状态，秒针每走一步，就从一种状态转移到另外一种状态；饮料自动售货机也是一个有限状态系统，售货机根据客户的按键输入以及投入硬币的面值和枚数来指导客户操作，并提供不同规格的饮料。有限状态机或者有限自动机的概念，则是来自现实世界中的这类有限状态系统。它是关于存储量有限的计算机的基本模型，也是许多形式化规格、验证技术的基础模型。

如图2.1(a)所示，超级商场的自动门控制器，自动门的前、后分别有一个缓冲区。自动门前面的缓冲区，用来检测是否有人接近。自动门后面的缓冲区，使得控制器把门打开足够长的时间让人走进去，并且不让门在打开的时候碰到站在它附近的人。

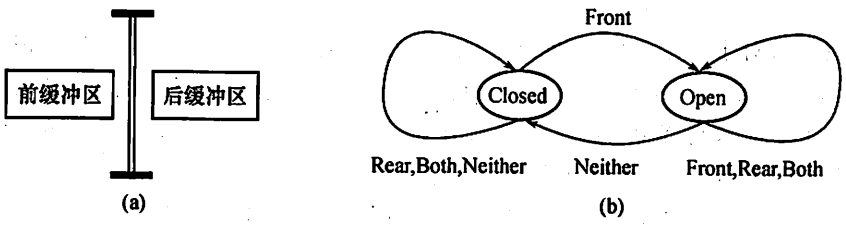


图2.1　自动门控制器

如图2.1(b)所示，控制器处于两个状态中的一个，这两个状态是：Closed和Open，分别表示门开着和关着。该控制器接收四种可能的输入：Front(表示门前面的缓冲区内有人)、Rear(表示门后面的缓冲区内有人)、Both(表示前、后缓冲区内都有人)、Neither(表示前、后

缓冲区内都没有人)。

控制器根据它接收的输入从一个状态转移到另一个状态。当它处于状态 Closed 且接收到输入 Rear 或 Neither 时,它仍处于状态 Closed;当接收到输入 Both 时,仍停留在状态 Closed,因为打开门有撞倒后缓冲区里的人的危险;但是,如果输入 Front 来到,它转移到状态 Open。在 Open 状态下,如果接收到输入 Front、Rear 或 Both,它保持在状态 Open 不动。如果输入 Neither 来到,它返回到状态 Closed。

通过上述例子不难发现,一个有限状态机包括如下几个部分:一个有限状态集,用于描述系统中的不同状态;一个输入符号集,用于表征系统所接收的不同输入信息;一个状态转移规则集,用于表述系统在接收不同输入符号下从一个状态转换到另外一个状态的规则。由此,可给出如下有限状态机的形式定义。

有限状态机　有限状态机是一个五元组 $M=(Q,\Sigma,\delta,q_0,F)$,其中

① $Q=\{q_0,q_1,\cdots,q_n\}$ 是有限状态集合。在任一确定的时刻,有限状态机只能处于一个确定的状态 q_i。

② $\Sigma=\{\sigma_1,\sigma_2,\cdots,\sigma_m\}$ 是有限输入字符集合。在任一确定的时刻,有限状态机只能接收一个确定的输入 σ_j。

③ $\delta:Q\times\Sigma\to Q$ 是状态转移函数。如果在某一确定的时刻,有限状态机处于某一状态 $q_i\in Q$,并接收一个输入字符 $\sigma_j\in\Sigma$,那么下一时刻将处于一个确定的状态 $q'=\delta(q_i,\sigma_j)\in Q$。在这里,规定 $q=\delta(q,\varepsilon)$,即,对任何状态 q,当读入空字符(即不存在任何输入字符)时,有限状态机不发生任何状态转移。

④ $q_0\in Q$ 是初始状态,有限状态机由此状态开始接收输入。

⑤ $F\subseteq Q$ 是终结状态集合,有限状态机在达到终态后不再接收输入。

给出一个有限状态机 $M=(\{q_0,q_1,q_2,q_3\},\{0,1\},\delta,q_0,\{q_0\})$,其中,状态转移函数 δ 具体定义如下:

$$\delta(q_0,1)=q_1,\ \delta(q_0,0)=q_2,\ \delta(q_1,1)=q_0,\ \delta(q_1,0)=q_3,$$
$$\delta(q_2,1)=q_3,\ \delta(q_2,0)=q_0,\ \delta(q_3,1)=q_2,\ \delta(q_3,0)=q_1。$$

我们注意到,对于一个有限状态机来说,状态转移函数是最关键的部分。根据定义,转移函数 δ 是从 $Q\times\Sigma$ 到 Q 的映射。也就是说,它是一个二元函数,第一个变元取自 Q 中的一个状态,第二个变元取自 Σ 中的一个符号,函数值是 Q 中的一个状态。在上例中,Q 中有 4 个状态,Σ 中有 2 个字符,所以要列出 8 个式子才能给出 δ 的完全定义。

事实上,状态转移函数 δ 建立了有限状态集上的一个关系 R_M,即

qR_Mq',当且仅当 $\delta(q,\sigma)=q'$

注意:

① 关系 R_M 是与输入字符有关的;

② $\delta(q,\sigma)$ 有时也简记为 $\delta_\sigma(q)$,即用 δ_σ 表示对于输入字符 σ 时的转移函数;

③ 对于状态 q 和输入字符 σ，如果不存在一个状态 q′，使得 δ(q,σ) = q′成立，则称状态 q 对于输入字符 σ 没有转移，记作 δ(q,σ) = Ø。

为直观、简便起见，状态转移函数通常可以用关系矩阵、状态转移表或状态转移图的形式表示。

状态转移矩阵 对于有限状态机 M = (Q,Σ,δ,q_0,F) 的转移函数 δ，用行表示状态机所处的当前状态，列表示将要到达的下一个状态，行列交叉处表示输入字符。称该矩阵为转移函数 δ 的关系矩阵，或者有限状态机 M 的状态转移矩阵。

上例中有限状态机 M 的关系矩阵为

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array}\begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{bmatrix} & 1 & 0 & \\ 1 & & & 0 \\ 0 & & & 1 \\ & 0 & 1 & \end{bmatrix} \end{array}$$

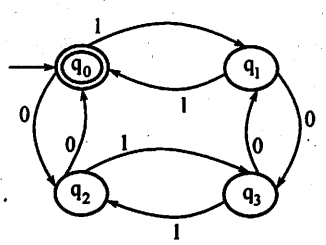
状态转移表 对于有限状态机 M = (Q,Σ,δ,q_0,F) 的转移函数 δ，用表格的行表示状态机所处的当前状态，列表示当前的输入字符，行列交叉处表示要到达的下一个状态。称该表格为有限状态机 M 的状态转移表。

状态转移图 对于有限状态机 M = (Q,Σ,δ,q_0,F) 的转移函数 δ，用圆圈(结点)表示状态；将存在转移关系的状态用有向弧连接，并标注相应的输入字符在有向弧旁；用标有箭头的结点表示初始状态；属于终结状态集中的状态用双圈表示。由上述规则所建立的有向图称为状态转移图。

图 2.2(a)、(b)分别为上例中有限状态机的状态转移表和状态转移图。

状态	输入字符 0	输入字符 1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

(a)



(b)

图 2.2 有限状态机的图形表示

在一些情况下，我们讨论问题的重点集中在有限状态机 M = (Q,Σ,δ,q_0,F) 的有限状态集 Q、有限输入字符集 Σ 和状态转移函数 δ 上。这时候，有限状态机可用四元组 M = (Q,Σ,δ,q_0) 或三元组 M = (Q,Σ,δ) 来表示。

从有限状态机模型可知，只要定义了状态转移函数 δ，那么对于任何输入字符序列，有限

状态机就可以进行一系列动作。那么,输入字符序列和有限状态机的关系、终结状态集的作用如何呢?为回答这个问题,引入如下关于字符串及语言的概念。

有限字母表　由有限个任意符号组成的非空集合,简称为字母表,用 Σ 表示。字母表上的元素称作字符或符号。常用的数字(0~9)、大小写字母(A~Z,a~z)、括号、运算符(+,-,*,/)等均可作为字母表上的元素。

字母表可以理解为计算机输入键盘上符号的集合。字符可以理解为键盘上的每一个英文字母、每一个数字、每一个标点符号、每一个运算符号等。

字符串　由字符组成的有限序列,称为字符串或者符号串,常用小写希腊字母表示。字母表 Σ 上的字符串以下列方式生成:

① ε 为 Σ 上的一个特殊串,称为空串,对任何 $a\in\Sigma$,$a\varepsilon=\varepsilon a=a$;

② 若 σ 是 Σ 上的符号串,且 $a\in\Sigma$,则 σa 是 Σ 上的符号串;

③ 若 α 是 Σ 上的符号串,当且仅当它由①和②导出。

直观来说,Σ 上的符号串是由其上的符号以任意次序拼接起来构成的,任何符号都可以在串中重复出现。ε 作为一个特殊的串,由零个符号组成。应当指出的是,空串 ε 不同于计算机键盘上的空格键。例如,当 $\Sigma=\{a,b\}$ 时,aabb、ab、abab、bba、ε 都是 Σ 上的符号串。

对于字符串可以定义如下一些运算或者操作。

字符串的长度　字符串中所包含的字母的个数,用 $|\alpha|$ 表示。例如,字符串 aabb 的长度为4,记作 $|aabb|=4$。空串 ε 的长度为0,记作 $|\varepsilon|=0$。

字符串的连接　依次把两个字符串拼接在一起,用"·"表示。字符串的连接仍为字符串。一般地,字符串的连接不满足交换律。并且,对任何字符串 α,$\alpha\cdot\varepsilon=\varepsilon\cdot\alpha=\alpha$。连接运算符在实际应用中也可略去不写。例如,$\alpha\cdot\varepsilon=\varepsilon\cdot\alpha=\alpha\varepsilon=\varepsilon\alpha=\alpha$;对于字符串 α = aabb 和字符串 β = abab,$\alpha\cdot\beta$ = aabbabab、$\beta\cdot\alpha$ = ababaabb。

字符串的重复连接　对于一个正整数 n 和一个字符串 α,重复地将其依次拼接 n 次,称为字符串 α 的 n 次重复连接,记为 α^n。例如,对于字符串 α = aab,α^3 = aabaabaab。

语言　给定字母表 Σ 上的字符串的集合称为语言。例如,当 $\Sigma=\{a,b\}$,则 $\{aabb,ab,abab,bba\}$、$\{\varepsilon\}$、$\{a^nb^n\mid n\geqslant 1\}$ 都是 Σ 上的语言。Σ 本身也是 Σ 上的语言。不包含任何字符串的语言称作空语言,用 $\varnothing$ 表示。注意:$\{\varepsilon\}$ 不同于 $\varnothing$,前者表示由空串组成的语言,而后者表示空语言。字母表 Σ 上的所有字符串连同空串 ε 一起构成的语言用 Σ^* 表示。显然,字母表 Σ 上的任何语言 L 是 Σ^* 的子集,即,$L\subseteq\Sigma^*$。

扩展转移函数 $\bar{\delta}$　对于有限状态机 $M=(Q,\Sigma,\delta,q_0,F)$,它的扩展转移函数 $\bar{\delta}$,是从 $Q\times\Sigma^*$ 到 Q 的映射,具体定义如下:

① $\bar{\delta}(q,\varepsilon)=q$;

② $\bar{\delta}(q,w\sigma)=\delta(\bar{\delta}(q,w),\sigma)$。

其中,$q\in Q$、$w\in\Sigma^*$、$\sigma\in\Sigma$。

上述是一个递归定义，它将原来 δ 中的第二个变元由一个字符扩展为一个字符串，函数值仍为一个状态。例如，对于图 2.2 中的有限状态机，就有

$$\bar{\delta}(q_0,010)=\delta(\bar{\delta}(q_0,01),0)=\delta(\delta(\bar{\delta}(q_0,0),1),0)$$
$$=\delta(\delta(\delta(\bar{\delta}(q_0,\varepsilon),0),1),0)=\delta(\delta(\delta(q_0,0),1),0)$$
$$=\delta(\delta(q_2,1),0)=\delta(q_3,1)=q_1$$

实际上，$\bar{\delta}(q,x)$ 的值就是从 q 出发用原来的转移函数 δ，每经过 x 中的一个符号后改变一次状态，直到经过 x 的最后一个符号所得到的状态。这样看来从 δ 到 $\bar{\delta}$ 的扩展是很自然的，而且 δ 就是 $\bar{\delta}$ 的特例（当 $|x|=1$ 时）。后面在讨论问题的时候，将不再强调 $\bar{\delta}$ 的写法，而一律用 δ 表示。

有限状态机所接受的语言 设有限状态机 $M=(Q,\Sigma,\delta,q_0,F)$，若对于输入串 $x\in\Sigma^*$ 有 $\delta(q_0,x)=q'\in F$，则称字符串 x 被有限状态机 M 所接受，或者称 x 为有限状态机 M 所接受的句子。进一步地，被 M 接受的全部字符串构成的集合，称为有限状态机 M 所接受的语言，并记作 L(M)，即

$$L(M)=\{x|\delta(q_0,x)\in F\}$$

上述定义刻画了一个有限状态机和一个字符串之间的关系，同时也表明了终结符号集的意义。一个有限状态机所能接受的只是 Σ^* 的一个子集，有很多 Σ^* 的子集是任何有限状态机所不能接受的。事实上，有限状态机所接受的语言也就是所对应有限状态机行为的符号串规格。

对于图 2.2 中的有限状态机，可以给出这样的结论：该有限状态机接受的语言是一切含有偶数个 0 和偶数个 1 的字符串组成的集合。为了说明这一结论，下面分析一下其中四个状态所起的作用。初始状态（也是惟一的终结状态）q_0 表示已接受过含有偶数个 0 和偶数个 1 的字符串所到达的状态，q_1 表示已接受过含有偶数个 0 和奇数个 1 的字符串所到达的状态，q_2 表示已接受过含有奇数个 0 和偶数个 1 所到达的状态，q_3 表示已接受过含有奇数个 0 和奇数个 1 所到达的状态。因为一切由 0 和 1 组成的字符串必属于这四种情况之一，该有限状态机所设的四个状态正好对应这四种情况，而且定义的 δ 函数也正好能正确处理状态间的转移关系（例如 $\delta(q_1,x)=q_3$ 使已接受过的 0 的个数由偶变奇）。这就不难看出，该有限状态机确实接受由含有偶数个 0 和偶数个 1 的字符串组成的集合。

如下给出的是接受 Pascal 语言中 $\pm\alpha.\beta E\pm\gamma$ 形式的实数（α、β、γ 均为非负整数）的有限状态机。

设有限状态机 $M=(Q,\Sigma,\delta,q_0,\{q_f\})$，其中，$Q=\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_f\}$，$\Sigma=\{0,1,2,3,4,5,6,7,8,9,.,E,+,-\}$，状态转移函数 δ 如图 2.3(a)所示。

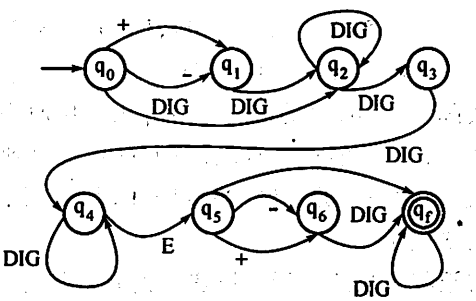
在状态转移表中，用 DIG 表示数字 0,1,2,3,4,5,6,7,8,9。下面考察这个有限状态机对输入串 123.45E+67 的接受过程：

$$\delta(q_0,123.45E+67)\rightarrow\delta(q_2,23.45E+67)\rightarrow\delta(q_2,3.45E+67)\rightarrow\delta(q_2,.45E+67)\rightarrow$$

$\delta(q_4,45E+67)\rightarrow\delta(q_4,5E+67)\rightarrow\delta(q_4,E+67)\rightarrow\delta(q_5,+67)\rightarrow\delta(q_6,67)\rightarrow\delta(q_f,7)\rightarrow q_f$。所以,123.45E+67 是该有限状态机所接受的句子。

状态	输入字符				
	DIG	.	E	+	-
q_0	q_2	ϕ	ϕ	q_1	q_1
q_1	q_2	ϕ	ϕ	ϕ	ϕ
q_2	q_2	q_3	ϕ	ϕ	ϕ
q_3	q_4	ϕ	ϕ	ϕ	ϕ
q_4	q_4	ϕ	q_5	ϕ	ϕ
q_5	q_f	ϕ	ϕ	q_6	q_6
q_6	q_f	ϕ	ϕ	ϕ	ϕ
q_f	q_f	ϕ	ϕ	ϕ	ϕ

(a)



(b)

图 2.3　有限状态机的状态转移表和状态转移图

前面讨论的有限状态机,可以看作仅接受某一种字母表上的输入符号串并发生状态改变,但无任何字符串输出的自动机器。实际上,现实生活中的许多有限状态系统对于不同的输入信号,除内部状态不断改变外,还不断向系统外部输出各种信号。下面将有限状态机进行推广,引出具有字符串输出的自动机器模型。这样的自动机器模型按照输出的不同分成两类:输出与状态有关——Moore 机;输出与状态和输入有关——Mealy 机。

Moore 机　Moore 机形式定义为六元组 $M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$,其中

① $Q=\{q_0,q_1,\cdots,q_n\}$ 是有限状态集合;

② $\Sigma=\{\sigma_1,\sigma_2,\cdots,\sigma_m\}$ 是有限输入字符集合;

③ $\Delta=\{a_1,a_2,\cdots,a_r\}$ 是有限输出字符集合;

④ $\delta:Q\times\Sigma\rightarrow 2^Q$ 是状态转移函数;

⑤ $\lambda:Q\rightarrow\Delta$ 是输出函数;

⑥ $q_0\in Q$ 是初始状态。

在上述 Moore 机的定义中没有包含终结状态部分,因此也就不存在“接受”或者“拒绝”一个输入串的问题。Moore 机只是在接受输入串的过程中不断改变状态,并且在每个状态上有字符输出。例如,对于输入串 $\sigma_1\sigma_2\cdots\sigma_m$,设 $\delta(q_0,\sigma_1)=q_1,\delta(q_1,\sigma_2)=q_2,\cdots,\delta(q_{i-1},\sigma_i)=q_i,\cdots,\delta(q_{n-1},\sigma_n)=q_n$。这时输出序列为 $\lambda(q_0)\lambda(q_1)\cdots\lambda(q_i)\cdots\lambda(q_n)$。

有限状态机可看作为 Moore 机的一个特例。事实上,对于任何一个有限状态机 $M=(Q,\Sigma,\delta,q_0,F)$,引入输出字符集合 $\Delta=\{0,1\}$,并定义 Q 到 Δ 的映射 λ 为:对于 $q\in F,\lambda(q)=1$;对于 $q\notin F,\lambda(q)=0$。这样就得到一个 Moore 机 $M'=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$,在该 Moore 机中,输出为 1 的状态即为终结状态,输出为 0 的状态即为非终结状态。

如下为一个接受二进制数输入、并输出所接受输入的模3余数的Moore机。

接受二进制数输入,则输入字符集合为 $\Sigma=\{0,1\}$。由于模3余数只能有0、1、2三个值,因此取 $\Delta=\{0,1,2\}$,并且设3个状态 q_0、q_1、q_2 分别对应这三种余数。相应的Moore机为 $M=(\{q_0,q_1,q_2\},\{0,1\},\{0,1,2\},\delta,\lambda,q_0)$,其中,状态转移函数 δ 由M的输入状态转移图(图2.4)给出;输出函数为 $\lambda(q_j)=j,j=1,2,3$。

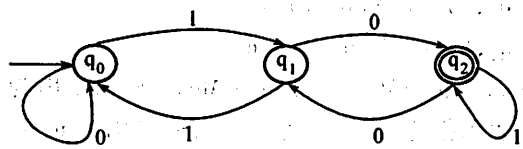


图2.4 计算模3余数的Moore机

对于输入串10100,由M的输入状态转换可知,Moore机所经过的状态序列为 q_0,q_1,q_2,q_2,q_1 相应的输出为 $\lambda(q_0)\lambda(q_1)\lambda(q_2)\lambda(q_2)\lambda(q_1)=01221$。也就是说,当输入为 ε 时,输出余数为0;当输入为1时,输出余数为1;当输入为10时,输出余数为2;当输入为101时,输出余数为2;当输入为1010时,输出余数为1;当输入为10100时,输出余数为2。

Mealy机 Mealy机形式定义为六元组 $M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$,其中

① $Q=\{q_0,q_1,\cdots,q_n\}$ 是有限状态集合;

② $\Sigma=\{\sigma_1,\sigma_2,\cdots,\sigma_m\}$ 是有限输入字符集合;

③ $\Delta=\{a_1,a_2,\cdots,a_r\}$ 是有限输出字符集合;

④ $\delta:Q\times\Sigma\rightarrow 2^Q$ 是状态转移函数;

⑤ $\lambda:Q\times\Sigma\rightarrow\Delta$ 是输出函数;

⑥ $q_0\in Q$ 是初始状态。

在上述Mealy机定义中,除输出函数 λ 外,Q、Σ、Δ、δ、q_0 的含义均同Moore机。$\lambda(q,\sigma)=a$ 给出了当机器进入状态 q,并得到输入 σ 时的输出为 a。当输入串为 $\sigma_1\sigma_2\cdots\sigma_m$ 时,设 $\delta(q_0,\sigma_1)=q_1,\delta(q_1,\sigma_2)=q_2,\cdots,\delta(q_{i-1},\sigma_i)=q_i,\cdots,\delta(q_{n-1},\sigma_n)=q_n$。这时输出序列为 $\lambda(q_0,\sigma_1)\lambda(q_1,\sigma_2)\cdots\lambda(q_i,\sigma_{i+1})\cdots\lambda(q_{n-1},\sigma_n)$。值得注意的是,Mealy机与Moore机不同,当输入串长度为n时,它输出n个符号,而不是n+1个符号。

图2.5所示为一Mealy机 $M=(\{q_0,q_1,q_2\},\{0,1\},\{y,n\},\delta,\lambda,q_0)$。根据Mealy机的特性,$q_1$ 表示M进入这个状态时,已接收的输入串的最后一个字符为0;q_2 表示M进入这个状态时,已接收的输入串的最后一个字符为1;q_0 表示M的初始状态。

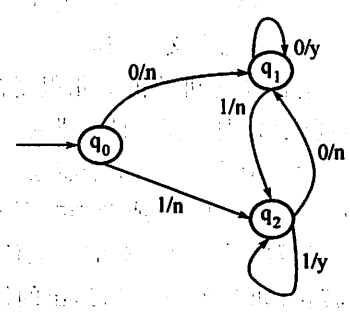


图2.5 Mealy机的状态转移图

在 Mealy 机的状态转移图中，在从状态 q_i 到状态 q_j 的弧上标记了 a/b，用以表示其输入和输出，即，$\delta(q_i,a)=q_j$，$\lambda(q_i,a)=b$。由图 2.5，可得出该 Mealy 机的状态转换函数和输出函数为：

$$\delta(q_0,0)=q_1,\delta(q_0,1)=q_2,\delta(q_1,1)=q_2,$$
$$\delta(q_1,0)=q_1,\delta(q_2,1)=q_2,\delta(q_2,0)=q_1,$$
$$\lambda(q_0,0)=n,\lambda(q_0,1)=n,\lambda(q_1,1)=n,$$
$$\lambda(q_1,0)=y,\lambda(q_2,1)=y,\lambda(q_2,0)=n。$$

对于输入串 w=01100，由状态转移图可知该 Mealy 机输出串为 nnyny。并作如下解释：当输入为 0 时，输出为 n，表示拒绝；当输入为 01 时，输出仍为 n，仍表示拒绝；当输入为 011 时，输出为 y，表示接受；当输入为 0110 时，输出为 n，表示拒绝；当输入为 01100 时，输出为 y，表示接受。

2.1.2　有限状态机的复合

大量的软件系统是以模块或子系统的形式出现的。整个软件系统的有限状态机规格，需要对各个模块的有限状态机进行复合。有限状态机复合的最简单方式是笛卡儿积。

对于有限状态机 $M_1=(Q_1,\Sigma_1,\delta_1,q_{10})$ 和 $M_2=(Q_2,\Sigma_2,\delta_2,q_{20})$，它们的笛卡儿积为 $M=M_1\times M_2=(Q,\Sigma,\delta,q_0)$，其中，

$$Q=Q_1\times Q_2;$$
$$\Sigma=(\Sigma_1\cup\{\varepsilon\})\times(\Sigma_2\cup\{\varepsilon\});$$
$$q_0=(q_{10},q_{20})\in Q_1\times Q_2;$$
$$\delta((q_1,q_2),(a_1,a_2))=$$
$$\begin{cases}(q_1',q_2) & \delta_1(q_1,a_1)=q_1',a_2=\varepsilon,a_1\in\Sigma_1\\(q_1,q_2') & \delta_2(q_2,a_2)=q_2',a_1=\varepsilon,a_2\in\Sigma_2\\(q_1',q_2') & \delta_1(q_1,a_1)=q_1',\delta_2(q_2,a_2)=q_2',a_1\in\Sigma_1,a_2\in\Sigma_2\\\text{无定义} & \text{其余}\end{cases}$$

有限状态机的笛卡儿积复合规格了多个有限状态机相互独立运行的行为。图 2.6(a)、(b)所示分别为模 3 计数器和模 4 计数器的有限状态机 M_1 和 M_2。两个模计数器的笛卡儿积复合具有 3×4=12 个状态。在每一个状态下，两个模计数器均可相互独立地进行加数(inc)、减数(dec)，或者维持不变，这样就有 3×3=9 种可能的状态转移选择。考虑到两个模计数器维持不变，对应于无任何状态转移发生。因此，每一个状态下具有 8 种可发生的状态转移，如图 2.6(c)所给出的笛卡儿积复合有限状态机 M 始于状态(0,0)的状态转移，该有限状态机具有 8×12=96 个状态转移。

软件系统之间的交互是大家经常遇到的问题，有限状态机的复合也必然涉及交互问题。对于图 2.6 的两个模计数器，可能希望它们耦合运行，阻止各自的独立运行行为。那么，就

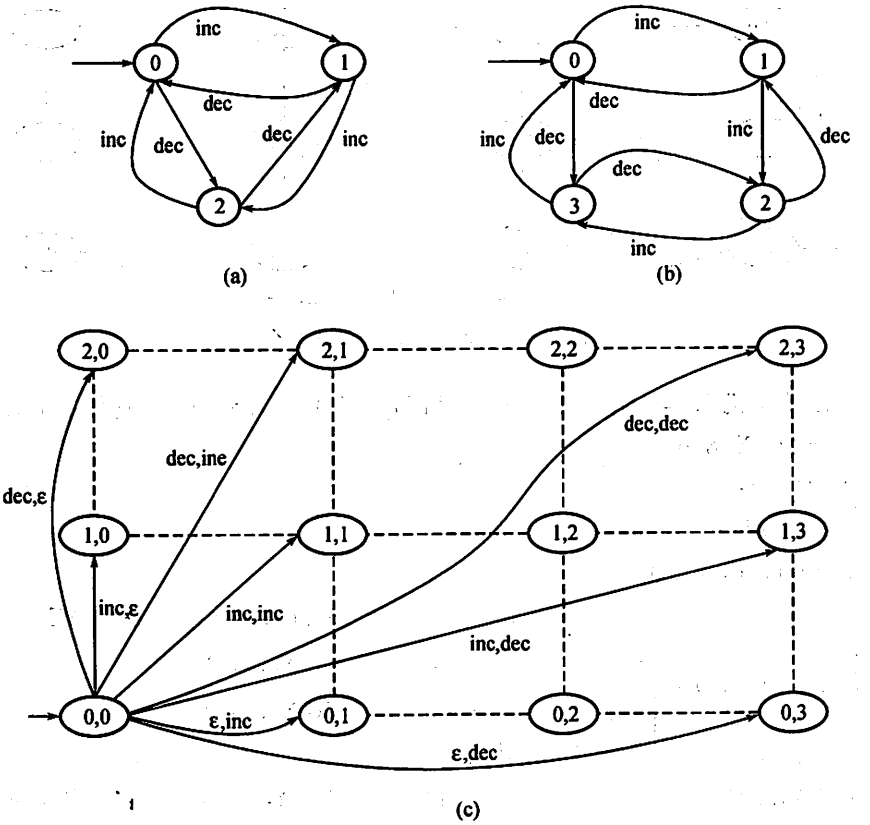


图 2.6 有限状态机的笛卡儿积复合

只有 inc,inc 和 dec,dec 两种共 24 个状态转移。另一方面,也可能希望它们耦合运行,但任何状态下仅有其中一个模计数器运行。考虑有限状态机之间交互的复合,可以通过如下同步积复合、异步积复合来得到。图 2.7、图 2.8 所示分别为模计数器的同步积复合和异步积复合,图中仅给出了始于状态(0,0)的状态转移。

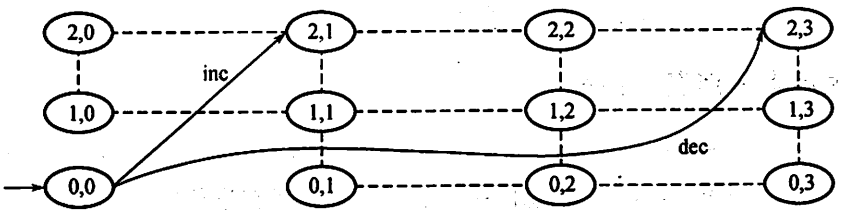


图 2.7 模计数器的同步积复合

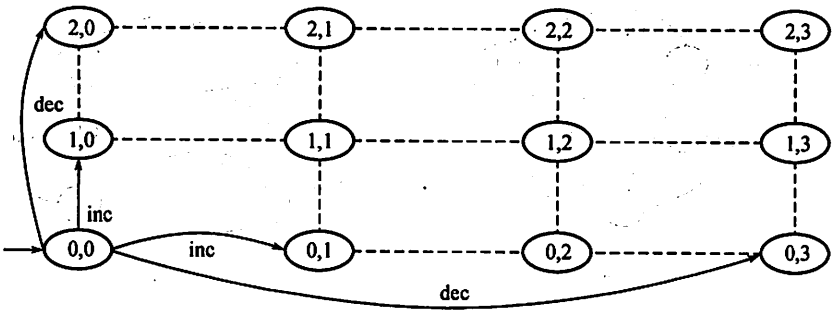


图 2.8　模计数器的异步积复合

对于有限状态机 $M_1=(Q_1,\Sigma_1,\delta_1,q_{10})$ 和 $M_2=(Q_2,\Sigma_2,\delta_2,q_{20})$，它们的同步积复合为 $M=M_1\otimes M_2=(Q,\Sigma,\delta,q_0)$，其中，

$$Q=Q_1\times Q_2;$$

$$\Sigma=\Sigma_1\cup\Sigma_2;$$

$$q_0=(q_{10},q_{20})\in Q_1\times Q_2;$$

$$\delta((q_1,q_2),a)=\begin{cases}(q_1',q_2') & \delta_1(q_1,a)=q_1',\delta_2(q_2,a)=q_2',a\in\Sigma_1\cap\Sigma_2\\(q_1',q_2) & \delta_1(q_1,a)=q_1',a\in\Sigma_1,a\notin\Sigma_2\\(q_1,q_2') & \delta_2(q_2,a)=q_2',a\in\Sigma_2,a\notin\Sigma_1\\\text{无定义} & \text{其余}\end{cases}$$

对于有限状态机 $M_1=(Q_1,\Sigma_1,\delta_1,q_{10})$ 和 $M_2=(Q_2,\Sigma_2,\delta_2,q_{20})$，它们的异步积复合为 $M=M_1\oplus M_2=(Q,\Sigma,\delta,q_0)$，其中，

$$Q=Q_1\times Q_2;$$

$$\Sigma=\Sigma_1\cup\Sigma_2;$$

$$q_0=(q_{10},q_{20})\in Q_1\times Q_2;$$

$$\delta((q_1,q_2),a)=\begin{cases}(q_1',q_2) & \delta_1(q_1,a)=q_1',a\in\Sigma_1\\(q_1,q_2') & \delta_2(q_2,a)=q_2',a\in\Sigma_2\\\text{无定义} & \text{其余}\end{cases}$$

2.1.3　生产者－消费者系统

“生产者－消费者”系统中，包含一个生产者和一个消费者。生产者进程产生消息，并把产生的消息写入一个能容纳两个消息的缓存器中。生产者在进行“生产”动作后，状态由 P_1 转变为 P_2；而在“写”动作后，状态由 P_2 恢复为 P_1。消费者进程能读取消息，并把消息从缓存器中取走。消费者在“读”动作后，状态由 C_1 转变为 C_2；而在进行“消费”动作后，状态由 C_2 恢复为 C_1。如果缓存器是满的，那么生产者进程必须等待，直到消费者进程从缓存器中

取出一个消息，使缓存器产生一个消息空位。同样，如果缓存器是空的，那么消费者进程就必须等待，直到生产者进程产生一个消息并把所产生的消息写入缓存器中。缓存器在进行“读”动作后，缓存器大小减“1”，而在“写”动作后，缓存器大小加“1”。

利用有限状态机，分别对生产者、消费者和缓存器进行规格，如图 2.9(a)、图 2.9(b)和图 2.9(c)所示。图中清楚地给出了两个进程和一个缓存器的描述，但是作为一个整体系统，还需要对系统的整体行为进行规格。在这里可以利用有限状态机的同步积复合。该问题中包含三个有限状态机 $M_1=(\{P_1,P_2\},\{写,生产\},\delta_1,P_1)$、$M_2=(\{C_1,C_2\},\{读,消费\},\delta_2,C_1)$、$M_3=(\{0,1,2\},\{读,写\},\delta_3,0)$，显然，$\Sigma_1\cap\Sigma_3=\{读\}$，$\Sigma_2\cap\Sigma_3=\{写\}$。

依据如下三个有限状态机的同步积复合 $M=M_1\otimes M_2\otimes M_3=(Q,\Sigma,\delta,q_0)$，其中，

$Q=Q_1\times Q_2\times Q_3$；

$\Sigma=\Sigma_1\cup\Sigma_2\cup\Sigma_3$；

$q_0=(q_{10},q_{20},q_{30})\in Q_1\times Q_2\times Q_3$；

$\delta((q_1,q_2,q_3),a)=$

$$\begin{cases}(q_1',q_2',q_3') & \delta_1(q_1,a)=q_1',\delta_2(q_2,a)=q_2',\delta_3(q_3,a)=q_3',a\in\Sigma_1\cap\Sigma_2\cap\Sigma_3\\ (q_1',q_2',q_3) & \delta_1(q_1,a)=q_1',\delta_2(q_2,a)=q_2',a\in\Sigma_1\cap\Sigma_2,a\notin\Sigma_3\\ (q_1',q_2,q_3') & \delta_1(q_1,a)=q_1',\delta_3(q_3,a)=q_3',a\in\Sigma_1\cap\Sigma_3,a\notin\Sigma_2\\ (q_1,q_2',q_3') & \delta_2(q_2,a)=q_2',\delta_3(q_3,a)=q_3',a\in\Sigma_2\cap\Sigma_3,a\notin\Sigma_1\\ (q_1',q_2,q_3) & \delta_1(q_1,a)=q_1',a\in\Sigma_1,a\notin\Sigma_2,a\notin\Sigma_3\\ (q_1,q_2',q_3) & \delta_2(q_2,a)=q_2',a\in\Sigma_2,a\notin\Sigma_1,a\notin\Sigma_3\\ (q_1,q_2,q_3') & \delta_3(q_3,a)=q_3',a\in\Sigma_3,a\notin\Sigma_1,a\notin\Sigma_2\\ 无定义 & 其余\end{cases}$$

可得到整个系统完整行为的有限状态机规格，如图 2.9(d)所示。

从上面例子不难发现，有限状态机在软件系统规格方面存在以下几方面的局限性：

① 在多有限状态机系统中，系统的状态为所有状态机的状态的笛卡儿积，因此系统的状态数具有状态组合复杂性问题。即使对于非常简单的情况，复合后系统的状态图的规模也会变得非常庞大，甚至无法进行。

② 有限状态机模型实质上存在一个限制性假定：在系统所处的每一个状态上，在任何时刻，最多执行一个操作。因此，有限状态机仅能够描述顺序系统，无并发描述的能力。

③ 有限状态机不能描述无限状态系统，同时缺乏描述时间特性的机制。有限状态机也不适合于描述系统的功能和结构特性。

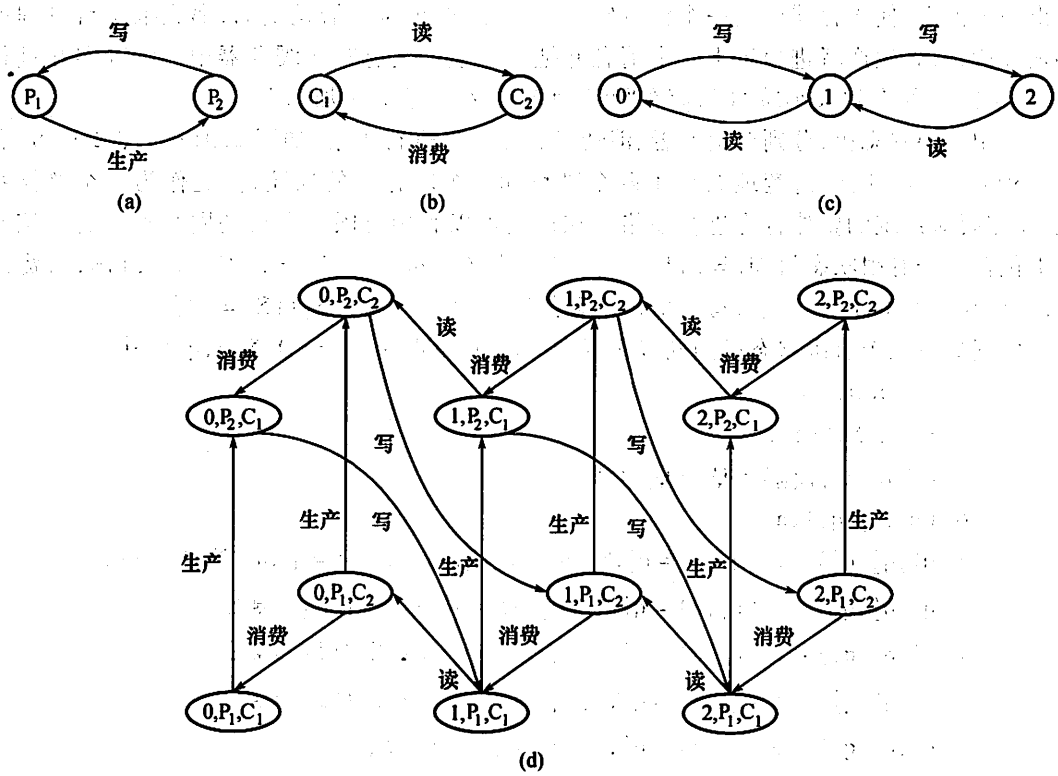


图 2.9　生产者 - 消费者系统规格

2.2　Statecharts

2.2.1　Statecharts 中的状态

Statecharts 是由以色列的 Harel 于 1987 年建立的传统有限状态机的一种扩展形式。Statecharts 中通过引入状态的递阶(层次化)、状态的“与(AND)”分解和“或(OR)”分解等高级特性,使得问题规格的状态图的状态数目极大减少,从而也具备了更强的描述能力。Statecharts 中状态用圆角方框图示,状态名标注在圆角方框内的上方(图 2.10)。

(1) 状态的层次化

传统有限状态机中的状态是分布在同一个平面层次上的,不便于大规模、复杂系统的规格。在图 2.10(a)中,有限状态机在输入 F 下均可从状态 S 或 T 转移到状态 U。这里可以将状态 S 和 T 聚合成一个新的状态,记为状态 V,并将两个状态转移弧用一个弧来代替(如图

2.10(b)所示)。状态的层次化或者递阶就是基于这一思想建立的,状态的层次化通过框图的嵌套来图示。

在图 2.10(c)中,称 S 为高级状态或超状态,S 也称为 S1 与 S2 的父状态,S1 和 S2 则称为 S 的子状态。在实际软件系统规格中,可根据问题的需要建立状态的多层次描述。在图 2.10(d)中,S 的子状态是 S1 和 S2,S2 的子状态是 S21 和 S22。相应地,状态 S1 和 S2 的父状态为 S,状态 S2 为状态 S21 和状态 S22 的父状态。状态 S 又称为状态 S21 和状态 S22 的先辈状态,状态 S21 和状态 S22 称为状态 S 的后代状态。具有子状态的状态又称为递阶状态,没有子状态的状态称为简单状态或者原子状态。

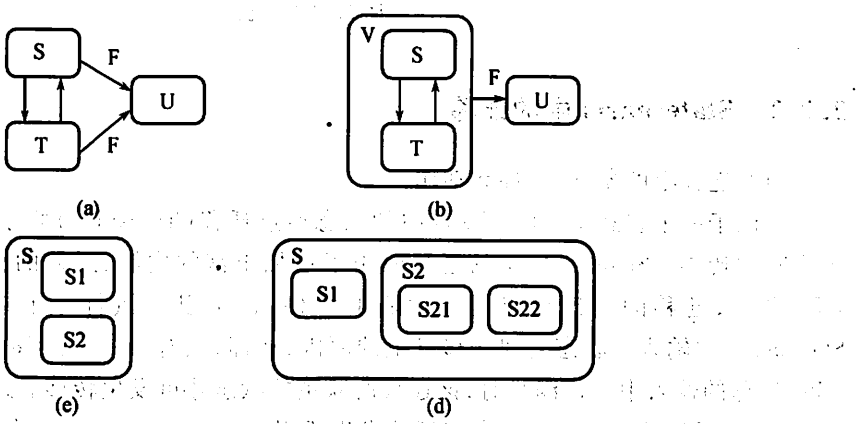


图 2.10 状态的层次化

(2)“或(OR)”状态和“与(AND)”状态

状态的层次化可通过自顶向下(top-down)的状态分解,或者自底向上(bottom-up)的状态合并来实现。在 Statecharts 中,状态的子状态具有“或(OR)”和“与(AND)”两种形式,并分别称该状态为“或(OR)”状态和“与(AND)”状态。

某一状态是或状态,当且仅当位于该状态就是位于其中的一个子状态,亦即,位于一个或状态不能同时位于该状态的两个以上的子状态。事实上,或状态表示了子状态之间的一种异或关系。状态的或状态表示,称为状态的或分解。如上图 2.10(b)所示,状态 V 为一或状态,位于状态 V 就意味着位于状态 S 或者状态 T。同理,图 2.10(c)和 2.10(d)所示状态 S 和 S2 也是或状态。

某一状态是与状态,当且仅当位于该状态就是同时位于其所有的子状态。事实上,与状态表示了子状态之间的一种与关系。状态的与状态表示,称为状态的正交分解;与状态的子状态,又称为状态的正交分量。图形表示中,用虚线将与状态的正交分量或子状态进行隔离,与状态的标识符(即状态名)标注在和状态边框连在一体的小方框内。如图 2.11 所示,与状态 U 的正交分量为状态 S 和状态 T,而状态 S 和状态 T 分别是子状态为 S1、S2 和 S2,T1

和 T1 的或状态。

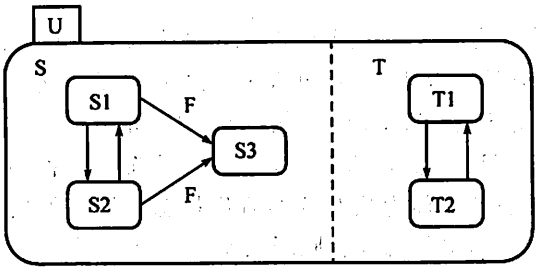


图 2.11　与状态

2.2.2　Statecharts 中的迁移

(1) 迁移上的条件、事件和动作

类似于有限状态机，Statecharts 中的状态通过迁移(Transition)联系起来，并最终实现状态之间的转移。Statecharts 中的迁移赋予了更加丰富的内涵，迁移的输入可以是事件或者(和)条件，迁移的输出可以是产生新的事件，也可以是执行一个过程或(和)函数等。Statecharts 中输入是状态之间迁移发生的原因，亦即，只有当一个迁移的输入事件出现或者(和)迁移的输入中条件满足时，该迁移所联系的状态才可发生转移。迁移的输入，称为触发(Trigger)；迁移的输出是状态之间迁移发生所引起的动作(Action)，动作可以是产生新的事件或执行一个过程或函数。动作的执行被认为是瞬时的。触发和动作以“事件[条件]/动作”的形式标注在迁移弧旁(如图 2.12 所示)。

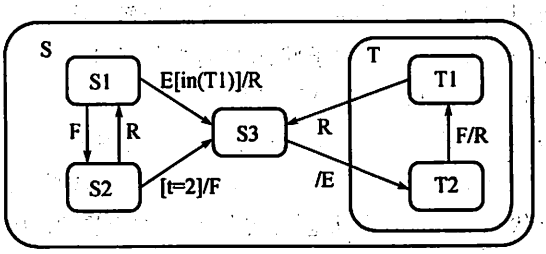


图 2.12　迁移的触发和动作

在 Statecharts 中，除了通常意义下的事件、条件与动作外，还预定义了一些与状态、数据项、时间有关的条件与事件。如：

条件：

in(S)　表示是否处于状态 S 中，如是则为真，否则为假；

ac(A)　表示活动 A 是否处于活动状态，如是则为真，否则为假；

hg(A) 表示活动 A 是否处于挂起状态,如是则为真,否则为假;

事件:

en(S) 表示进入状态 S;

ex(S) 表示退出状态 S;

ns 表示正在进入当前状态;

xs 表示正在退出当前状态;

st(A) 表示活动 A 被启动;

st 表示当前活动被启动;

sp(A) 表示活动 A 被停止;

tr(C) 表示条件 C 变为真;

fs(C) 表示条件 C 变为假;

rd(X) 表示 X 被动作 rd! 读;

wr(X) 表示 X 被动作 wr! 写;

ch(X) 表示数据项 X 的值发生改变;

tm(E,N) 表示事件 E 发生后,已过了 N 个时间单位。

动作:

tr!(C) 表示对条件 C 赋值真;

fs!(C) 表示对条件 C 赋值假;

st!(A) 表示启动活动 A;

sp!(A) 表示停止活动 A;

sd!(A) 表示挂起活动 A;

rs!(A) 表示重新开始活动 A;

rd!(X) 表示读数据或条件 X;

wr!(X) 表示写数据或条件 X;

hc!(S) 表示忘记状态 S 的历史信息;

hd!(S) 表示忘记状态 S 的后继状态的历史信息。

此外,这些事件、条件和动作还可进行逻辑 and、or、not 等运算,例如,E1 and E2 表示事件 E1 与 E2 同时发生;C_1 or C_2 表示 C_1 和 C_2 的或逻辑运算。

在图 2.12 中,在输入事件 F 下,状态 S1 可转移到状态 S2;在输入事件 R 下,状态 S2 可转移到状态 S1;在输入事件 E 且条件 in(T1)成立(即位于状态 T1)下,状态 S1 转移到状态 S3 并输出事件 R;在条件 t=2 成立下,状态 S2 转移到状态 S3 并输出事件 F;在输入事件 R 下,状态 T1 可转移到状态 S3;在输入事件 F 下,状态 T2 可转移到状态 T1 并输出事件 R;状态 S3 可直接转移到状态 T2,并输出事件 E。

(2) 迁移与状态的连接

迁移描述了简单状态和简单状态、简单状态和递阶状态、递阶状态和递阶状态之间的状态转移关系。类似于有限状态机,迁移用有向弧图示。有向弧可以从一个状态的边界引出,终止于另一状态的边界(图2.13(a)中迁移E1和迁移E2),也可以穿过先辈状态的边界(图2.13(c)中的迁移F和图2.13(d)中的迁移G)。

当一个迁移没有指明其源状态时,则称之为缺省迁移。该迁移所指向的状态,称为缺省状态。缺省迁移用圆点箭线弧表示,如图2.13(a)所示。当从OFF状态进入ON状态时,因为没有明确指明其进入的子状态,这时缺省状态起作用,即进入子状态S2;状态OFF则是整个Statecharts的缺省状态。缺省迁移可以指向一个状态,也可以穿过先辈状态的边框直接指向某一子状态,如图2.13(b)所示。

当迁移指向一个与状态时,则相当于指向其所有的子状态;当迁移指向一个或状态时,则相当于指向其缺省的子状态。当退出一个状态时,则相当于退出其所有的子状态。图2.13(a)所示,当通过迁移E2退出ON状态时,即是退出ON的所有子状态,当通过迁移E1进入ON状态时,即是进入ON的缺省子状态S2。

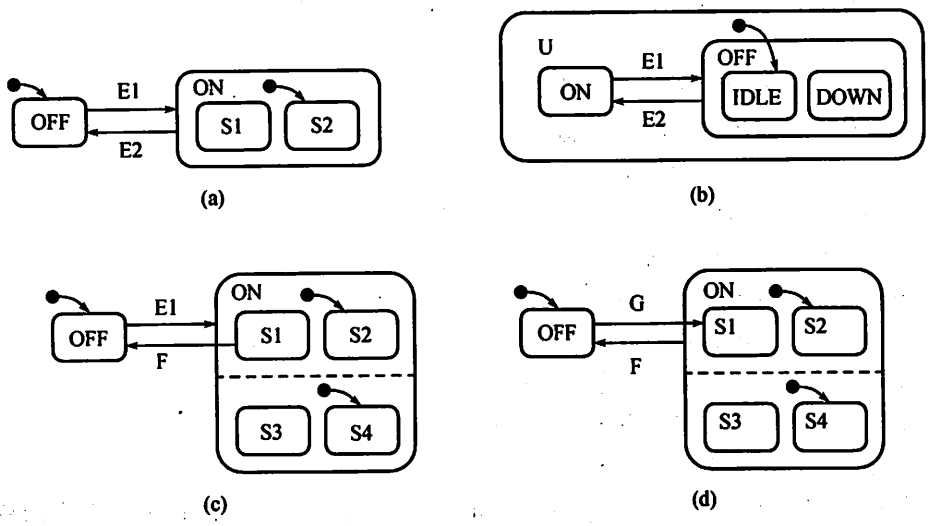


图2.13　迁移与状态的连接

(3) 复合迁移

为了简化图形中迁移的有向弧线,Statecharts中引入了一些辅助符号,从而得到状态之间转移关系的复合迁移。

条件连接符　条件连接符也可称为C-连接符,用于条件的连接。如图2.14(a)所示为采用了C-连接符的复合迁移,其等价形式如图2.14(b)所示。Statecharts规定,始于C-连接符的条件分支可以多个,但是这些条件分支必须是异或的。

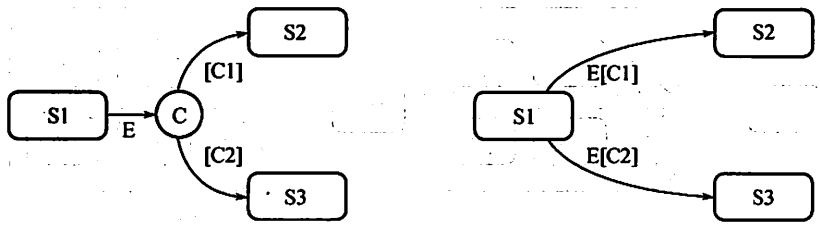


图 2.14 条件连接符

开关连接符 开关连接符也可称为 S－连接符，用于事件的连接。如图 2.15 所示是采用了 S－连接符的复合迁移。图中表示了：在事件 A 和事件 E1 发生下，状态 S 转移到状态 T1；在事件 A 和 E2 发生下，状态 S 转移到状态 T2；在事件 A 和事件 E3 发生下，状态 S 转移到自身。

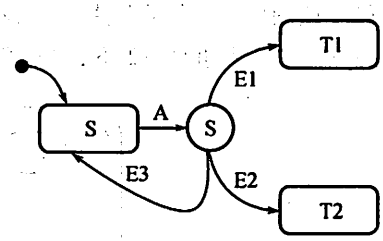


图 2.15 开关连接符

分叉(汇合)连接符 具有不同触发和动作的迁移，可以通过分叉(汇合)连接符形成一个单一的弧线进入(离开)连接符。图 2.16(a)、(b)所示分别为由分叉、汇合连接符形成的复合迁移。图 2.16(a)中，在输入事件 R 下，状态 S1 转移到状态 S2，并输出事件 A；在输入事件 S 下，状态 S1 转移到状态 S3，并输出事件 A。图 2.16(b)中，在输入事件 SET 下，状态 S2 转移到状态 S1；状态 S3 转移到状态 S1。

分叉(汇合)结构 分叉(汇合)结构是具有相同触发和动作迁移的一种简化描述方式。图 2.17(a)、(b)所示分别为由分叉、汇合结构形成的复合迁移。此类结构下复合迁移所联系的状态转移，要求迁移上所有的触发同时满足。图 2.17(a)中，只有在输入事件 R、E1 和 E2 下，状态 T 转移到状态 S1 和状态 S3，并输出事件 A1 和事件 A2；图 2.17(b)中，在输入事件 E 下，状态 S1 和状态 S3 转移到状态 U。

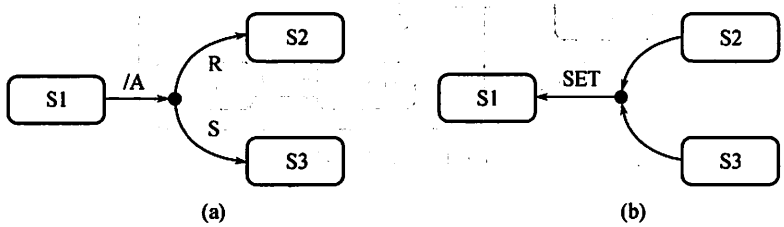


图 2.16 分叉(汇合)连接符

图形连接符 图形连接符可以实现迁移的异地连接。图形连接符，用椭圆符号并标注以相同的名称来图示。Statecharts 规定图形连接符出现相连接的迁移必须具有相同方向的

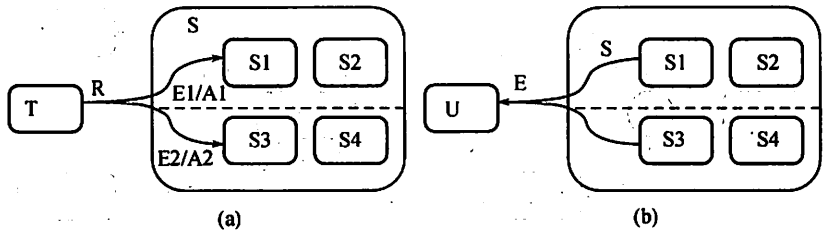


图2.17　分叉(汇合)结构

弧线(离开或者进入)。图2.18中,有三个标以INT的椭圆,由该图形连接符构成了符合迁移。该图中,在输入事件A下,状态S2转移到状态S1;在输入事件E1下,状态S3转移到状态S1;在输入事件E2下,状态S3转移到状态S1。

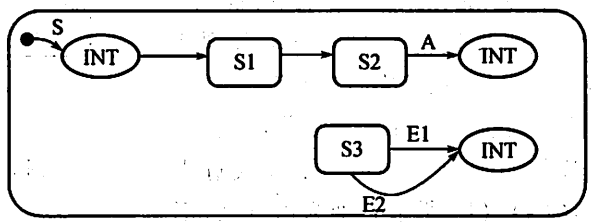


图2.18　图形连接符

历史连接符　历史连接符用以规定系统进入一个状态组中的一个最近访问历史状态。历史连接符用标注以H的圆圈图示。事实上,历史连接符可以理解为一个特殊的状态。进入历史连接符就表示进入该连接符所在状态组中最近访问的那个状态,如果访问历史状态为空,则进入历史连接符所指向的状态。如图2.19所示,从OFF状态进入CON状态时,首先进入其历史状态H,即进入最近访问的那个状态,如果H为空,则进入IDLE状态。

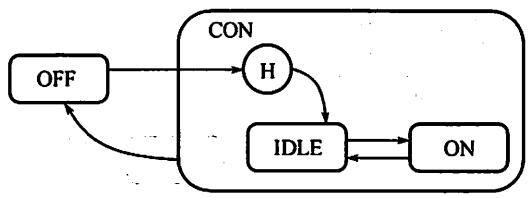


图2.19　历史连接符

2.2.3　电梯控制系统

考虑一个m层、n个电梯的大楼的服务系统。需要建立一个用以描述电梯在楼层之间的运动进而完成电梯控制系统开发的规格,系统存在如下一些约束和要求:

① 每个电梯内部都有一组按钮，每个按钮用于指示一个楼层。当按下某个按钮后，该按钮指示灯就会发亮，直到到达相应的楼层后按钮的指示灯自动熄灭。

② 除了最高层和最低层以外，每个楼层都有两个按钮：一个用于请求电梯向下移动，一个用于请求电梯向上移动。这些按钮在被按下后，相应的指示灯就会发亮。当电梯按照所请求的方向移动到该楼层时，按钮的指示灯自动熄灭。

③ 在没有服务请求时，电梯停留在其当前所处的楼层，并且电梯门关闭。

考虑电梯正常运行情况，假定电梯在处于两楼层之间时不会突然停下来，并且也不会改变移动方向。下面采用自底向上的方式来建立该电梯系统的 Statecharts 规格。整个电梯控制系统可分为电梯控制按钮、楼层控制按钮和电梯运动系统共三个部分。

电梯控制按钮 用以描述电梯按钮、按钮的状态以及状态转换的符号定义如下：

eb_{ij} 电梯 i 中对应于楼层 j 的按钮；

$ebon_{ij}$ eb_{ij} 的指示灯为亮的状态；

$eboff_{ij}$ eb_{ij} 的指示灯为熄灭的状态；

press-eb_{ij} eb_{ij} 被按下时所产生的事件；

arrives-at_{ij} 电梯 i 到达楼层 j 时所产生的事件。

图 2.20 所示为电梯按钮 $eb_{i,j}$ 的状态转换图。

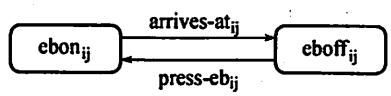


图 2.20 电梯控制按钮 eb_{ij} 的状态转移图

楼层控制按钮 除了最低层和最高层以外，其他各楼层都有两个按钮：一个用于请求电梯向上运动，另一个用于请求电梯向下运动。用以描述的相应符号为：

fb_{bj} 楼层 j 对应于运动方向 b 的按钮；

$fbon_{bj}$ fb_{bj} 的指示灯为亮的状态；

$fboff_{bj}$ fb_{bj} 的指示灯为熄灭的状态；

call-fb_{bj} 在 fb_{bj} 被按下时所产生的事件；

depart-at_{bj} 在方向 b 上运动的电梯离开楼层 j 时所产生的事件。

图 2.21 所示为楼层按钮 fb_{bj} 的状态转换图。最低层和最高层按钮的状态转换图类似于图 2.21，只不过其中的运动方向 b 仅有惟一选择（最高层按钮的方向向下、最低层按钮的方向向上）。

电梯运动系统 为了建立电梯运动系统的规格，现就某个楼层 j 所能观察到的电梯运动进行分析，图 2.22 给出了所有可能的几种情形。对其说明如下：

① 电梯一直向上移动；

② 电梯一直向下移动；

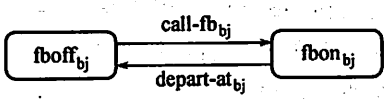


图 2.21　楼层按钮 $fb_{b,j}$的状态转换图

③ 电梯从下面往上逐渐靠近,停止,然后继续向上运动;
④ 电梯从上面往下逐渐靠近,停止,然后继续向下运动;
⑤ 电梯从下面往上逐渐靠近,停止,然后反向向下运动;
⑥ 电梯从上面往下逐渐靠近,停止,然后反向向上运动;
⑦ 电梯从下面往上逐渐靠近,停止,然后在空闲状态等待;
⑧ 电梯从上面往下逐渐靠近,停止,然后在空闲状态等待;
⑨ 从空闲状态起动,然后向下运动;
⑩ 从空闲状态起动,然后向上运动。

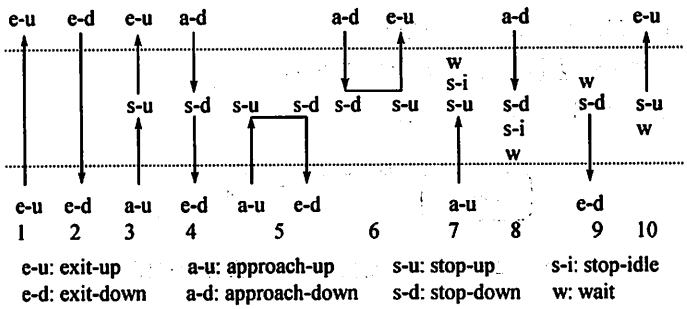


图 2.22　电梯运动系统分析

对于处在某个楼层的观察者来说,电梯可能会处于下列 8 种状态之一:
① 等待(wait):在该层等待,并且门未打开;
② 停止 - 空闲(stop-idle):停在该层,并且门打开;
③ 停止 - 向上(stop-up):电梯在向上运动过程中在该楼层暂停;
④ 停止 - 向下(stop-down):电梯在向下运动过程中在该楼层暂停;
⑤ 靠近 - 向上(approach-up):电梯在向上运动过程中逐渐靠近该楼层;
⑥ 靠近 - 向下(approach-down):电梯在向下运动过程中逐渐靠近该楼层;
⑦ 退出 - 向上(exit-up):电梯离开该楼层,继续向上运动;
⑧ 退出 - 向下(exit-down):电梯离开该楼层,继续向下运动。

对电梯运动过程进行如下假定:当电梯停留在某楼层并且没有请求需要处理时,它的门将保持打开状态。电梯在向上运动过程中,在进入 stop-up 或 stop-idle 状态之前要经过状态 approach-up;同理,电梯在向下运动过程中,在进入 stop-down 或 stop-idle 状态之前要经过状态 approach-down。这一转换过程,在实际电梯系统运行中需要对电梯进行减速,可以用

reduce-speed 来表示相应的事件。

可将状态 stop-up、stop-down 和 stop-idle 进行合并得到一个超状态 stop；将状态 approach-up 和 approach-down 进行合并得到超状态 approach；将状态 exit-up 和 exit-down 合并得到超状态 exit。电梯经过在某楼层的暂停后将合上门，然后离开该楼层，这一过程规格为状态 stop 在触发事件 close-door 下转移到状态 exit。当存在某楼层对电梯的请求，并且此时的电梯正处于 wait 状态时，电梯将离开空闲状态，这一过程规格为状态 wait 在触发事件 call-fb 下转移到状态 stop(如图 2.23 所示)。

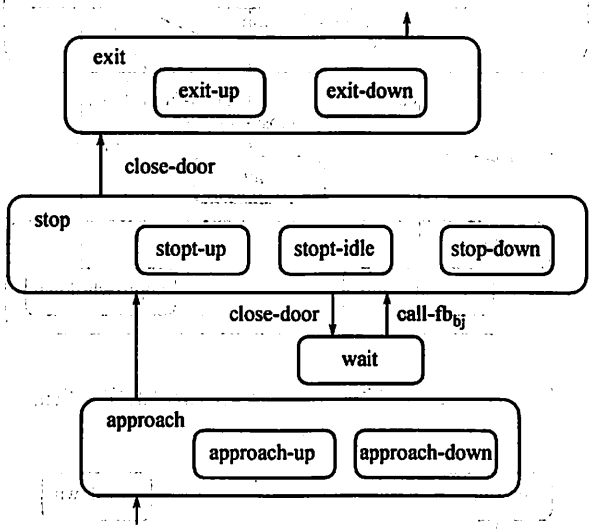


图 2.23 某楼层观测到的电梯运动情况规格

图 2.23 中的规格是不够详尽和准确的，因为基于 Statecharts 的语义，图中从超状态 approach 到超状态 stop，或者从超状态 stop 到超状态 exit 的转换是指从 approach 或者 stop 中的某个子状态转换到 stop 或者 exit 中的某个子状态。为此，需要通过标记出与所有子状态相关的迁移，对该规格进行细化。

如图 2.24 所示，当电梯处于 stop-idle 状态时，如果发生某个用户对电梯进行请求，则根据 call-fb$_{bj}$(call-from-up 或者 call-from-down)状态转移到 stop-up 或 stop-down；如果没有使用请求，则电梯将进入 wait 状态。图 2.24 中还通过触发 called-from-up、called-from-down、change-direction 和 no-request 等将状态 stop 的子状态 stop-up、stop-down 与状态 wait 联系了起来。此外，还可以对状态 stop-up 和 stop-down 进行进一步细化，以描述电梯的开关门情况(如图 2.25 所示)。

将在不同楼层所观测到的状态转移图复合起来可以得到一个服务于多个楼层的电梯的状态转移图。设 F_j 为在第 j 层所观察到的该电梯的状态机。则从 F_j 的 exit-up 状态到 F_{j+1} 的 exit-up 状态的转移对应于电梯在向上运动的过程中不在第(j+1)层停留的情况。而从

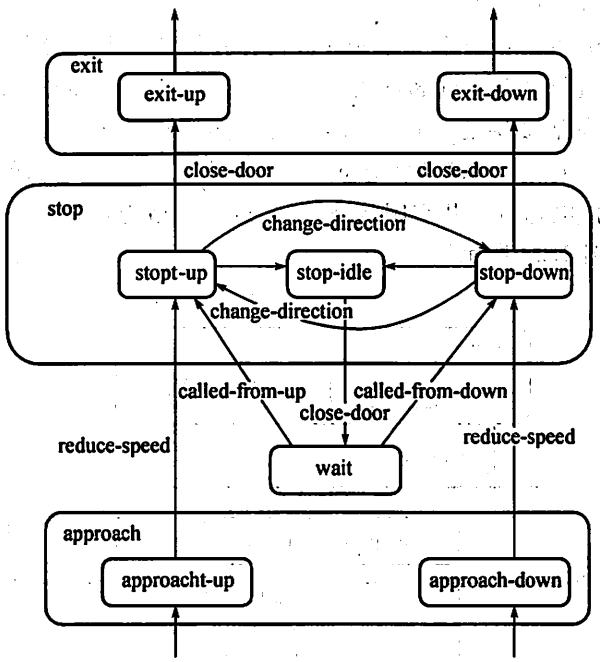


图 2.24　某楼层观测到的电梯运动情况的完全规格

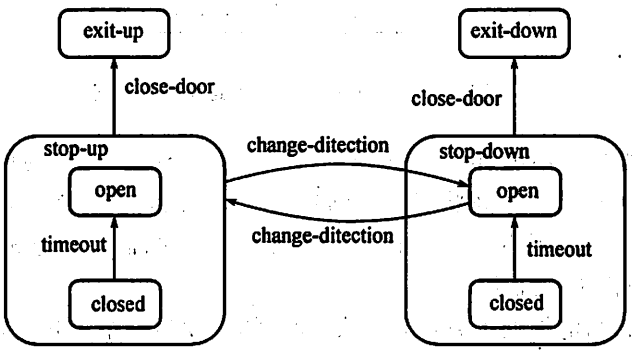


图 2.25　状态 stop-up 和 stop-down 的细化

F_j 的 exit-up 状态到 F_{j+1} 的 approach-up 状态的转移对应于电梯在向上运动的过程中在第(j+1)层停留的情况。类似的状态转移也存在于 F_j 的 exit-down 状态与 F_{j-1} 的 exit-down 及 approach-down 状态之间。但最顶层和最底层所观察到的有限状态机稍有不同:相应于最顶层的有限状态机中不存在 exit-up 状态和 approach-down 状态;同样,相应于最底层的有限状态机中不存在 exit-down 状态和 approach-up 状态。图 2.26 给出了一个 4 个楼层、1 个服务电梯系统的 Statecharts 规格。

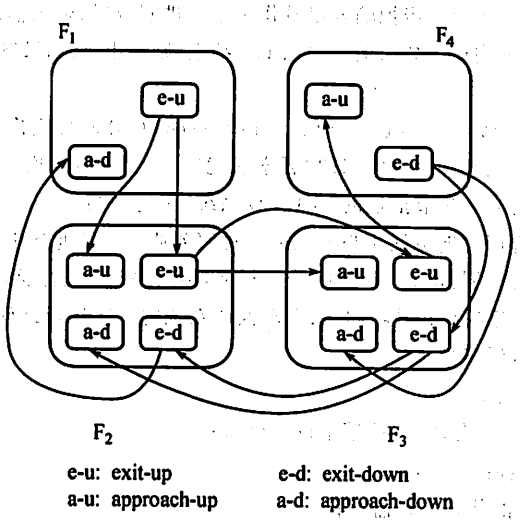


图 2.26 具有 4 个楼层和 1 个电梯的电梯运动系统规约

对于具有 n 个电梯服务的系统，所有电梯运行的 Statecharts 规格，可以使用有限状态机的同步积 $E = E_1 \otimes E_2 \otimes \cdots \otimes E_i \otimes \cdots \otimes E_n$ 来描述，其中 E_i 是电梯 i 的 Statecharts 规格。n 个电梯服务系统需要使用一个控制器来进行请求队列管理，从而使得多个电梯之间能够协调、高效地运行（该请求队列管理控制器的规格从略）。

习　题

2.1　对于如下图 2.27 和 2.28 所示有限状态机的状态转移图，分别给出其关系矩阵和状态转移表。

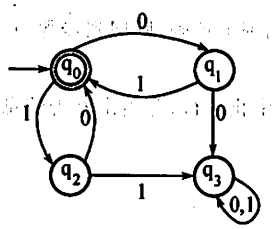


图 2.27　状态转移图之一

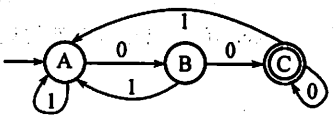


图 2.28　状态转移图之二

2.2　对于题 2.1 的有限状态机，分别给出其所接受的语言。

2.3　构造有限状态机，使其接受的语言为由 0 和 1 构成的字符串的集合，并且分别满足：

(1) 每个字符串以 00 结束；

(2) 每个字符串中有 3 个连续的 0 出现。

2.4　分别构造 Moore 机和 Mealy 机，使其满足：

(1) 输入字符串由0、1、2构成。当把输入串看作一个三进制数时,输出输入串的模5余数。

(2) 输入字符串由0和1构成。如果输入串以111结束,则输出A;如果输入串以100结束,则输出B;其他情况输出C。

2.5　试对有限状态机和Statecharts进行比较,阐述Statecharts在软件规格方面体现的优点。

2.6　利用有限状态机对具有3个哲学家的"哲学家就餐问题"进行规格。

2.7　利用有限状态机对4.2.2节描述的"AB协议系统"进行规格。

2.8　自动回叫是电信部门可向电话用户提供的一种服务业务。该业务中,在被呼叫方正忙的情况下,系统继续保持对被呼叫方状态的检测;一旦被呼叫方出现空闲状态,系统将通知呼叫方,并自动对被呼叫方进行呼叫。用A、B分别表示呼叫端和被呼叫端,自动回叫系统的一次运行过程如下:

(1) A呼叫B,但B的电话正忙;

(2) A按下"自动回叫"键;

(3) 交换系统代理对A的"自动回叫"请求进行确认;

(4) A听到确认提示,然后放下电话;

(5) 交换系统代理对B的状态进行监控;

(6) 当B回到空闲状态时,交换系统代理向A发出特殊响铃提示;

(7) A端的电话进行特殊的响铃提示;

(8) A拿起听筒;

(9) 系统自动地再次向B发出呼叫;

(10) B拿起听筒;

(11) A与B进行通话。

试利用有限状态机分别对自动回叫系统中的呼叫端、被呼叫端、和交换系统代理进行规格。

2.9　用户身份鉴定是许多数据库系统提供的一种安全保护措施,系统根据用户名和相应的口令来进行用户身份鉴定,只有合法的用户才准许使用系统。用户身份鉴定过程如下:

(1) 提示用户输入用户名;

(2) 如果系统不能识别用户名,则提示用户再次输入,直到输入一个有效的用户名;

(3) 提示用户输入密码;

(4) 如果密码不正确,则提示用户第二次输入;如果两次输入密码都不正确,则回到步骤(1);

(5) 如果密码正确,则提示用户输入其用户权限;

(6) 如果所输入的权限级别比赋予该用户的权限级别高,则系统中止执行,并给出警告信息。

试利用Statecharts给出相应软件的规格。