第3章 迁移系统

本章学习目标

(1) 掌握迁移系统的基本概念及其应用。

(2) 掌握迁移系统到迁移图的转换。

使用形式化方法,通常需要首先对被检测的(并发)系统进行形式化建模。由于并发系统比串行系统表现出更为复杂的并发行为,其形式化模型必须能很好地表达并发性。根据并发行为的方式,并发系统的计算模型可分为交错[①](interleaving,也称交替)模型和非交错(non-interleaving)/独立(independent)模型两大类。其中交错模型是通过各个原子迁移以不确定的顺序交错执行来表示并发行为的,其计算行为表现为状态迁移序列,不允许两个并行进程在相同的时间点上执行它们各自的语句,要求其中一个进程执行时,其余进程处于非激活状态,不允许干扰(interference)情形出现。

迁移系统(transition system)是 R. M. Keller 最初于 1976 年提出的一种基于交错并发执行方式的计算模型,也是描述计算机软硬件系统行为的基本抽象模型。迁移系统可以对处于不同情况下的并发系统建模,如可以是在进程完全自动运行的简单情况下,也可以是在进程间以某种方式通信的更现实的环境中。本章主要介绍迁移系统及相关概念,简要阐述迁移图的概念和其到迁移系统的转换,并通过实例说明迁移系统的具体应用。

3.1 基本概念

在迁移系统中,一个迁移表示系统的一个原子操作,系统可以处于有限或无限数量的状态中的某一个;在每个状态上,系统可以执行一系列原子迁移中的一个,这一系列迁移就是该状态可行(使能)的迁移,其他迁移是不可行(非使能)的;在每个状态中选择一个可行的状态,将系统迁移为一个新的状态,只要至少有一个可行状态,则该过程不断继续。

3.1.1 形式定义

状态用于描述系统在某个时刻的行为信息,例如,一个交通灯的一个状态表示了灯当前的颜色。类似地,一个串行程序的一个状态表示了所有程序变量当前的值和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)。在一个时序电路中,一个状态表示了寄存器当前的值和输入位的值。

---

① "交错"一词最初由 E. W. Dijkstra 于 1971 年提出。

迁移表示系统如何从一个状态转换为另一个状态。在交通灯的例子中,迁移可以表示交通灯从一种颜色转换到另一种颜色。对串行程序来说,一个迁移对应一个语句的执行,也可以包含一些变量和程序计数器值的改变。在时序电路中,一个迁移是指寄存器值的改变和在一组新的输入下输出位的改变。

迁移系统包含迁移(状态的改变)的动作名称和状态的原子命题。动作名称将被用来描述进程间的通信机制,原子命题用来形式化在某状态下的一些特性,直观地表示了关于系统状态的一些简单的已知事实。例如,对于给定的整型变量 x,"x 等于 0"或"x 小于 200"都是原子命题。

定义 3.1 一个迁移系统 TS 是一个六元组[①](S,T,→,I,AP,L),其中:

$S=\{s_0,s_1,s_2,\cdots,s_n,\cdots\}$表示状态集;

$T=\{\tau_0,\tau_1,\tau_2,\cdots\}$表示迁移动作集;

$\rightarrow\subseteq S\times T\times S$ 表示迁移关系,如$(s,\tau,s')\in\rightarrow$,也可用 $s\xrightarrow{\tau}s'$或$\rho_\tau(s,s')$表示;

$I\subseteq S$ 表示初始状态集;

$\mathrm{AP}=\{a,b,c,\cdots\}$表示原子命题集;

$L:S\rightarrow 2^{\mathrm{AP}}$ 称为标签函数。

如果 S、T 和 AP 都是有穷集,那么称 TS 是有穷迁移系统。

一个迁移系统的直观行为描述如下:系统始于一些初始状态 $s_0\in I$,通过迁移关系→发生状态转变。也就是说,如果 s 是当前状态,那么源于 s 的迁移 $s\xrightarrow{\tau}s'$被不确定地选择并且执行,即执行迁移动作 τ,并且系统从状态 s 迁移到状态 s'。这个选择过程会在状态 s'重复并且终止于一个没有出迁移的状态(需要注意的是,I 可以为空,在这种情形下,迁移系统由于没有初始状态可以选择,不产生任何行为)。更重要的是,当一个状态有多个出迁移时,下一个迁移的选择完全是不确定的,也就是这个选择过程的结果是不可以推理得到的,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,起始状态的选择也是不确定的。

标签函数 L 将原子命题的一个集合 $L(s)\in 2^{\mathrm{AP}}$ 与状态 s 联系起来。2^{AP} 是 AP 的幂集。$L(s)=\{a\mid a\in\mathrm{AP}$ 且状态 s 完全满足 $a\}$,假设 φ 是一个命题逻辑公式,如果 $L(s)$使公式 φ 为真,那么可以推导 s 满足公式 φ,则可以表示为

$$s\vDash\varphi \quad \text{iff} \quad L(s)\vDash\varphi$$

定义 3.2 一个迁移系统 TS=(S,T,→,I,AP,L),对于 $s\in S$ 和 $\tau\in T$:

(1) s 的直接 τ-后继集合定义为 $\mathrm{Post}(s,\tau)=\{s'\in S\mid s\xrightarrow{\tau}s'\}$,$\mathrm{Post}(s)=\bigcup_{\tau\in T}\mathrm{Post}(s,\tau)$,每个状态 $s'\in\mathrm{Post}(s,\tau)$ 被称作 s 的一个直接 τ-后继。直接后继集合的概念由以下方式扩展为 S 的子集,对于 $C\subseteq S$,有 $\mathrm{Post}(C,\tau)=\bigcup_{s\in C}\mathrm{Post}(s,\tau)$,$\mathrm{Post}(C)=\bigcup_{s\in C}\mathrm{Post}(s)$。

(2) s 的 τ-前趋集合定义为 $\mathrm{Pre}(s,\tau)=\{s'\in S\mid s'\xrightarrow{\tau}s\}$,$\mathrm{Pre}(s)=\bigcup_{\tau\in T}\mathrm{Pre}(s,\tau)$。

$\mathrm{Pre}(C,\tau)$ 和 $\mathrm{Pre}(C)$的概念以类似的方式定义:

① 在许多文献中,迁移系统采用 Kripke 三元组(S,R,L)表示,有关 Kripke 内容介绍见第 6 章。

$$\mathrm{Pre}(C,\tau)=\bigcup_{s\in C}\mathrm{Pre}(s,\tau),\quad \mathrm{Pre}(C)=\bigcup_{S\in C}\mathrm{Pre}(s)$$

定义 3.3 对于迁移动作集 T 中的一个迁移动作 τ 和状态集 S 中的一个状态 s：

(1) 如果 $\mathrm{Post}(s,\tau)\neq\varnothing$，则用 enabled($\tau$)(简记 En($\tau$))表示迁移 τ 在状态 s 是使能(能行)的，也就是说，s 有一个直接 τ-后继。

(2) 如果 $\mathrm{Post}(s,\tau)=\varnothing$，则称迁移 τ 在状态 s 是非使能的(disenabled)，也就是说，s 没有一个直接 τ-后继。

定义 3.4 对于一组迁移 $T_1\subseteq T$ 和状态集 S 中的一个状态 s：

(1) 如果 T_1 中有 τ 在 s 上是使能的，则称 T_1 在 s 是使能的。

(2) 如果 T_1 中所有 τ 在 s 上都是非使能的，则称 T_1 在 s 是非使能的。

定义 3.5 (1) 如果对每个状态 $s\in S$，都有 $\mathrm{Post}(s,\tau)=\{s\}$，则称 τ 为空迁移；除了空迁移之外的迁移都叫作勤勉(diligent)迁移。

(2) 如果 s 上仅有的能行的迁移是空迁移 τ_I，那么状态 s 是终止的。

一个迁移系统 TS 的终止状态是那些没有任何出迁移(即仅有空迁移)的状态，一旦由 TS 描述的系统到达了一个终止状态，整个系统将会停止。对于串行(顺序)程序而言，终止状态出现表示程序终止。

上面提到非确定性对于系统建模是非常重要的，但是迁移系统的“可见”行为是确定的，也常常是很有用的。一般有两种方法刻画一个迁移系统的可见行为：一种依靠动作；另一种依靠状态的标签。以动作为基础的方法从外部只看到执行的动作，以状态为基础的方法忽略了动作，并且要求约束当前状态的原子命题是可见的。从以动作为基础的方法的观点来看，迁移系统是确定的就要使每个状态都至多有一个标记动作 τ 的出迁移，然而从状态标签的观点出发，确定性意味着对于任何状态标签 $A\in 2^{\mathrm{AP}}$ 和任何状态来说，至多有一个出迁移指向一个标签为 A 的状态。在这两种情况下，都要满足至多有一个初始状态。

定义 3.6 一个迁移系统 $TS=(S,T,\rightarrow,I,\mathrm{AP},L)$。

(1) 如果对于所有状态 s 和动作 τ，$|I|\leqslant 1$ 和 $|\mathrm{Post}(s,\tau)|\leqslant 1$，则称 TS 是动作-确定的。

(2) 如果对于所有状态 s 和状态标签 $A\in 2^{\mathrm{AP}}$，$|I|\leqslant 1$ 和 $|\mathrm{Post}(s)\cap\{s'\in S\mid L(s')=A\}|\leqslant 1$，则称 TS 是 AP-确定的。

3.1.2 迁移图

迁移图是 L. Lamport 于 1983 年提出的一种表示并发程序的图形化建模方法。与流程图类似，迁移图是一个带结点和有向边的有向图。不同的是，流程图用结点表示迁移，而迁移图是用有向边表示迁移的。

设 $P_1,P_2,\cdots,P_m\ (m\geqslant 1)$ 是 m 个可并发执行的进程，每一进程 P_i 是带有标号(位置)结点的迁移图，勤勉迁移对应进程中出现的有标记的边。标号结点集 $L_i=\{l_0,l_1,\cdots,l_t\}$，这里 l_i 是互不相交的，引入控制变量 $\pi_1,\pi_2,\cdots,\pi_m$，其中 π_i 表明进程 P_i 控制的当前位置。

设 α 是连接进程 P_i 中位置 l 到位置 l' 的一条边，用指令 $c\rightarrow[\bar{y}:=\bar{e}]$ 标记(见图 3-1)，这里 $\bar{y}=(y_1,y_2,\cdots,y_n)$ 是各进程共享的程序变量，每个进程都可以引用或修改这些变量。则与 α 关联的迁

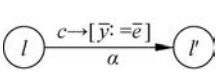

图 3-1 迁移图

移 τ 定义为 $(\pi_i = l) \wedge c \wedge (\pi_i' = l') \wedge (\bar{y} = \bar{e})$。如果在状态 s 下，P_i 当前的位置是 l，并且布尔表达式 c 为真，那么称迁移 τ 在状态 s 是使能的。如果对于一些属于进程 P_i 的边 α 来说，和 α 相关的迁移在状态 s 是使能的，则称进程 P_i 在状态 s 是使能的，否则称进程 P_i 在状态 s 是非使能的。

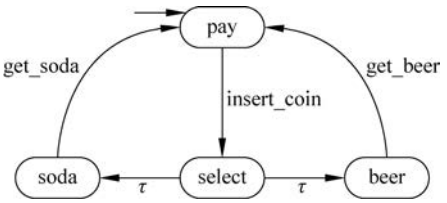

图 3-2　一个简单的饮料自动售货机的迁移图

例 3.1　图 3-2 的迁移图是对一个简单的饮料自动售货机的建模，自动售货机可以卖啤酒或苏打水。状态用圆角矩形表示，迁移用带标记的边表示，状态的名称写在圆角矩形里面，初始状态用一个没有来源的进入箭头表示。

状态集 $S = \{\text{pay}, \text{select}, \text{soda}, \text{beer}\}$，初始状态集仅有一个状态，即 $I = \{\text{pay}\}$，对于饮料机的一些内部动作，用动作 τ 表示。动作集 $T = \{\text{insert\_coin}, \text{get\_soda}, \text{get\_beer}, \tau\}$，如 $\text{pay} \xrightarrow{\text{insert\_coin}} \text{select}$ 和 $\text{beer} \xrightarrow{\text{get\_beer}} \text{pay}$ 就是一些迁移的例子。

需要注意的是，投了一个硬币后，自动售货机不能确定提供啤酒还是苏打水。

迁移系统中的原子命题以待考虑的属性而定，有一个简单的办法是让状态名作为原子命题，也就是对于任何状态 s，$L(s) = \{s\}$。然而，如果仅有的相关属性指的不是选择的饮料，如属性“在投了一个硬币后，自动售货机只递送一种饮料”，那么它就可以使用两个元素的命题集合 $\text{AP} = \{\text{paid}, \text{drink}\}$，伴随标签函数：

$$L(\text{pay}) = \varnothing, \quad L(\text{soda}) = L(\text{beer}) = \{\text{paid}, \text{drink}\}, \quad L(\text{select}) = \{\text{paid}\}$$

这里原子命题 paid 表示那些使用者已经付费，但还没有获得饮料。

前面的例子说明了有关原子命题和动作名称选择的任意性，即使一个迁移系统的形式化定义需要确定动作集 T 和命题集合 AP，T 和 AP 也可以在之后进行临时的处理。在许多情况下，动作名称是不相关的，例如，由于迁移代表一个内部流程活动，使用了一个特殊的标记 τ 或者在动作名称不相关的情况下，甚至可以省略动作标记。在描述迁移系统时，原子命题集合 AP 的选择通常是不确定的，如可以假定 $\text{AP} \subseteq S$ 和标签函数 $L(s) = \{s\} \cap \text{AP}$。

使用迁移系统对软硬件系统建模时，需要注意非确定性的问题，这里采用的非确定性选择是通过交错对独立活动并行的建模和对产生冲突情况的建模。例如，如果两个进程都要获取一个共享资源，那么本质上交错指的是控制并行进程的动作指令的非确定选择。除了并行性，非确定性对于抽象目标、规约不足、未知或不可预测环境接口的建模也是很重要的。饮料自动售货机(见图 3-2)是后面这种情况的例子，其中使用者需要做一个非确定的选择，也就是在 select 状态的两个 τ-迁移中选一个来获得两种饮料中的一种。“规约不足”指的是在早期设计阶段给系统提供的一个粗糙模型中，通过非确定性表示几种可能行为的选择。这个想法是由于在后面更细化的步骤中，设计者会实现非确定选择中的一个，而舍弃其他的选择。从这个意义上说，迁移系统中的非确定性可以代表实现上的自由性。

3.1.3　计算

一个迁移系统的计算(也称执行或运行)来自系统非确定性的选择，表示迁移系统的一

个可能的行为。

定义 3.7 一个迁移系统 TS=$(S,T,\rightarrow,I,\mathrm{AP},L)$。

(1) TS 的一个有限计算片断 σ 是一个以状态为结尾的状态与动作交错的序列：

$$\sigma=s_0\tau_1 s_1\tau_2\cdots\tau_n s_n,\quad \text{对所有}\quad 0\leqslant i<n, s_i \xrightarrow{\tau_{i+1}} s_{i+1}$$

其中 $n\geqslant 0$，把 n 作为计算片断 σ 的长度。

(2) TS 的一个无限计算片断 ρ 是一个无限的状态与动作的交错序列：

$$\rho=s_0\tau_1 s_1\tau_2 s_2\tau_3\cdots,\quad \text{对所有}\quad i\geqslant 0, s_i \xrightarrow{\tau_{i+1}} s_{i+1}$$

需要注意的是，序列 s ($\in S$)是一个长度 $n=0$ 的有限计算片断。一个无限计算片断的每个奇数长度的前缀都是一个有限的计算片断。计算片断 $\sigma=s_0\tau_1\cdots\tau_n s_n$ 和 $\rho=s_0\tau_1 s_1\tau_2\cdots$分别被写成

$$\sigma=s_0 \xrightarrow{\tau_1}\cdots\xrightarrow{\tau_n} s_n \quad \text{和} \quad \rho=s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2}\cdots$$

当一个计算片断不能再延长了，就称它是最大化的。

定义 3.8 一个最大化的计算片断要么是一个以某终止状态结尾的有限计算片断，要么是一个无限计算片断。如果一个计算片断始于某初始状态(也就是 $s_0\in I$)，则称它是初始化的。

例 3.2 例 3.1 中描述的饮料自动售货机的计算片断的例子如下，为了简单起见，动作名称被简写，例如 sget 是 get_soda 的简写，coin 表示 insert_coin。

$$\rho_1=\mathrm{pay}\xrightarrow{\mathrm{coin}}\mathrm{select}\xrightarrow{\tau}\mathrm{soda}\xrightarrow{\mathrm{sget}}\mathrm{pay}\xrightarrow{\mathrm{coin}}\mathrm{select}\xrightarrow{\tau}\mathrm{soda}\xrightarrow{\mathrm{sget}}\cdots$$

$$\rho_2=\mathrm{select}\xrightarrow{\tau}\mathrm{soda}\xrightarrow{\mathrm{sget}}\mathrm{pay}\xrightarrow{\mathrm{coin}}\mathrm{select}\xrightarrow{\tau}\mathrm{beer}\xrightarrow{\mathrm{bget}}\cdots$$

$$\sigma=\mathrm{pay}\xrightarrow{\mathrm{coin}}\mathrm{select}\xrightarrow{\tau}\mathrm{soda}\xrightarrow{\mathrm{sget}}\mathrm{pay}\xrightarrow{\mathrm{coin}}\mathrm{select}\xrightarrow{\tau}\mathrm{soda}$$

计算片断 ρ_1 和 σ 是初始化的，而 ρ_2 不是初始化的。σ 不是最大化的，因为它没有以一个终止状态结尾，假设 ρ_1 和 ρ_2 都是无限的，则它们是最大化的。

定义 3.9 迁移系统的一个计算是一个初始化的最大化的计算片断。

在例 3.2 中，ρ_1 是一个计算，而 ρ_2 和 σ 不是，因为 ρ_2 是最大化的却不是初始化的，σ 是初始化的却不是最大化的。

定义 3.10 一个迁移系统 TS=$(S,T,\rightarrow,I,\mathrm{AP},L)$，如果存在一个初始化的有限的计算片断：

$$s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2}\cdots\xrightarrow{\tau_n} s_n=s$$

则称状态 $s\in S$ 是可到达的，Reach(TS)表示 TS 中所有可达状态的集合。

3.2 应 用 举 例

本节通过对时序电路、数据依赖系统(一种简单的串行程序)和并发系统的建模来阐述迁移系统的应用。在这些例子中，状态代表可能的存储设置(即相关“变量”的值)，状态改变(即迁移)代表“变量”的改变。这里的变量需要从广义理解，对计算机程序来说，一个变量可以是一个控制变量(如一个程序计数器)，也可以是一个程序变量，而对电路来说，一个变量

代表一个寄存器或一个输入位。

3.2.1 时序电路

考虑一个用迁移系统对时序电路建模的简单例子。

例 3.3 在图 3-3 的时序电路的电路图中,输入变量是 x,输出变量是 y,寄存器是 r,输出变量 y 的控制函数通过表达式 $\lambda_y=\neg(x\oplus r)$ 给出,其中⊕代表异或(XOR),寄存器值的改变是通过电路函数 $\delta_r=x\vee r$ 实现的。

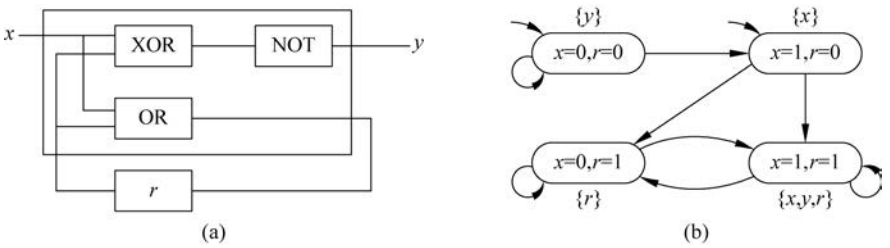

图 3-3 一个简单时序电路的迁移图

需要注意的是,一旦寄存器的值是[$r=1$],r 就会保持这个值不变,在寄存器初始值是[$r=0$]的情况下,电路的行为可以通过状态空间为 $S=\mathrm{Eval}(x,r)$ 的迁移系统 TS 进行建模,其中 $\mathrm{Eval}(x,r)$ 代表输入变量 x 和寄存器变量 r 的值的集合,TS 的初始状态是 $I=\{<x=0,r=0>,<x=1,r=0>\}$。注意,这里有两个初始状态,是因为没有假定输入位 x 的初始值。

动作集合在这里不相关,可省略,迁移直接来自函数 λ_y 和 δ_r。例如,如果下一个输入位等于 0,则迁移为 $<x=0,r=1>\rightarrow<x=0,r=1>$;如果下一个输入位等于 1,则迁移为 $<x=0,r=1>\rightarrow<x=1,r=1>$。

下面考虑标签 L,使用的原子命题集合 $\mathrm{AP}=\{x,y,r\}$,那么状态 $<x=0,r=1>$ 可以被标记为 $\{r\}$,它没有标签 y 是因为电路函数 $\neg(x\oplus r)$ 在该状态的值是 0。对于状态 $<x=1,r=1>$,由于 λ_y 的值是 1,则标签 $L(<x=1,r=1>)=\{x,y,r\}$,于是可以得到:$L(<x=0,r=0>)=\{y\}$,$L(<x=1,r=0>)=\{x\}$,该迁移系统的标签在图 3-3(b)中已经表示出来。

还可以使用命题 $\mathrm{AP}'=\{x,y\}$,寄存器的值假定是不可见的,那么可以得到:

$$L'(<x=0,r=0>)=\{y\}\quad L'(<x=0,r=1>)=\varnothing$$
$$L'(<x=1,r=0>)=\{x\}\quad L'(<x=1,r=1>)=\{x,y\}$$

AP'的命题足以被形式化,如属性"输出位 y 经常被设成无限的",但是和寄存器 r 相关的属性是不可表达的。

在这个例子中使用的方法可以推广到任意的时序电路,这个电路有 n 个输入位 $x_1,x_2,\cdots,x_n$,m 个输出位 $y_1,y_2,\cdots,y_m$ 和 k 个寄存器 $r_1,r_2,\cdots,r_k$。迁移系统的状态代表这 $n+k$ 个输入位和寄存器 $x_1,x_2,\cdots,x_n,r_1,r_2,\cdots,r_k$ 的值,输出位的值通过输入位和寄存器的值得到,也可以从状态中得到。假定输入位的值(通过电路环境)是非确定得到的,另外,设寄存器的初始值是[$r_1=c_{0,1},\cdots,r_k=c_{0,k}$],其中 $c_{0,i}$ 表示寄存器 i 的初始值,$0<i\leqslant k$。对该时序电路建模的迁移系统 $\mathrm{TS}=(S,T,\rightarrow,I,\mathrm{AP},L)$ 由以下组成部分,状态空间

$S=\mathrm{Eval}(x_1,x_2,\cdots,x_n,r_1,r_2,\cdots,r_k)$，这里的 $\mathrm{Eval}(x_1,x_2,\cdots,x_n,r_1,r_2,\cdots,r_k)$代表输入变量 x_i 和寄存器 r_i 的值的集合，并且等同于集合$\{0,1\}^{n+k}$（一个 $s\in\mathrm{Eval}(\cdot)$的值将值 $s(x_i)\in\{0,1\}$映射到输入位 x_i 上，类似地，每个寄存器 r_j 是$s(r_j)\in\{0,1\}$的映射。为了简化这个问题，假设每个 $s\in S$ 是一个$n+k$ 位的元组，当且仅当 x_i 的值为 1 时，第 i 位才会被设置，因此第 $n+j$ 位表示 r_j 的值）。初始状态的形式是$(\cdots,c_{0,1},\cdots,c_{0,k})$，其中 k 个寄存器的值已经给出，前面的 n 位表示输入位的值，这些值是任意的，因此初始状态集是

$$I=\{(a_1,a_2,\cdots,a_n,c_{0,1},c_{0,2},\cdots,c_{0,k})\mid a_1,a_2,\cdots,a_n\in\{0,1\}\}$$

动作集合 T 是不相关的，用 $T=\{\tau\}$表示，为了简洁起见，原子命题集合设为

$$\mathrm{AP}=\{x_1,x_2,\cdots,x_n,y_1,y_2,\cdots,y_m,r_1,r_2,\cdots,r_k\}$$

（实际上它可以定义为该 AP 的任意子集）因此任意寄存器、输入位和输出位都可以作为一个原子命题，标签函数设定为任意状态 $s\in\mathrm{Eval}(x_1,x_2,\cdots,x_n,r_1,r_2,\cdots,r_k)$，这正好是在状态 s 下 x_i,r_j 赋值为 1 的原子命题。如果对于状态 s，输出位 y_i 的值为 1，那么原子命题 y_i 也是$L(s)$的一部分。因此，有

$$L(a_1,a_2,\cdots,a_n,c_1,c_2,\cdots,c_k)=\{x_i\mid a_i=1\}\cup\{r_j\mid c_j=1\}$$
$$\cup\{y_i\mid s\vDash\lambda_{yi}(a_1,a_2,\cdots,a_n,c_1,c_2,\cdots,c_k)=1\}$$

其中 $\lambda_{yi}:S\to\{0,1\}$是一个跳转函数，对应于由门电路得到的输出位 y_i。

迁移用来表示行为，用 δ_{rj} 表示电路图中寄存器 r_j 的迁移函数，当且仅当 $c_j'=\delta_{rj}(a_1,a_2,\cdots,a_n,c_1,c_2,\cdots,c_k)$时，有

$$(a_1,a_2,\cdots,a_n,c_1,c_2,\cdots,c_k)\xrightarrow{\tau}(a_1',a_2',\cdots,a_n',c_1',c_2',\cdots,c_k')$$

这里假设输入位值的改变是不确定的，因此没有在 $a_1',a_2',\cdots,a_n'$上加约束。

读者可以使用这种方法自行检验图 3-3(a)的电路，该电路确实可以使用图 3-3(b)的迁移系统表示。

3.2.2 数据依赖系统

一个数据依赖系统的可执行动作一般来自分支条件，例如：

$$\text{if}\quad x\ \%\ 2=1\ \text{then}\ x:=x+1\ \text{else}\ x:=2\cdot x\quad\text{fi}$$

当用一个迁移系统对该程序片断进行建模时，迁移的条件可以省略并且条件分支可以通过非确定性代替，但这将导致该迁移系统只能验证较少的属性。此外还可以使用条件迁移，并将（标有条件的）结果图应用到随后要被验证的迁移系统中，以下通过示例详细介绍这个方法。

例 3.4 考虑例 3.1 中描述的饮料自动售货机的一个扩展，该机器可以计算苏打水和啤酒瓶子的数量，当自动售货机为空时，将返回投入的硬币。为了简便起见，自动售货机由两个位置 start 和 select 表示，下面两个条件迁移是对投了一个硬币和重新加满售货机的建模：

$$\text{start}\xrightarrow{\text{true:coin}}\text{select}\qquad\text{start}\xrightarrow{\text{true:refill}}\text{start}$$

条件迁移的标记形式是 $g:\tau$，其中 g 是一个布尔表达式（叫作卫式），τ 是一个 g 成立才可能发生的动作。由于上面两个条件迁移的条件总是成立的，那么动作 coin 在开始位置也总是可行的，为了简洁起见，假定重新填充后，两种饮料都会装满。下面两个条件迁移是对如果饮料售货机中还有苏打水（或啤酒），那么就能获得苏打水（或啤酒）的建模：

$$\text{select} \xrightarrow{\text{nsoda}>0:\text{sget}} \text{start} \quad \text{select} \xrightarrow{\text{nbeer}>0:\text{bget}} \text{start}$$

变量 nsoda 和 nbeer 分别记录了机器中苏打水和啤酒的数量。下面这个迁移表示当机器里没有任何瓶子,返还投的硬币时,自动售货机会自动转到初始 start 位置:

$$\text{select} \xrightarrow{\text{nsoda}=0 \wedge \text{nbeer}=0:\text{ret\_coin}} \text{start}$$

设两个饮料储藏室的最大容量是 max,投硬币(通过动作 coin)不会使饮料的数量发生改变,同样返还硬币(通过动作 ret_coin)也不会使饮料的数量发生改变,其他动作的效果如下:

$$\begin{aligned} &\text{refill} && \text{nsoda} := \text{max};\ \text{nbeer} := \text{max} \\ &\text{sget} && \text{nsoda} := \text{nsoda} - 1 \\ &\text{bget} && \text{nbeer} := \text{nbeer} - 1 \end{aligned}$$

由位置作为结点和条件迁移作为边组成的图不是一个迁移系统,因为边当中加了条件,不过迁移系统可以由这张图演变得到。例如,图 3-4 描绘的是 max 等于 2 的扩展迁移系统,迁移系统的状态与其所处图中的位置和自动售货机中苏打水和啤酒的数量(在图中分别用白点和黑点表示)都有关系。

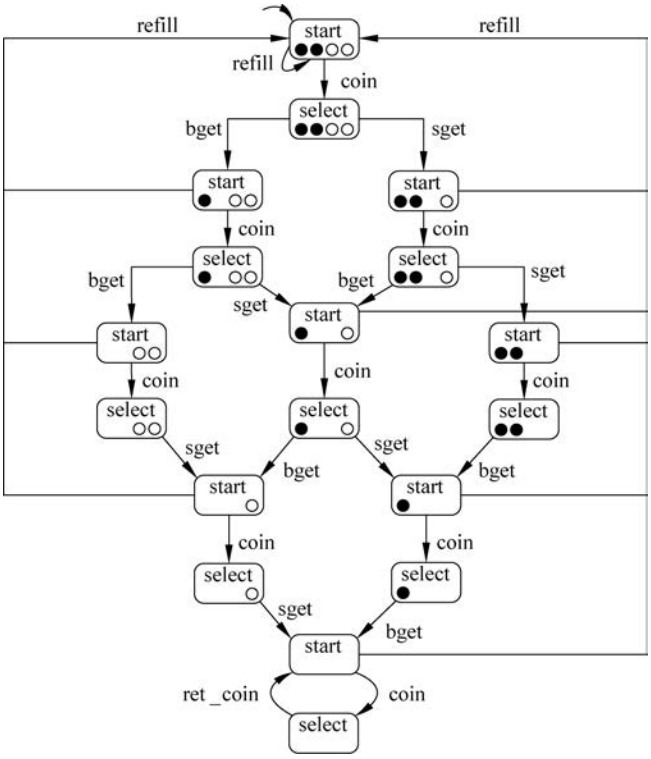

图 3-4　对扩展的饮料自动售货机建模的迁移系统

前面这个例子可以通过程序图(program graph)在一个类型变量集合 Var 上形式化,在这个例子中的变量就如 nsoda 和 nbeer。实际上,这意味着一个标准化的类型(如 boolean、integer 或 char)和每个变量都有关系,变量 x 的类型叫作 x 的论域 dom(x)。设 Eval(Var)表示变量赋值后变量集合的值,Cond(Var)是 Var 上的布尔条件集合,也就是命题符号形如 $\bar{x} \in \overline{D}$ 的命题逻辑公式,其中 $\bar{x} = (x_1, x_2, \cdots, x_n)$ 是一个由一组在 Var 中不同变量组成的

元组，$\overline{D}$ 是 $\mathrm{dom}(x_1)\times\mathrm{dom}(x_2)\times\cdots\times\mathrm{dom}(x_n)$的一个子集。例如，下面的命题是一个关于变量 x，x'和变量 y 的布尔条件：

$$(-3<x-x'\leqslant 5)\wedge(x\leqslant 2\cdot x')\wedge(y=\mathrm{green})$$

其中 x 和 x'是整型变量，$\mathrm{dom}(y)$可以是{red，green}。

不限制论域，$\mathrm{dom}(x)$可以是一个任意的集合，元素个数也可能是无限的，然而在实际的计算机系统中，所有的论域都是有限的(例如，integer 类型只包含整数 n 的一个有限的定义域，即$-2^{16}<n<2^{16}$)。而一个程序的逻辑和算法结构常常是建立在无限论域的基础上的，在论域上设置限制对于实现是很有用的，例如需要多少位表示整型变量会在后面的设计阶段考虑，但是这里先忽略。

一个类型变量集合上的程序图是一个边上标有变量和动作条件的有向图，动作的影响通过映射 Effect 表示：$T\times\mathrm{Eval}(\mathrm{Val})\to\mathrm{Eval}(\mathrm{Val})$。

映射 Effect 说明了在执行一个动作后，这些变量的值 η 会发生什么样的改变。例如，如果 τ 表示动作 $x:=y+5$，其中 x 和 y 都是整型变量，η 的值是 $\eta(x)=17$ 和 $\eta(y)=-2$，那么，有

$$\mathrm{Effect}(\tau,\eta)(x)=\eta(y)+5=-2+5=3,\quad \mathrm{Effect}(\tau,\eta)(y)=\eta(y)=-2$$

因此 $\mathrm{Effect}(\tau,\eta)$将 3 赋给 x，将-2 赋给 y。程序图中的结点叫作位置(location)，并且由于它们指定了哪些条件迁移是可行的而有了一种控制的功能。

定义 3.11 类型变量集合 Var 上的一个程序图是一个六元组 $\mathrm{PG}=(\mathrm{Loc},T,\mathrm{Effect},\hookrightarrow,\mathrm{Loc}_0,g_0)$，其中：

Loc 是位置集；

T 是动作集；

$\mathrm{Effect}: T\times\mathrm{Eval}(\mathrm{Val})\to\mathrm{Eval}(\mathrm{Val})$是效应函数；

$\hookrightarrow\subseteq\mathrm{Loc}\times\mathrm{Cond}(\mathrm{Var})\times T\times\mathrm{Loc}$ 是条件迁移关系；

$\mathrm{Loc}_0\subseteq\mathrm{Loc}$ 是初始位置集；

$g_0\in\mathrm{Cond}(\mathrm{Var})$是初始条件。

标记 $l\overset{g:\tau}{\hookrightarrow}l'$是$(l,g,\tau,l')\in\hookrightarrow$的简洁写法，条件 g 也叫作条件迁移 $l\overset{g:\tau}{\hookrightarrow}l'$的卫式，如果卫式是一个重言式(如 $g=\mathrm{true}$ 或 $g=(x<1)\vee(x\geqslant 1)$)，那么可以简写成 $l\overset{\tau}{\hookrightarrow}l'$。

在位置 $l\in\mathrm{Loc}$ 上的行为依赖当前变量的值 η，在值 η 上满足条件 g(即 $\eta\vDash g$)的所有迁移 $l\overset{g:\tau}{\hookrightarrow}l'$之间会存在一个非确定的选择。根据 $\mathrm{Effect}(\tau,\cdot)$，动作 τ 的执行可以改变变量的值，接着系统迁移到位置 l'，如果没有可用的迁移，系统就会停止。

例 3.5 例 3.4 中的图是一个程序图，变量集是 Var={nsoda，nbeer}，其中变量的定义域是{0，1，…，max}，位置集是 Loc={start，select}，Loc_0={start}，并且 T={bget，sget，coin，ret_coin，refill}，动作的效应可以表述为

$$\mathrm{Effect}(\mathrm{coin},\eta)=\eta$$
$$\mathrm{Effect}(\mathrm{ret\_coin},\eta)=\eta$$
$$\mathrm{Effect}(\mathrm{sget},\eta)=\eta[\mathrm{nsoda}:=\mathrm{nsoda}-1]$$
$$\mathrm{Effect}(\mathrm{bget},\eta)=\eta[\mathrm{nbeer}:=\mathrm{nbeer}-1]$$
$$\mathrm{Effect}(\mathrm{refill},\eta)=[\mathrm{nsoda}:=\mathrm{max},\mathrm{nbeer}:=\mathrm{max}]$$

这里,$\eta[\text{nsoda} := \text{nsoda} - 1]$是对 $\eta'(\text{nsoda}) = \eta(\text{nsoda}) - 1, \eta'(x) = \eta(x)$中 η'求值的简写,初始条件 $g_0 = (\text{nsoda} = \max \wedge \text{nbeer} = \max)$说明开始时两种饮料都已装满。

每个程序图都可以转变为一个迁移系统,迁移系统的状态可以由一个控制部分(即程序图中的一个位置 l)和变量的值组成,因此状态可以用$<l, \eta>$表示,初始状态是满足初始条件 g_0 的初始位置。为了描述一个程序图描述的系统的属性,命题的 AP 集合由位置集 $l \in \text{Loc}$(能够说明系统当前所在的控制位置)和变量的布尔表达式组成。例如:

$$(x \leqslant 5) \wedge (y \text{ 是偶数}) \wedge (l \in \{1,2\})$$

该命题是用整型变量 x、y 和自然数位置描述的。状态标签形如$<l, v>$,是由 l 和在 Var 上满足 η 的所有条件组成的。迁移关系表示为,无论什么时候在程序图中有一个条件迁移 $l \overset{g:\tau}{\hookrightarrow} l'$,并且卫式 g 使 η 值不变,那么都有一个迁移从状态$<l, \eta>$到状态$<l', \text{Effect}(\tau, \eta)>$,以上形式化表述为

定义 3.12 程序图 $\text{PG} = (\text{Loc}, T, \text{Effect}, \hookrightarrow, \text{Loc}_0, g_0)$中变量的 Var 集上的迁移系统是一个六元组 $\text{TS} = (S, T, \rightarrow, I, \text{AP}, L)$,其中:

$S = \text{Loc} \times \text{Eval}(\text{Var})$;

$\rightarrow \subseteq S \times T \times S$ 定义为 $\dfrac{l \xrightarrow{g:\tau} l' \wedge \eta \vDash g}{<l, \eta> \xrightarrow{\tau} <l', \text{Effect}(\tau, \eta)>}$;

$I = \{<l, \eta> \mid l \in \text{Loc}_0, \eta \vDash g_0\}$;

$\text{AP} = \text{Loc} \cup \text{Cond}(\text{Var})$;

$L(<l, \eta>) = \{l\} \cup \{g \in \text{Cond}(\text{Var}) \mid \eta \vDash g\}$。

TS(PG)的定义给出了一个很大的命题集合 AP,一般地,在描述系统属性时,只需要 AP 中的一小部分。

3.2.3 并发和交错

前面介绍了迁移系统的定义,并且阐述了如何通过迁移系统对时序电路和数据依赖系统进行有效的建模。实际上,大多数的软硬件系统不是串行的,而是并行的。当有多个迁移系统 $\text{TS}_1, \text{TS}_2, \cdots, \text{TS}_n$ 且它们的进程行为是并行的,可以用以下方式表示:

$$\text{TS} = \text{TS}_1 \parallel \text{TS}_2 \parallel \cdots \parallel \text{TS}_n$$

这里的 $\parallel$ 是连接的符号,本节通过例子的方式介绍 $\parallel$ 的几个变式。注意,上面的组合会在 TS_i 中进行重用,即 TS_i 是由几个迁移系统组成的迁移系统:

$$\text{TS}_i = \text{TS}_{i,1} \parallel \text{TS}_{i,2} \parallel \cdots \parallel \text{TS}_{i,n}$$

通过分级的方式使用并行组合,复杂的系统可以用一种结构化的方式描述。

采用交错执行方式作为并发系统建模的基本想法最初由 E. W. Dijkstra 于 1965 年提出。并发系统由多个单独部分组成,整个系统的状态是由多个单独部分的状态构成的。系统的动作同样也会由多个单独部分的动作交织构成。因此交错可以用来表示并发,也就是在同时运行的进程之间的非确定性选择。这个观点是建立在只有一个处理器是可用的基础上的,进程的动作在其中都是相互关联的。“单处理器的观点”只是一个建模概念,也可以用在运行在不同处理器上的进程。因此不需要假设不同进程间的执行顺序,例如,如果有两个完全互不依靠的无终止的进程 P 和 Q,那么下面的顺序都是可能的:

$$P\ Q\ P\ Q\ P\ Q\ Q\ Q\ P\cdots$$
$$P\ P\ Q\ P\ P\ Q\ P\ P\ Q\cdots$$
$$P\ Q\ P\ P\ Q\ P\ P\ P\ Q\cdots$$

其中 P 和 Q 的动作可以是关联的。

例 3.6 一个非交叉(平行)路的两个交通灯的迁移系统,假定两个交通灯的跳转是彼此之间完全独立的。例如,交通灯由过路的行人控制,每个交通灯用一个两个状态的简单迁移系统来建模,一个状态表示红灯,另一个状态表示绿灯。两个灯的并行组合如图 3-5 所示,其中 ||| 表示交错符号。原则上,交通灯之间的任何连接形式都是可行的。例如,在初始状态两个交通灯都是红色,那么在哪个灯变绿之中就需要做一个非确定性选择。注意,非确定性是可以描述出来的,只是这里没有对交通灯之间的调度问题进行建模。

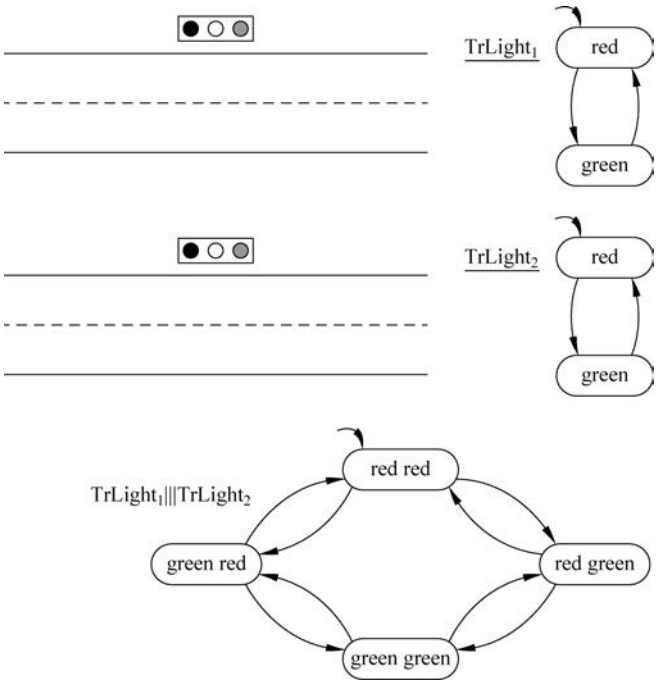

图 3-5 交通灯的迁移系统

系统中交错的重要依据是当并发运行的独立动作 α 和 β 以任意顺序成功执行时,都会产生同一个结果,这种情况可以形式化表示为

$$\mathrm{Effect}(\alpha \mid\mid\mid \beta,\eta)=\mathrm{Effect}((\alpha;\beta)+(\beta;\alpha),\eta)$$

其中分号(;)代表顺序执行;+代表非确定性选择;||| 代表独立活动的并发执行。可以用两个独立的赋值简单地理解上述内容:

$$\underbrace{x:=x+1}_{=\alpha} \mid\mid\mid \underbrace{y:=y-2}_{=\beta}$$

当初始值 $x=0$,$y=7$ 时,无论 α 和 β 的赋值是并发执行(即同时)还是以一个任意的顺序执行的,之后 x 的值是 1,y 的值是 5。可以用如图 3-6 所示的迁移系统表示:

注意,动作之间的无关性很重要。如果动作之间是相关的,那么动作的顺序就很重要。

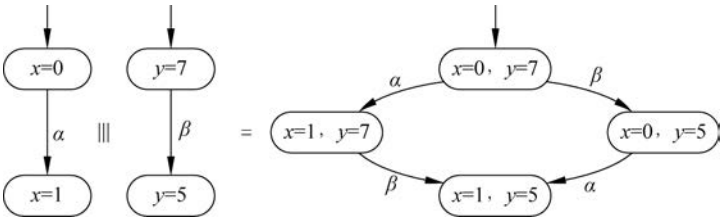

图 3-6　具有独立动作的交错

例如并行程序 $x:=x+1 \,|||\, x:=2\cdot x$(初始值是 $x=0$),那么变量 x 的最终值就与 $x:=x+1$ 和 $x:=2\cdot x$ 的执行顺序有关。

下面给出迁移系统交错的形式化定义。迁移系统 $TS_1 \,|||\, TS_2$ 代表一个由 TS_1 和 TS_2 描述的交错动作的并行系统。该描述假定没有通信和共享变量,$TS_1 \,|||\, TS_2$ 的(全局)状态是由 TS_i 的局部状态组成的状态对$<s_1,s_2>$,全局状态$<s_1,s_2>$的出迁移也是由 s_1 和 s_2 的出迁移组成的。因此,无论什么时候系统处于状态$<s_1,s_2>$时,都要在 s_1 和 s_2 的所有出迁移中做一个非确定性选择。

定义 3.13　设 $TS_i=(S_i,T_i,\rightarrow_i,I_i,AP_i,L_i)$是两个迁移系统($i=1,2$),则迁移系统 $TS_1 \,|||\, TS_2$ 定义为

$$TS_1 \,|||\, TS_2=(S_1\times S_2,T_1\cup T_2,\rightarrow,I_1\times I_2,AP_1\cup AP_2,L)$$

其中迁移关系→定义如下:

$$\frac{s_1 \xrightarrow{\tau}_1 s_1'}{<s_1,s_2>\xrightarrow{\tau}<s_1',s_2>} \qquad \frac{s_2 \xrightarrow{\tau}_2 s_2'}{<s_1,s_2>\xrightarrow{\tau}<s_1,s_2'>}$$

标签函数定义为 $L(<s_1,s_2>)=L(s_1)\cup L(s_2)$。

交错符 ||| 可以用来对异步并发建模,异步并发就是子进程间是完全独立的,即没有任何信息的传递和共享变量。但是迁移系统的交错符对于大多数并发或通信的并行系统来说过于简化。下面通过涉及共享变量的例子说明这一点。

例 3.7　考虑下面并行程序的程序图,如图 3-7 所示。

$$\underbrace{x:=2\cdot x}_{\text{action}\alpha} \,|||\, \underbrace{x:=x+1}_{\text{action}\beta}$$

其中,设定初始值 $x=3$。(为了简化图形,位置可省略)迁移图 $TS(PG_1) \,|||\, TS(PG_2)$包含的不一致状态$<x=6,x=4>$并不能反映 α 和 β 并行的行为,有

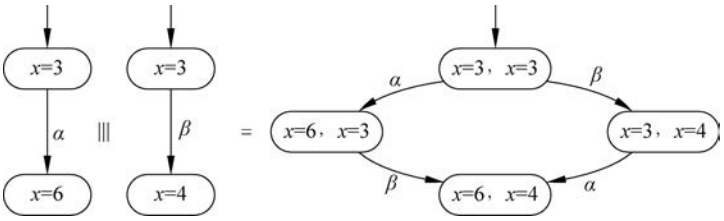

图 3-7　具有共享变量的交错

这个例子中的问题在于动作 α 和 β 使用共享变量 x，但是迁移系统的交错符没有考虑潜在的冲突，就直接构造了这些单独状态空间上的笛卡儿积。因此局部状态 $x=6$ 和 $x=4$ 不能描述互斥事件。

定义 3.14 设 $PG_i=(Loc_i, T_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$，$i=1,2$，是在变量 Var_i 上的两个程序图，则在 $Var_1 \cup Var_2$ 程序图 $PG_1 \,|||\, PG_2$ 定义为

$$PG_1 \,|||\, PG_2=(Loc_1 \times Loc_2, T_1 \uplus T_2, Effect, \hookrightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \wedge g_{0,2})$$

其中 $\hookrightarrow$ 定义如下：

$$\frac{l_1 \overset{g:\tau}{\hookrightarrow}_1 l_1'}{<l_1, l_2> \overset{g:\tau}{\hookrightarrow} <l_1', l_2>} \qquad \frac{l_2 \overset{g:\tau}{\hookrightarrow}_2 l_2'}{<l_1, l_2> \overset{g:\tau}{\hookrightarrow} <l_1, l_2'>}$$

如果 $\tau \in T_i$，$Effect(\tau, \eta)=Effect_i(\tau, \eta)$。

程序图 PG_1 和 PG_2 有同样的变量 $Var_1 \cap Var_2$，这些变量称为共享变量（也叫全局变量），在 Var_1/Var_2 中的变量是 PG_1 的局部变量，同样地，在 Var_2/Var_1 中的变量是 PG_2 的局部变量。

例 3.8 程序图 PG_1 和 PG_2 对应的赋值分别是 $x:=x+1$ 和 $x:=2 \cdot x$，程序图 $PG_1 \,|||\, PG_2$ 和它转化后的迁移系统 $TS(PG_1 \,|||\, PG_2)$ 如图 3-8 所示，其中假定 x 的初始值等于 3。注意，在迁移系统初始状态中的非确定性不表示并发，这正好是解决 $x:=x+1$ 和 $x:=2 \cdot x$ 都可以修改共享变量 x 的一种方式。

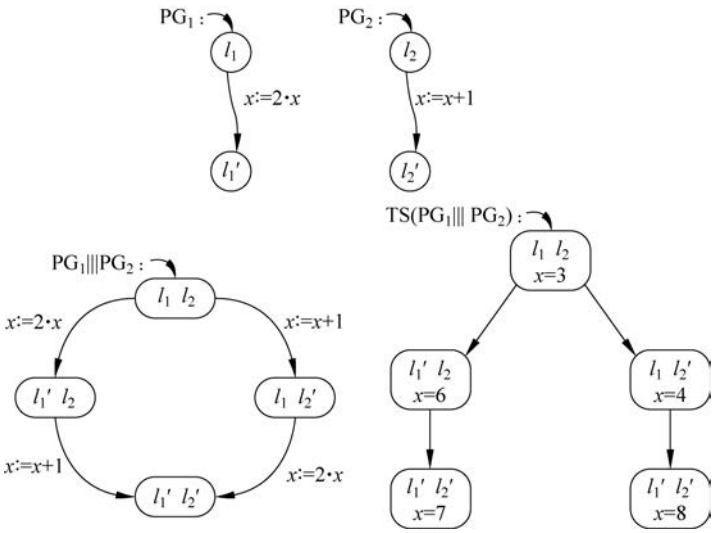

图 3-8 程序图的交错

局部变量和共享变量之间的区别对程序图 $PG_1 \,|||\, PG_2$ 的动作也有影响，获取共享变量的动作被认为是“临界的”，其他的动作被认为是非临界的。临界的动作和非临界的动作之间的区别可以在表示迁移系统 $TS(PG_1 \,|||\, PG_2)$ 的非确定性时清楚地看到。迁移系统一个状态的非确定性可以表示为：

（1）程序图 PG_1 或 PG_2 中的一个内部非确定性选择。

（2）PG_1 和 PG_2 的非临界动作的交错。

(3) PG_1 和 PG_2 的临界动作之间选择的解决方式(并发)。

特别地,PG_1 的非临界动作可以和 PG_2 的临界性和非临界动作并行,因为它只影响 PG_1 的局部动作,对于 PG_2 的非临界动作同样适用。但是 PG_1 和 PG_2 的临界动作不能同时执行,因为共享变量的值依赖动作的执行顺序。所以需要通过一个合适的调度策略描述 PG_1 和 PG_2 的临界动作的并发情况。

例 3.9 用迁移系统描述一个多用户终端执行程序 MUTEX(MULtiuser Terminal Executive):

$$\textbf{MUTEX}:\ \text{flag}:\ \text{array}[0..n-1]\text{of}\ 0..4\ \text{where flag}:=0;$$
$$P[0]\ \|\ P[1]\ \|\ \cdots\ \|\ P[n-1]$$

其中每一进程 P_i 为

```
l0:  loop forever do
     begin
      l1: Non Critical
      l2: flag[i] := 1
      l3: wait until ∀ j:0≤j<n: (flag[j]<3)
      l4: flag[i] := 3
      l5: if ∃j:0≤j<n: (flag[j] = 1) then
        begin
          l6: flag[i] := 2
          l7: wait until ∃j: 0≤j<n (flag[j] = 4)
        end
      l8: flag[i] := 4
      l9: wait until ∀j: 0≤j<i: (flag[j] <2)
      l10: Critical
      l11: wait until ∀j:i<j<n: (flag[j]<2∨flag[j]>3)
      l12: flag[i] := 0
    end
```

(1) 原子命题集 AP=$\{l_0,\cdots,l_{12},\text{flag}[0],\cdots,\text{flag}[n-1]\}$,其中 $l_0,\cdots,l_{12}\in\{0,\cdots,n-1\}$ 是控制变量,变量 flag[0],…,flag[$n-1$]表示对应的程序变量的当前值。

(2) 状态集 S 包含原子命题集 AP 在定义域上所有可能的赋值。

(3) 初始状态集 I:$(l_0\in\{0,\cdots,n-1\})\wedge(l_{1..12}=\phi)\wedge(\text{flag}[i]=0)$,即在程序的初始状态,所在进程驻留在 l_0 位置,且 flag(0),…,flag($n-1$)均为 0。

(4) 迁移关系:算法所在测试(如其中之一在语句 l_1 中)当作原子处理,即通过单一迁移完成每一次执行,相应地给出每一进程 $P[i]$及每一位置 l_k 的一个迁移 $\tau_k[i]$和对应的迁移关系 $\rho_k[i]$。例如:

状态 l_1 的迁移关系为 $\rho_1[i]$:$(i\in l_1)\wedge(\text{stay}\vee\text{move}(i,1,2))$,根据上述公式,进程 $P[i]$可以不确定地选择在 l_1 处停留或者从 l_1 移至 l_2。

状态 l_5 的迁移关系为 $\rho_5[i]$:$(i\in l_5)\wedge[(F_1\neq\phi)\wedge\text{move}(i,5,6)]\vee[(F_1=\phi)\wedge\text{move}(i,5,8)]$,当在 l_5 时,如果一些进程 $P[j]$具有 flag[j]=1 和 $F_1\neq\phi$,$P[i]$可以处理 l_6;如果没有进程 flag 值等于 1 和 $F_1=\phi$,也可以处理 l_8。

其余状态对应的迁移关系可类似给出。

(5) 迁移动作集 T：略。

(6) 标签函数 L：略。

3.3 本章小结

迁移系统是一种基于状态迁移的基本计算模型，通过系统(程序)的状态集合及对应状态间的迁移(也称操作)集合描述系统的计算行为，可以从不同角度对并发系统进行建模。迁移系统本质上是一种交错并发模型，即系统进程间的并发执行并不是真并发，而是通过各个原子迁移以不确定的顺序交错执行来表示的，其计算行为表现为状态与动作交错的序列。

基于迁移系统，人们提出多种并发模型，如进程代数 CSP 模型、CCS 模型等；通过对基本迁移系统进行扩展，如将公平性引入迁移系统，可得到公平迁移系统；将时间(或时钟)引入迁移系统，可分别得到时间(时钟)迁移系统等。

习 题 3

1. 给出如图 3-9 和图 3-10 所示的时序电路的迁移系统。

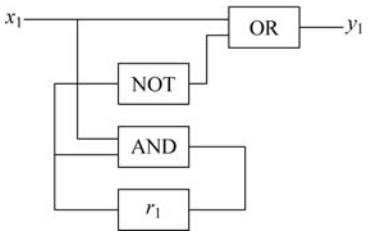

图 3-9　题 1 图(1)

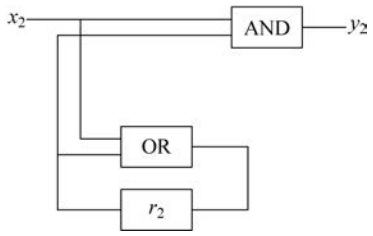

图 3-10　题 1 图(2)

2. 给出容量为 3 的栈的迁移系统(使用 top、pop、push 操作)。

3. 给定三个进程 P_1、P_2 和 P_3，共享一个整型变量 x，进程 P_i($i=1,2,3$)的程序如下：

```
for k_i = 1, … ,10 do
      LOAD(x);
      INC(x);
      STORE(x);
  od
```

P_i 执行 10 次 $x:=x+1$，$x:=x+1$ 通过 LOAD(x)、INC(x)和 STORE(x)这三个动作实现，如果这三个进程是并行的，且以 $x:=0$ 开始，那么整个过程是否有一个执行以 $x=2$ 结束？