# 第 3 章　线性时间性质

为了实现验证目的, 系统的迁移系统模型需要同时给出要验证的性质的描述. 本章介绍相对简单但很重要的几类性质, 形式化地定义这些性质, 给出自动检验它们的基本模型检验算法. 本章集中讨论线性时间性质的行为, 建立几类性质与迹行为之间的关系, 介绍公平性的初级形式, 并对其进行比较.

## 3.1　死　　锁

不发散 (即没有死循环) 的顺序程序都有终止状态, 即没有任何出迁移的状态. 但是, 对于并行系统, 计算通常不会终止, 例如, 考虑曾经讨论过的互斥程序. 在这样的系统中, 终止状态不是设计者所期望的, 多半是设计错误. 除了忘记指出某些活动这样的 “低级” 设计错误外, 这样的终止状态在大多数情况下意味着*死锁*. 即使有一些组件处于 (局部) 非终止状态, 但是只要整个系统处于终止状态, 也是死锁. 于是, 尽管有一些组件还有继续操作的可能, 但是整个系统却已停止. 当各组件都互相等待其他组件的进展时, 典型的死锁情形就发生了.

**例 3.1**　*错误设计造成的交通灯死锁*

考虑两个迁移系统的并行合成

$$\mathrm{TrLight}_1 \parallel \mathrm{TrLight}_2$$

它给两条交叉道路的交通灯建模. 两个交通灯用改变颜色的动作 $\alpha$ 和 $\beta$ 同步, 如图 3.1 所示.

让两个交通灯都从红灯开始是一个明显的低级错误, 这将导致死锁. 当第一个交通灯等待在动作 $\alpha$ 上同步时, 却阻塞了第二个交通灯, 因为它正等待在动作 $\beta$ 上同步. ■

**例 3.2**　*哲学家就餐*[译注 21]

这个例子起源于 Dijkstra, 它是并发系统领域最著名的例子之一.

5 位哲学家围坐在一张桌子旁, 桌子中央有一碗米饭. (有点脱俗的) 哲学家的生活由思考和吃饭 (还有下面将看到的等待) 组成. 为了从碗中得到一些米饭, 哲学家需要一双筷子. 但是, 两位相邻的哲学家之间只有一根筷子. 因此, 在任何时刻, 两位相邻的哲学家中至多只能有一人吃饭. 当然, 筷子的使用是独占的, 也不允许用手吃饭.

注意, 当每位哲学家都占有一根筷子时就会死锁. 要解决的问题是: 为哲学家设计一个协议, 使整个系统无死锁, 即至少有一位哲学家可以无限经常地吃饭和思考. 另外, 一个公平的解决方案可能要求每位哲学家都能够无限经常地吃饭和思考. 后一要求被称为无个体饥饿.

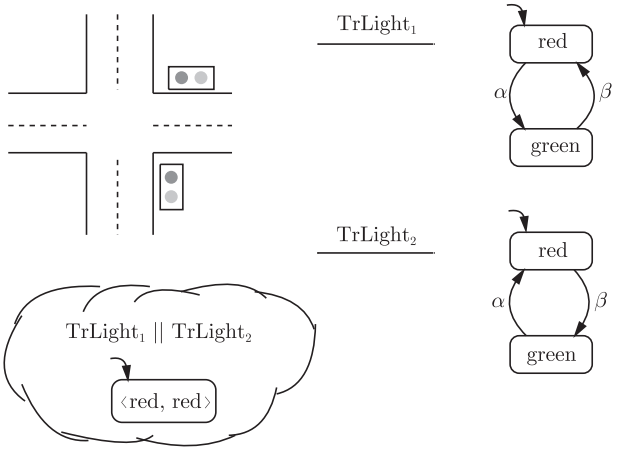


图 3.1　死锁情形的例子

下面这个设计显然不能保证无死锁. 假设哲学家和筷子都从 0 到 4 编号. 进一步地, 假设所有计算都是模 5 的, 即, 对于 $i = 0$, 筷子 $i - 1$ 表示筷子 4, 等等.

哲学家 $i$ 的左边是筷子 $i$, 右边是筷子 $i - 1$. 动作 $\mathrm{req}_{i,i}$ 表示哲学家 $i$ 拿起筷子 $i$, 对应地 $\mathrm{req}_{i-1,i}$ 表示哲学家 $i$ 拿起筷子 $i - 1$. 动作 $\mathrm{rel}_{i,i}$ 和 $\mathrm{rel}_{i-1,i}$ 表示相应地放下筷子.

哲学家 $i$ 的行为 (记为进程 $\mathrm{Phil}_i$) 由图 3.2 左侧所示的迁移系统描述. 实线箭头表示与筷子 $i$ 同步, 虚线箭头则表示与筷子 $i - 1$ 通信. 这些筷子作为独立的进程建模 (记为 $\mathrm{Stick}_i$), 哲学家通过动作 req 和 rel 与这些进程同步, 见图 3.2 中表示进程 $\mathrm{Stick}_i$ 的右侧部分. 筷子进程可在哲学家 $i + 1$ 使用筷子 $i$ 时避免哲学家 $i$ 再取它.

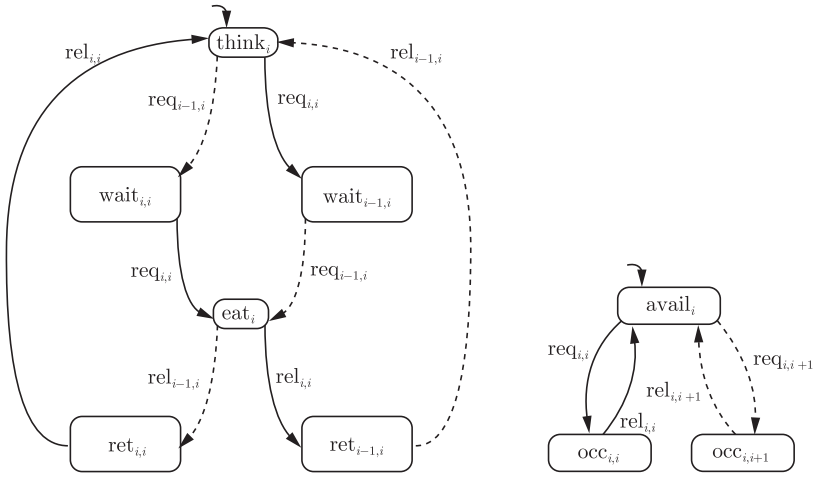


图 3.2　哲学家 $i$ 与筷子 $i$ 的迁移系统

整个系统的形式是:

$$\mathrm{Phil}_4 \parallel \mathrm{Stick}_3 \parallel \mathrm{Phil}_3 \parallel \mathrm{Stick}_2 \parallel \mathrm{Phil}_2 \parallel \mathrm{Stick}_1 \parallel \mathrm{Phil}_1 \parallel \mathrm{Stick}_0 \parallel \mathrm{Phil}_0 \parallel \mathrm{Stick}_4$$

这个 (在一开始认为理所当然的) 设计会造成死锁. 例如, 假如所有哲学家在同一时刻都拿

起他左边的筷子. 从初始状态

$$\langle \text{think}_4, \text{avail}_3, \text{think}_3, \text{avail}_2, \text{think}_2, \text{avail}_1, \text{think}_1, \text{avail}_0, \text{think}_0, \text{avail}_4 \rangle$$

开始, 经动作序列 $\text{req}_{4,4}$, $\text{req}_{3,3}$, $\text{req}_{2,2}$, $\text{req}_{1,1}$, $\text{req}_{0,0}$ (也可以是这 5 个动作的其他任何排列), 对应的执行就进入终止状态[译注 22]

$$\langle \text{wait}_{3,4}, \text{occ}_{3,3}, \text{wait}_{2,3}, \text{occ}_{2,2}, \text{wait}_{1,2}, \text{occ}_{1,1}, \text{wait}_{0,1}, \text{occ}_{0,0}, \text{wait}_{4,0}, \text{occ}_{4,4} \rangle$$

这一终止状态表示死锁, 其中每位哲学家都在等待另一位哲学家释放他需要的那根筷子.

解决这个问题的可能方案之一是在某一时间让筷子仅对一位哲学家可用. 对应的筷子进程如图 3.3 的右侧所示. 在状态 $\text{avail}_{i,j}$ 中, 只允许哲学家 $j$ 拿起筷子 $i$. 让一些筷子 (例如, 第 1 根、第 3 根和第 5 根) 从状态 $\text{avail}_{i,i}$ 开始, 而让其余筷子都从状态 $\text{avail}_{i,i+1}$ 开始, 这样就可以避免死锁情形.

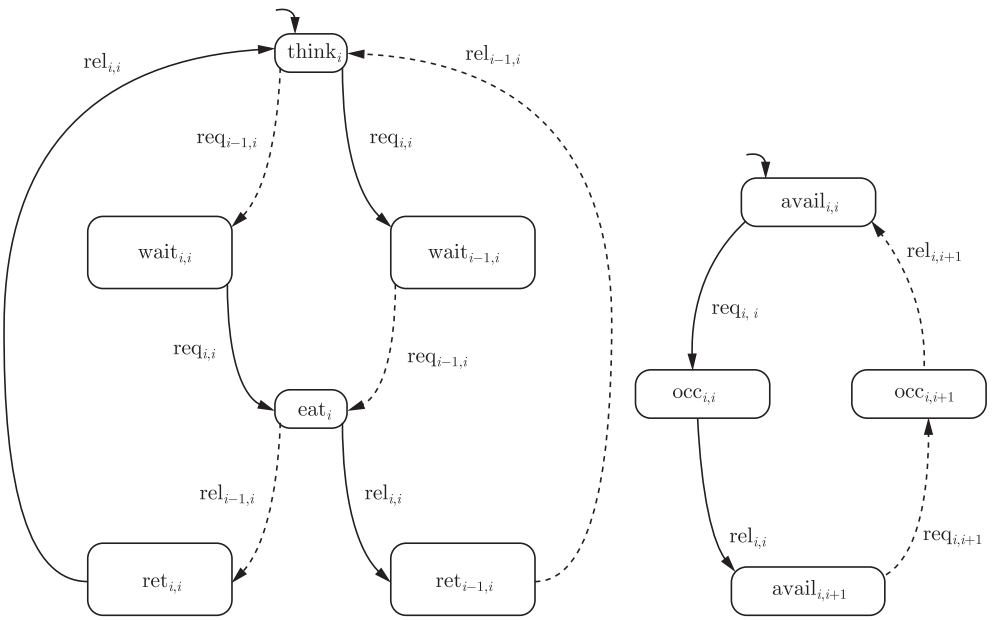


图 3.3　哲学家 $i$ 和筷子 $i$ 的改进的迁移系统

对并行系统经常要求的更高特性是针对组件失败的鲁棒性. 在哲学家就餐的情形中, 鲁棒性可用以下方式准确表述: 即使某位哲学家有了 “毛病” (如不再离开思考阶段)①, 也要保证无死锁和无饥饿. 改变哲学家和筷子的迁移系统, 使得只要哲学家 $i$ 正在思考 (即不需要筷子 $i$), 哲学家 $i+1$ 就可以拿起筷子 $i$, 而与筷子 $i$ 是处于状态 $\text{avail}_{i,i}$ 还是 $\text{avail}_{i,i+1}$ 无关紧要. 当对换第 $i$ 和第 $i+1$ 位哲学家的角色时也是如此. 通过这样的改变就可把上面简述的无死锁无饥饿解决方案改进为容错解决方案. 只需为每位哲学家 $i$ 添加一个布尔变量 $x_i$ (见图 3.4) 就可实现这种改变. 变量 $x_i$ 告诉哲学家 $i$ 的邻居他当前所处的位置. 如图 3.4所示, $x_i$ 是布尔变量, 它为 true 当且仅当哲学家 $i$ 正在思考. 如果筷子 $i$ 处于位置

① 形式上, 在有 “毛病” 的哲学家的迁移系统的状态 $\text{think}_i$ 处添加一个循环.

$\text{avail}_{i,i}$ (像以前一样), 或筷子 $i$ 处于位置 $\text{avail}_{i,i+1}$ 并且哲学家 $i+1$ 正在思考, 那么筷子 $i$ 对哲学家 $i$ 是可用的[译注 23].

注意, 上面 (最后几句话) 的描述是在程序图的层面上进行的. 整个系统是一个通道系统, 通道的容量为 0, 那些 req 和 rel 是握手动作. ■

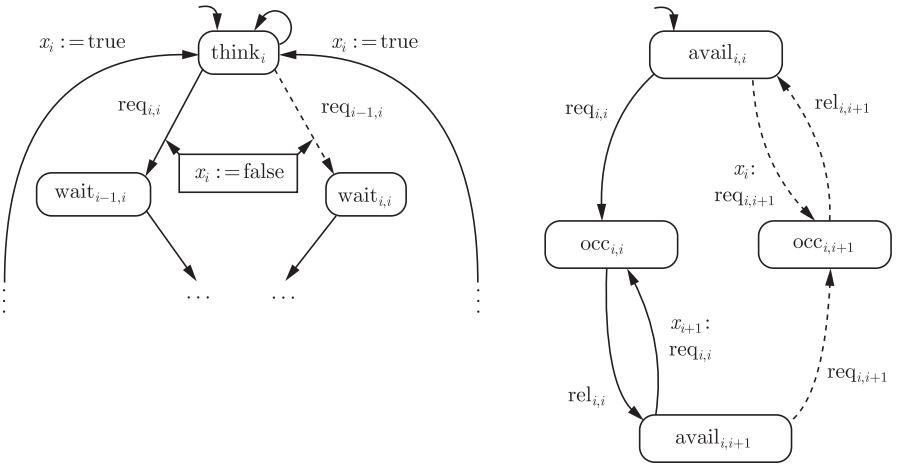


图 3.4 哲学家就餐的容错变体[译注 24]

## 3.2 线性时间行为

为了分析由迁移系统表示的计算机系统, 可以沿用基于动作或基于状态的方法. 基于状态的方法抛开动作, 只考虑状态序列中的标记; 相反, 基于动作的观点则抛开状态而只涉及迁移的动作标记 (也可把基于动作和基于状态的观点相结合, 但会涉及更多的定义和概念. 有鉴于此, 通常的做法是抛开动作标记或状态标记之一). 大部分现行的形式化描述及其相关的验证方法都可用这两个方面的对应方式确切地表达.

本章主要聚焦于基于状态的方法. 迁移的动作标记仅在通信模型中是必要的. 因此, 它们与后续章节无关. 因而用状态的原子命题来精准地表达系统性质. 所以, 验证算法在迁移系统的状态图上进行. 状态图是从迁移系统抽掉动作标记后得到的有向图.

### 3.2.1 路径与状态图

令 $\text{TS} = (S, \text{Act}, \to, I, \text{AP}, L)$ 是迁移系统.

**定义 3.1** 状态图

TS 的状态图记为 $G(\text{TS})$, 是有向图 $(V, E)$, 其顶点集是 $V = S$, 边集是 $E = \{(s, s') \in S \times S \mid s' \in \text{Post}(s)\}$. ■

在迁移系统 TS 的状态图中, 对于 TS 的每个状态都有一个顶点; 对于任何两个顶点 $s$ 和 $s'$, 只要 $s'$ 是 $s$ 在 TS 中的某个动作下的直接后继, 这两个顶点之间就有一条边. TS 的状态图可简单地用以下方式从 TS 得到: 删除所有状态标记 (即原子命题集合), 删除所有迁移标记 (即动作), 并忽略状态是否初始状态. 另外, (两个) 状态之间的多个迁移 (它们有

不同的动作标记) 只用一条边表示. 这好像意味着状态标记不再有用, 稍后将看到状态标记是如何用来检验性质的有效性的.

令 $\mathrm{Post}^*(s)$ 表示状态图 $G(\mathrm{TS})$ 中从 $s$ 开始可达的状态. 这个记号可以按通常方式 (即逐点扩张) 推广到状态的集合: 对于 $C \subseteq S$, 令

$$\mathrm{Post}^*(C) = \bigcup_{s \in C} \mathrm{Post}^*(s)$$

记号 $\mathrm{Pre}^*(s)$ 和 $\mathrm{Pre}^*(C)$ 有相似的含义. 从某个初始状态可达的状态的集合记为 $\mathrm{Reach}(\mathrm{TS})$, 等于 $\mathrm{Post}^*(I)$.

像第 2 章中解释的那样, 迁移系统的行为由执行片段定义. 回顾一下, 执行片段就是状态和动作的交替序列. 由于主要考虑基于状态的方法, 动作并不重要, 故略之. 迁移系统产生的运行叫作路径. 下面定义路径片段、起始路径片段和极大路径片段等. 这些概念可从执行的相同概念中删除动作得到.

**定义 3.2**　*路径片段*

TS 的*有限路径片段* $\widehat{\pi}$ 是满足以下条件的有限状态序列 $s_0 s_1 \cdots s_n$: 对任意 $0 < i \leqslant n$ 都有 $s_i \in \mathrm{Post}(s_{i-1})$, 其中 $n \geqslant 0$. 一个*无限路径片段* $\pi$ 是一个无限状态序列 $s_0 s_1 s_2 \cdots$, 它对所有 $i > 0$ 都满足 $s_i \in \mathrm{Post}(s_{i-1})$. ■

对无限路径片段 $\pi = s_0 s_1 s_2 \cdots$ 使用以下习惯记号. $\pi$ 的初始状态记为 $\mathrm{first}(\pi) = s_0$. 对于 $j \geqslant 0$, 令 $\pi[j] = s_j$ 表示 $\pi$ 的第 $j$ 个状态, $\pi[..j]$ 表示 $\pi$ 的第 $j$ 个前缀, 即, $\pi[..j] = s_0 s_1 \cdots s_j$. 类似地, $\pi$ 的第 $j$ 个后缀记为 $\pi[j..]$, 定义为 $\pi[j..] = s_j s_{j+1} s_{j+2} \cdots$. 对有限路径类似地定义这些概念. 此外, 对有限路径 $\widehat{\pi} = s_0 s_1 \cdots s_n$, 令 $\mathrm{last}(\widehat{\pi}) = s_n$ 表示 $\widehat{\pi}$ 的最后一个状态, $\mathrm{len}(\widehat{\pi}) = n$ 表示 $\widehat{\pi}$ 的长度; 对于无限路径 $\pi$, 这些概念定义为 $\mathrm{len}(\widehat{\pi}) = \infty$ 和 $\mathrm{last}(\widehat{\pi}) = \perp$, 其中 $\perp$ 表示未定义.

**定义 3.3**　*极大路径片段与起始路径片段*

无限路径片段或以终止状态结束的有限路径片段称为*极大路径片段*. 如果路径片段从一个初始状态开始, 即 $s_0 \in I$, 则称其为*起始路径片段*. ■

极大路径片段不能再延长, 它要么是无限的, 要么是结束于一个不能再发生迁移的状态. 令 $\mathrm{Paths}(s)$ 表示满足 $\mathrm{first}(\pi) = s$ 的极大路径片段 $\pi$ 的集合, $\mathrm{Paths}_{\mathrm{fin}}(s)$ 表示满足 $\mathrm{first}(\widehat{\pi}) = s$ 的有限路径片段 $\widehat{\pi}$ 的集合.

**定义 3.4**　*路径*

迁移系统 TS 的*路径*就是起始极大路径片段. [①] ■

令 $\mathrm{Paths}(\mathrm{TS})$ 表示 TS 中所有路径的集合, $\mathrm{Paths}_{\mathrm{fin}}(\mathrm{TS})$ 表示 TS 的起始有限路径片段的集合.

**例 3.3**　*饮料售货机*

考虑例 2.1 的饮料售货机. 为方便起见, 图 3.5 再一次给出它的迁移系统.

因为对每个状态 $s$, 它的标记都是简单的 $L(s) = \{s\}$, 所以状态的名字既可以用在路径中 (像本例这样), 也可用作原子命题 (以后将这样用). 以下是此迁移系统的路径片段:

① 重要的是要认识到路径在迁移系统和有向图中的概念差异. 迁移系统中的路径是极大的, 而图论意义下的有向图的路径未必是极大的. 另外, 通常要求有向图中的路径是有限的, 而迁移系统中的路径却可能是无限的.

$$\pi_1 = \text{pay select soda pay select soda} \cdots$$
$$\pi_2 = \text{select soda pay select beer} \cdots$$
$$\widehat{\pi} = \text{pay select soda pay select soda}$$

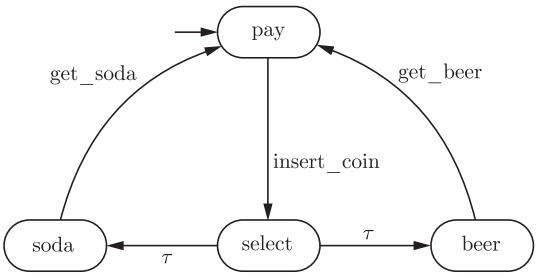


图 3.5 简单饮料售货机的迁移系统

这些路径片段是从例 2.2 给出的执行片段产生的. 只有 $\pi_1$ 是路径. 无限路径片段 $\pi_2$ 是极大的但不是起始的. $\widehat{\pi}$ 是起始的但不是极大的, 因为它是有限的而且结束于一个有出迁移的状态. 还可得到: $\text{last}(\widehat{\pi}) = \text{soda}$, $\text{first}(\pi_2) = \text{select}$, $\pi_1[0] = \text{pay}$, $\pi_1[3] = \text{pay}$, $\pi_1[..5] = \widehat{\pi}$, $\widehat{\pi}[..2] = \widehat{\pi}[3..]$, $\text{len}(\widehat{\pi}) = 5$, 以及 $\text{len}(\pi_1) = \infty$. ■

### 3.2.2 迹

(第 2 章中介绍的) 执行是由状态和动作构成的交替序列. 动作主要为互动 (的可能性) 建模, 无论是同步还是异步通信. 本章以后的主要兴趣不是互动, 而是执行期间访问的状态. 事实上, 可观察的不是状态自身, 而只是原子命题. 因此, 不用形为 $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \cdots$ 的执行, 而是考虑形为 $L(s_0)L(s_1)L(s_2)\cdots$ 的序列, 它登记在执行过程中成立的原子命题 (的集合). 这样的序列叫作迹.

所以, 迁移系统的迹就是字母表 $2^{\text{AP}}$ 上的单词. 接下来, 假设迁移系统没有终止状态. 在这种情况下, 所有迹都是无限字 (迁移系统的迹已定义为它的起始极大执行片段诱导的迹. 也可参见附录 A 的 A.2 节). 这一假设是为简单而做出的, 它不会产生严重制约. 首先, 在验证任何 (线性时间) 性质之前, 要做可达性分析以决定终止状态的集合. 如果确实遇到终止状态, 那么系统包含死锁, 在进行进一步的分析前必须修复. 也可对每个 (可能有终止状态的) 迁移系统 TS 作如下扩充: 对每个终止状态 $s$, 增加一个新状态 $s_{\text{stop}}$ 和一个迁移 $s \to s_{\text{stop}}$, 并给 $s_{\text{stop}}$ 增加一个自循环, 即 $s_{\text{stop}} \to s_{\text{stop}}$. 如此产生的等价迁移系统显然没有终止状态. ①

**定义 3.5** 迹与迹片段

设 $\text{TS} = (S, \text{Act}, \to, I, \text{AP}, L)$ 是没有终止状态的迁移系统. 无限路径片段 $\pi = s_0 s_1 s_2 \cdots$ 的迹定义为 $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2)\cdots$, 有限路径片段 $\widehat{\pi} = s_0 s_1 \cdots s_n$ 的迹定义为 $\text{trace}(\widehat{\pi}) = L(s_0)L(s_1)\cdots L(s_n)$. ■

从而, 路径片段的迹就是它在字母表 $2^{\text{AP}}$ 上诱导的有限或无限单词, 即在路径所含状态上成立的原子命题集合的序列. 路径集合 $\Pi$ 的迹集按通常方式定义为

① 另一种选择是为带终止状态的迁移系统修改线性时间框架. 本章主要概念仍然可用, 但需要某些修改以区别非极大有限路径与极大有限路径.

$$\mathrm{trace}(\Pi) = \{\mathrm{trace}(\pi) \mid \pi \in \Pi\}.$$

状态 $s$ 的一个迹就是具有 $\mathrm{first}(\pi) = s$ 的一个无限路径片段 $\pi$ 的迹. 相应地, 状态 $s$ 的一个有限迹就是始于 $s$ 的一个有限路径片段的迹. 令 $\mathrm{Traces}(s)$ 表示 $s$ 的迹的集合, $\mathrm{Traces}(\mathrm{TS})$ 是迁移系统 TS 的初始状态的迹的集合:

$$\mathrm{Traces}(s) = \mathrm{trace}(\mathrm{Paths}(s)), \quad \mathrm{Traces}(\mathrm{TS}) = \bigcup_{s \in I} \mathrm{Traces}(s)$$

类似地, 状态和迁移系统的有限迹可以定义为

$$\mathrm{Traces}_{\mathrm{fin}}(s) = \mathrm{trace}(\mathrm{Paths}_{\mathrm{fin}}(s)), \quad \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) = \bigcup_{s \in I} \mathrm{Traces}_{\mathrm{fin}}(s)$$

**例 3.4**　基于信号的互斥

考虑图 3.6 所示的迁移系统 $\mathrm{TS}_{\mathrm{Sem}}$. 这个双进程互斥的例子前面已在例 2.10 中讨论过.

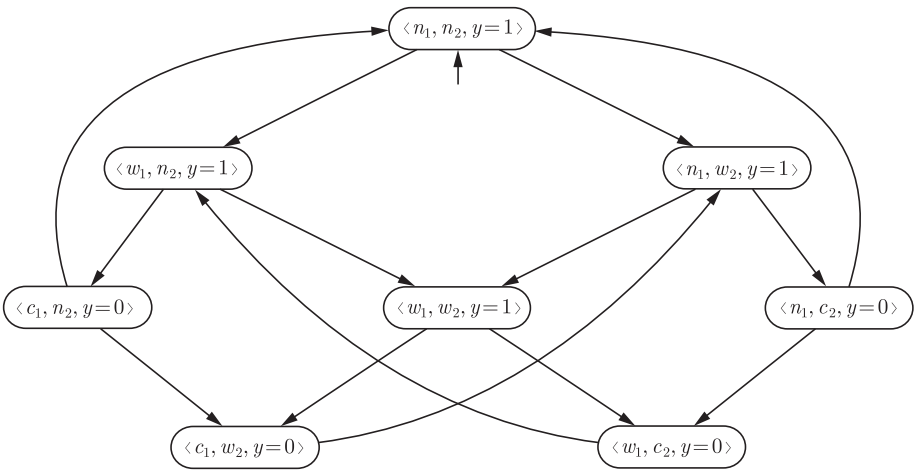


图 3.6　基于信号的互斥算法的迁移系统

设可用的原子命题是 $\mathrm{crit}_1$ 和 $\mathrm{crit}_2$, 即

$$\mathrm{AP} = \{\mathrm{crit}_1, \mathrm{crit}_2\}$$

在迁移系统 $\mathrm{TS}_{\mathrm{Sem}}$ 中, 命题 $\mathrm{crit}_1$ 在第一个进程 (记为 $P_1$) 处于关键节段的任何状态处都成立. 命题 $\mathrm{crit}_2$ 对于第二个进程 (即 $P_2$) 有相同的含义.

考虑如下执行: 进程 $P_1$ 和 $P_2$ 交替地进入它们的关键节段; 另外, 它们也只在另一个进程不处于关键节段时请求进入关键节段. 一个进程处于其关键节段而另一进程从非关键状态移入等待状态的情形是不可能的.

在 $\mathrm{TS}_{\mathrm{Sem}}$ 的状态图中, $P_1$ 首先进入关键节段的路径 $\pi$ 的形式是

$$\begin{aligned}\pi = \langle n_1, n_2, y = 1\rangle \to \langle w_1, n_2, y = 1\rangle \to \langle c_1, n_2, y = 0\rangle \to \\ \langle n_1, n_2, y = 1\rangle \to \langle n_1, w_2, y = 1\rangle \to \langle n_1, c_2, y = 0\rangle \to \cdots\end{aligned}$$

这个路径的迹是无限单词:

$$\mathrm{trace}(\pi) = \varnothing\varnothing\{\mathrm{crit}_1\}\varnothing\varnothing\{\mathrm{crit}_2\}\varnothing\varnothing\{\mathrm{crit}_1\}\varnothing\varnothing\{\mathrm{crit}_2\}\cdots$$

有限路径片段

$$\widehat{\pi} = \langle n_1, n_2, y=1\rangle \to \langle w_1, n_2, y=1\rangle \to \langle w_1, w_2, y=1\rangle \to \\ \langle w_1, c_2, y=0\rangle \to \langle w_1, n_2, y=1\rangle \to \langle c_1, n_2, y=0\rangle$$

的迹是

$$\mathrm{trace}(\widehat{\pi}) = \varnothing\varnothing\varnothing\{\mathrm{crit}_2\}\varnothing\{\mathrm{crit}_1\}\varnothing \qquad \blacksquare$$

### 3.2.3 线性时间性质

线性时间性质描述迁移系统应该呈现的迹. 通俗地说, 线性时间性质描述所考虑系统的容许 (或需要) 的行为. 下面给出这类性质的形式化定义. 这个定义是相当基本的, 由此可很好地理解什么是线性时间性质. 在第 5 章中, 将介绍逻辑形式化, 它确切地描述线性时间性质.

下面, 假设一个固定的命题集合 AP. 线性时间性质是对迁移系统的迹的要求. 可把这样的性质理解为对 AP 上的所有单词的要求, 并且定义为 (AP 上容许的) 单词的集合.

**定义 3.6** LT 性质

原子命题集合 AP 上的线性时间性质 (LT 性质) 就是 $(2^{\mathrm{AP}})^{\omega}$ 的子集. $\blacksquare$

这里, $(2^{\mathrm{AP}})^{\omega}$ 表示由 $2^{\mathrm{AP}}$ 中的元素作为字母无限连接后得到的单词的集合. 因此, LT 性质就是字母表 $2^{\mathrm{AP}}$ 上的无限单词构成的语言 (集合). 注意, 仅考虑无限单词 (而不考虑有限单词) 就够了, 因为考虑的是没有终止状态的迁移系统. 迁移系统对 LT 性质的满足性定义如下.

**定义 3.7** LT 性质的满足关系

令 $P$ 是 AP 上的 LT 性质, $\mathrm{TS} = (S, \mathrm{Act}, \to, I, \mathrm{AP}, L)$ 是没有终止状态的迁移系统. 若 $\mathrm{Traces(TS)} \subseteq P$, 则称 $\mathrm{TS} = (S, \mathrm{Act}, \to, I, \mathrm{AP}, L)$ 满足 $P$, 记作 $\mathrm{TS} \models P$. 若 $\mathrm{Traces}(s) \subseteq P$, 则称状态 $s \in S$ 满足 $P$, 记作 $s \models P$. $\blacksquare$

因而, 迁移系统满足 LT 性质 $P$ 是指其所有迹都遵守 $P$, 即, 其行为都是容许的. 状态满足 $P$ 是指从它开始的所有迹都满足 $P$.

**例 3.5** 交通灯

考虑两个简化的交通灯, 只有两种可能的设置: 红 (red) 和绿 (green). 设本例关心的原子命题是

$$\mathrm{AP} = \{\mathrm{red}_1, \mathrm{green}_1, \mathrm{red}_2, \mathrm{green}_2\}$$

下面考虑这些交通灯的两个 LT 性质并给出这两个 LT 性质包含的单词的例子. 首先, 考虑下述性质 $P$:

“第一个交通灯无限经常地亮绿灯”

这个 LT 性质就是 $2^{\mathrm{AP}}$ 上的形为 $A_0A_1A_2\cdots$ 的单词的集合, 每个单词都使得 $\mathrm{green}_1 \in A_i$ 对无穷多个 $i$ 成立. 例如, $P$ 包含无限单词

$$\{\mathrm{red}_1, \mathrm{green}_2\}\{\mathrm{green}_1, \mathrm{red}_2\}\{\mathrm{red}_1, \mathrm{green}_2\}\{\mathrm{green}_1, \mathrm{red}_2\}\cdots$$
$$\varnothing\{\mathrm{green}_1\}\varnothing\{\mathrm{green}_1\}\varnothing\{\mathrm{green}_1\}\varnothing\{\mathrm{green}_1\}\varnothing\cdots$$
$$\{\mathrm{red}_1, \mathrm{green}_1\}\{\mathrm{red}_1, \mathrm{green}_1\}\{\mathrm{red}_1, \mathrm{green}_1\}\{\mathrm{red}_1, \mathrm{green}_1\}\cdots$$
$$\{\mathrm{green}_1, \mathrm{green}_2\}\{\mathrm{green}_1, \mathrm{green}_2\}\{\mathrm{green}_1, \mathrm{green}_2\}\{\mathrm{green}_1, \mathrm{green}_2\}\cdots$$

无限单词 $\{\text{red}_1, \text{green}_1\}\{\text{red}_1, \text{green}_1\}\varnothing\varnothing\varnothing\varnothing\cdots$ 不在 $P$ 中, 因为它只包含有限个 $\text{green}_1$.

再考虑第二个 LT 性质 $P'$:

“永不同时亮绿灯”

这一性质由形为 $A_0A_1A_2\cdots$ 的无限单词的集合形式化, 其中, 对任意 $i \geqslant 0$ 都有 $\text{green}_1 \notin A_i$ 或 $\text{green}_2 \notin A_i$. 例如, 无限单词

$$\{\text{red}_1, \text{green}_2\}\{\text{green}_1, \text{red}_2\}\{\text{red}_1, \text{green}_2\}\{\text{green}_1, \text{red}_2\}\cdots$$
$$\varnothing\{\text{green}_1\}\varnothing\{\text{green}_1\}\varnothing\{\text{green}_1\}\varnothing\{\text{green}_1\}\varnothing\cdots$$
$$\{\text{red}_1, \text{green}_1\}\{\text{red}_1, \text{green}_1\}\{\text{red}_1, \text{green}_1\}\{\text{red}_1, \text{green}_1\}\cdots$$

在 $P'$ 中, 而无限单词 $\{\text{red}_1,\text{green}_2\}\{\text{green}_1, \text{green}_2\}\cdots$ 却不在 $P'$ 中.

图 3.7 所示的两个交通灯位于同一路口并且同步改变, 即, 若其中一个从红变绿, 则另一个就从绿变红. 这样, 两个交通灯就永远具有相反的颜色. 显然, 这两个交通灯同时满足 $P$ 和 $P'$. 但是, 两个完全独立地改变颜色的交通灯就既不满足 $P$ (不能保证第一个交通灯无限经常地亮绿灯) 也不满足 $P'$. ■

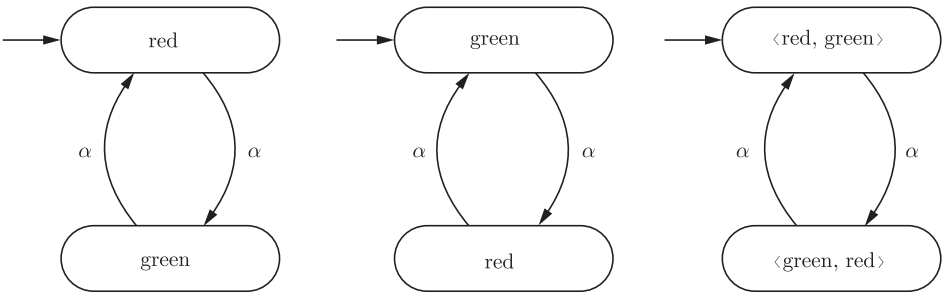


图 3.7　完全同步的两个交通灯 (左和中) 及其并行合成 (右)

通常, 线性性质不会涉及迁移系统中出现的全部原子命题, 而只是较小的一个子集. 对于命题集合 $\text{AP}' \subseteq \text{AP}$ 上的性质 $P$, 只有 $\text{AP}'$ 中的标记是相关的. 令 $\hat{\pi}$ 是迁移系统 TS 的有限路径片段. 用 $\text{trace}_{\text{AP}'}(\hat{\pi})$ 表示只考虑 $\text{AP}'$ 中的原子命题的 $\hat{\pi}$ 的有限迹. 对应地, $\text{trace}_{\text{AP}'}(\pi)$ 表示无限路径片段 $\pi$ 的仅限于 $\text{AP}'$ 中的原子命题的迹. 从而, 对于 $\pi = s_0s_1s_2\cdots$, 我们有

$$\text{trace}_{\text{AP}'}(\pi) = L'(s_0)L'(s_1)\cdots = (L(s_0) \cap \text{AP}')(L(s_1) \cap \text{AP}')\cdots$$

令 $\text{Traces}_{\text{AP}'}(\text{TS})$ 表示迹的集合 $\text{trace}_{\text{AP}'}(\text{Paths}(\text{TS}))$. 当从上下文中清楚原子命题的 $\text{AP}'$ 时, 就省略下标 $\text{AP}'$. 在本章剩余部分中, 到原子命题的一个相关集合的限制通常是隐式给出的.

**例 3.6**　*互斥性质*

在第 2 章中, 已考虑几种互斥算法. 为指明互斥性质 “总是至多有一个进程处于其关键节段”, 只需考虑原子命题 $\text{crit}_1$ 和 $\text{crit}_2$ 就够了. 其他原子命题与此性质无任何关系. 互斥性质的形式化由 LT 性质

$$P_{\text{mutex}} = \Big\{A_0A_1A_2\cdots \mid \forall 0 \leqslant i.\, \{\text{crit}_1, \text{crit}_2\} \not\subseteq A_i\Big\}$$

给出. 例如, 下面三个无限单词

$$\{\mathrm{crit}_1\}\{\mathrm{crit}_2\}\{\mathrm{crit}_1\}\{\mathrm{crit}_2\}\{\mathrm{crit}_1\}\{\mathrm{crit}_2\}\cdots$$
$$\{\mathrm{crit}_1\}\{\mathrm{crit}_1\}\{\mathrm{crit}_1\}\{\mathrm{crit}_1\}\{\mathrm{crit}_1\}\{\mathrm{crit}_1\}\cdots$$
$$\varnothing\varnothing\varnothing\varnothing\varnothing\varnothing\cdots$$

都包含在 $P_{\mathrm{mutex}}$ 中. 但它不包含形为

$$\{\mathrm{crit}_1\}\varnothing\{\mathrm{crit}_1,\mathrm{crit}_2\}\cdots$$

的单词. 例 2.12 中描述的迁移系统 $\mathrm{TS}_{\mathrm{Arb}}=(\mathrm{TS}_1\,|||\,\mathrm{TS}_2)\parallel \mathrm{Arbiter}$ 满足互斥性质, 即

$$\mathrm{TS}_{\mathrm{Arb}}\models P_{\mathrm{mutex}}$$

留给读者证明基于信号的互斥算法 (见图 3.6) 和 Peterson 算法 (见例 2.11) 满足互斥性质. ■

**例 3.7** *无饥饿*

保证互斥是互斥算法的重要性质, 但不是唯一相关的性质. 永不允许进程进入关键节段的算法可以保证互斥, 这当然不是我们想要的. 应该施加一个性质, 它使得想要进入关键节段的进程最终总能进入. 这个性质可防止进程无限等待, 并可形式化地描述为 LT 性质

$$P_{\mathrm{finwait}}=\Big\{A_0A_1A_2\cdots\mid \forall i\in\{1,2\}.\,(\forall j.\,\mathrm{wait}_i\in A_j\Rightarrow\exists k>j.\,\mathrm{crit}_i\in A_k)\Big\}$$

这里假定命题的集合为

$$\mathrm{AP}=\{\mathrm{wait}_1,\mathrm{crit}_1,\mathrm{wait}_2,\mathrm{crit}_2\}$$

性质 $P_{\mathrm{finwait}}$ 表示, 对任一进程, 只要它等待就会进入关键节段. 即, 任一进程在进入它的关键节段前只需要有限的等待. 它不表示一个进程经常等待并经常进入关键节段.

考虑下面的变量. LT 性质 $P_{\mathrm{nostarve}}$ 是无限单词 $A_0A_1A_2\cdots$ 的集合, 它对每一 $i\in\{1,2\}$ 都有

$$(\forall k\geqslant 0.\,\exists j\geqslant k.\,\mathrm{wait}_i\in A_j)\Rightarrow(\forall k\geqslant 0.\,\exists j\geqslant k.\,\mathrm{crit}_i\in A_j)$$

把它缩写为

$$\Big(\overset{\infty}{\exists} j.\,\mathrm{wait}_i\in A_j\Big)\Rightarrow\Big(\overset{\infty}{\exists} j.\,\mathrm{crit}_i\in A_j\Big)$$

其中, $\overset{\infty}{\exists}$ 表示 "存在无穷多个".

性质 $P_{\mathrm{nostarve}}$ 表示, 如果两个进程无限经常地等待, 那么它们中的每个进程都无限经常地进入关键节段. 但是, 基于信号的解决方案不满足这个自然的要求, 因为

$$\varnothing(\{\mathrm{wait}_2\}\{\mathrm{wait}_1,\mathrm{wait}_2\}\{\mathrm{crit}_1,\mathrm{wait}_2\})^{\omega}$$

是迁移系统的可能的迹, 但不属于 $P_{\mathrm{nostarve}}$. 此迹表示如下执行: 只有第一个进程无限经常地进入其关键节段. 事实上, 第二个进程无限等待进入关键节段.

Peterson 算法的迁移系统 (见例 2.11) 却满足 $P_{\mathrm{nostarve}}$. 请读者证之. ■

### 3.2.4　迹等价与线性时间性质

LT 性质描述迁移系统应该呈现的 (无限) 迹. 如果迁移系统 TS 和 TS′ 具有相同的迹, 我们就会期望它们满足同样的 LT 性质. 很明显, 如果 TS $\models P$, 那么 TS 的所有迹都包含于 $P$, 并且当 Traces(TS) = Traces(TS′) 时, TS′ 的迹同样包含于 $P$; 否则, 只要 TS $\not\models P$, 那么 Traces(TS) 中就存在被 $P$ 排斥的一个迹, 即, 它不包含于迹的集合 $P$ 中. 因为 Traces(TS) = Traces(TS′), TS′ 也含有这一被排斥的迹, 因此, TS′ $\not\models P$. 迹等价、迹包含与 LT 性质的满足性之间的关系是本节的主题.

本节从迹包含及其对并发系统设计的重要性开始. 迁移系统 TS 与 TS′ 之间的迹包含要求 TS 展现的所有的迹也都是 TS′ 展现的, 即 Traces(TS) $\subseteq$ Traces(TS′). 注意, TS′ 可能会展现更多的迹, 即可能会有 TS 中没有的一些 (线性时间) 行为. 在系统分步设计中, 其设计是逐步细化的, 迹包含经常在以下意义上看作实现关系:

$$\text{Traces(TS)} \subseteq \text{Traces(TS}') \text{ 意为 “TS 是 TS}' \text{ 的正确实现”}$$

例如, 令 TS′ 是一个 (更抽象的) 设计, 其并行合成用交错建模; TS 是它的实现, 其中 (某些) 交错已被一些调度策略解决. 因而 TS 可看作 TS′ 的一个实现, 并且显然 Traces(TS) $\subseteq$ Traces(TS′).

迹包含对 LT 性质有什么用? 定理 3.1 表明迹包含与以 LT 性质表示的需求准述是一致的.

**定理 3.1**　*迹包含与 LT 性质*

令 TS 和 TS′ 是没有终止状态而具有相同命题集合 AP 的迁移系统. 那么, 下列两个命题是等价的:

(a) Traces(TS) $\subseteq$ Traces(TS′).

(b) 对于任何 LT 性质 $P$: TS′ $\models P$ 蕴涵 TS $\models P$.

**证明**: 先证 (a) $\Rightarrow$ (b). 假设 Traces(TS) $\subseteq$ Traces(TS′), 令 $P$ 是一个 LT 性质并且 TS′ $\models P$. 由定义 3.7 可得, Traces(TS′) $\subseteq P$. 因为 Traces(TS) $\subseteq$ Traces(TS′), 所以 Traces(TS) $\subseteq P$. 再由定义 3.7 得 TS $\models P$.

再证 (b) $\Rightarrow$ (a). 假设对所有 LT 性质, TS′ $\models P$ 蕴涵 TS $\models P$ 都成立. 令 $P$ = Traces(TS′). 显然, Traces(TS′) $\subseteq$ Traces(TS′), 故 TS′ $\models P$. 由假设可得 TS $\models P$, 因此 Traces(TS) $\subseteq$ Traces(TS′). ■

这个简单的观察对逐步细化设计起决定作用. 如果 TS′ 是初步设计的迁移系统表示, TS 是从 TS′ 的细化 (即更详细的设计) 而来的迁移系统, 那么, 从关系 Traces(TS) $\subseteq$ Traces(TS′) 可直接得到而不必再显式地证明: 对 TS′ 成立的性质对 TS 也成立.

**例 3.8**　*细化基于信号的互斥算法*

令 TS′ = $\text{TS}_{\text{Sem}}$ 是基于信号的互斥算法的迁移系统表示 (见图 3.6), TS 是从 TS′ 删除迁移

$$\langle \text{wait}_1, \text{wait}_2, y = 1 \rangle \to \langle \text{wait}_1, \text{crit}_2, y = 0 \rangle$$

后得到的迁移系统. 换言之, 第二进程 ($P_2$) 不能再从两个进程都等待的状态进入关键节段. 这样将产生一个模型, 它在两个进程争入关键节段时让 $P_1$ 比 $P_2$ 有更高的优先权. 由于移除了一个迁移, 因此可得出 $\text{Traces(TS)} \subseteq \text{Traces(TS}')$. 又因为 TS′ 保证互斥, 即 $\text{TS}' \models P_{\text{mutex}}$, 故由定理 3.1 得 $\text{TS} \models P_{\text{mutex}}$. ■

如果两个迁移系统的迹集相同, 则称它们为迹等价的.

**定义 3.8** 迹等价

若 $\text{Traces}_{\text{AP}}(\text{TS}) = \text{Traces}_{\text{AP}}(\text{TS}')$, 则称迁移系统 TS 和 TS′ 关于命题集合 AP 是迹等价的. [①] ■

定理 3.1 蕴涵着两个迹等价的迁移系统关于表述为 LT 性质的需求的等价性.

**推论** 迹等价与 LT 性质

令 TS 和 TS′ 是没有终止状态而具有相同原子命题集合 AP 的迁移系统. 那么,

$$\text{Traces(TS)} = \text{Traces(TS}') \iff \text{TS 与 TS}'\text{满足相同的 LT 性质} \quad ■$$

因此, 不存在可以区分两个迹等价的迁移系统的 LT 性质. 换言之, 为了确认迁移系统 TS 和 TS′ 不是迹等价的, 只要找到对其中一个成立而对另一个不成立的 LT 性质就够了.

**例 3.9** 两台饮料售货机

考虑图 3.8 中的两个迁移系统, 它们都是饮料售货机的模型. 为简单起见, 省略了迁移的可见动作标记. 两台机器都可提供苏打水和啤酒. 以左边的迁移系统为模型的饮料售货机在投入硬币后未定地选择苏打水或啤酒; 但是, 右边的系统有两个选择按钮 (每个对应一种饮料), 并在投入硬币后未定地阻塞一个按钮. 在这两个迁移系统中, 用户不能控制得到的饮料, 售货机完全控制饮料选择.

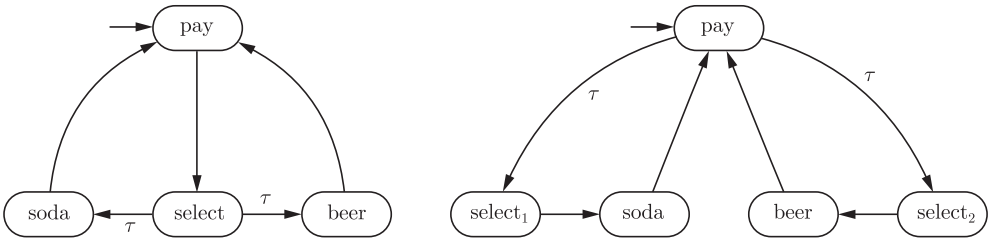


图 3.8 两台饮料售货机的迁移系统

令 $\text{AP} = \{\text{pay}, \text{soda}, \text{beer}\}$. 尽管两台售货机的行为不同, 但不难看出, 当考虑 AP 时它们表现出相同的迹, 因为这两台机器的迹都是 pay 与 soda 或 beer 之一的交替序列. 因此, 两台售货机是迹等价的. 由上面的推论可知, 两台售货机恰好满足同样的 LT 性质. 换言之, 这意味着不存在区分两台售货机的 LT 性质. ■

## 3.3 安全性质与不变式

安全性质经常被说成是 “坏事不发生”. 互斥性质 (总是至多只有一个进程处于其关键节段) 是一个典型的安全性质. 它说明坏事 (有两个或更多进程同时处于关键节段) 永不发

① 此处假设两个迁移系统的命题集合都包含 AP.

生. 另一个典型的安全性质是无死锁. 例如, 对于哲学家就餐问题 (见例 3.2), 所谓死锁可以刻画为所有哲学家都在等待拿起第二根筷子的情形. 这一坏事 (即不想要的情形) 应该永不发生.

### 3.3.1　不变式

事实上, 上面讨论的是一种特殊的安全性质: 它们是*不变式*. 不变式是由状态的条件 $\Phi$ 给出的 LT 性质, 其中 $\Phi$ 对所有可达状态都成立.

**定义 3.9**　*不变式*

对于 AP 上的 LT 性质 $P_{\text{inv}}$, 若存在 AP 上的命题逻辑公式 [①] $\Phi$ 使得

$$P_{\text{inv}} = \left\{ A_0 A_1 A_2 \cdots \in (2^{\text{AP}})^\omega \mid \forall j \geqslant 0.\, A_j \models \Phi \right\}$$

则称 $P_{\text{inv}}$ 为*不变式*, 称 $\Phi$ 为 $P_{\text{inv}}$ 的*不变条件* (或*状态条件*). ■

注意,

$$\begin{aligned} \text{TS} \models P_{\text{inv}} \quad & \text{iff} \quad \text{对 TS 中的所有路径 } \pi,\ \text{trace}(\pi) \in P_{\text{inv}} \\ & \text{iff} \quad \text{对于所有属于 TS 的某条路径的 } s,\ L(s) \models \Phi \\ & \text{iff} \quad \text{对于所有 } s \in \text{Reach(TS)},\ L(s) \models \Phi \end{aligned}$$

因而, 概念 "不变式" 可解释为: 在给定的迁移系统中, 所有初始状态必须满足条件 $\Phi$, 并且 $\Phi$ 的可满足性在给定迁移系统的可达部分的迁移下是不变的. 后一句意为：如果 $\Phi$ 对迁移 $s \xrightarrow{\alpha} s'$ 的源状态 $s$ 成立, 则 $\Phi$ 对目标 $s'$ 也成立.

回到互斥和哲学家就餐无死锁的例子. 互斥性质可用使用命题逻辑公式

$$\Phi = \neg\text{crit}_1 \vee \neg\text{crit}_2$$

的不变式描述. 至于哲学家就餐的无死锁, 不变式要保证至少有一位哲学家不等待拿起筷子. 这可用命题公式

$$\Phi = \neg\text{wait}_0 \vee \neg\text{wait}_1 \vee \neg\text{wait}_2 \vee \neg\text{wait}_3 \vee \neg\text{wait}_4$$

建立. 其中, 命题 $\text{wait}_i$ 刻画哲学家 $i$ 正等待一根筷子的状态.

如何判断迁移系统是否满足不变式? 如果迁移系统 TS 是*有限的*, 稍加改造的标准的图遍历算法即可, 例如深度优先搜索 (Depth First Search, DFS) 或广度优先搜索 (Breadth First Search, BFS) 等, 这是由于对命题公式 $\Phi$ 验证不变式等同于验证 $\Phi$ 对从初始状态可达的每个状态的有效性.

算法 3.1 概括了对状态图 $G(\text{TS})$ 用深度优先前向搜索方法验证不变条件 $\Phi$ 的主要步骤. *前向搜索*的意思是, 从初始状态开始并检查所有可达状态. 如果至少有一个访问到的状态使 $\Phi$ 不成立, 那么由 $\Phi$ 诱导的不变性就被打破. 在算法 3.1 中, $R$ 存储所有已访问的状态, 即, 如果算法 3.1 结束, 那么 $R = \text{Reach(TS)}$ 包含所有可达状态. 此外, $U$ 是堆栈, 它

① 命题逻辑的基本原理在附录 A 的 A.3 节中给出.

用于组织仍然需要访问但还不在 $R$ 中的状态. 操作 push、pop、top 是堆栈上的标准操作. 符号 $\varepsilon$ 用于表示空堆栈. 也可以使用后向搜索方法, 它从 $\Phi$ 不成立的状态开始, (通过 DFS 或 BFS) 计算集合 $\bigcup_{s\in S, s\not\models\Phi} \mathrm{Pre}^*(s)$.

**算法 3.1** 使用深度优先前向搜索的朴素不变式检验

输入: 有限迁移系统 TS 和命题公式 $\Phi$.

输出: 若 TS 满足不变式 "总是 $\Phi$" 则为 true, 否则为 false.

```
set R := ∅;                                   (* 已访问状态的集合 *)
stack U := ε;                                 (* 空堆栈 *)
bool b := true;                               (* R 中的状态都满足 Φ *)
for all s ∈ I do
  if s ∉ R then
    visit(s)                                  (* 对未访问的初始状态进行深度优先前向搜索 *)
  fi
od
return b
procedure visit(state s)
  push(s, U);                                 (* s 入栈 *)
  R := R ∪ {s};                               (* s 标记为可达的 *)
  repeat
    s' := top(U);
    if Post(s') ⊆ R then
      pop(U);
      b := b ∧ (s' ⊨ Φ);                      (* 检验 Φ 在 s 处的有效性 *)
    else
      let s'' ∈ Post(s') \ R
      push(s'', U);
      R := R ∪ {s''};                         (* 状态 s'' 是一个新可达状态 *)
    fi
  until (U = ε)
endproc
```

可对算法 3.1 稍加改进, 一旦遇到不满足 $\Phi$ 的状态就退出计算. 这一状态是 "坏" 状态, 因为它让迁移系统否定了不变式, 并可将其作为错误标识返回. 然而, 这样的错误标识并不是很有用.

相反, 更有用的是初始路径片段 $s_0 s_1 \cdots s_n$, 其中所有状态 (除最后一个外) 都满足 $\Phi$, 而 $s_n \not\models \Phi$. 这样的路径片段标志着违反不变式的迁移系统的可能行为. 可以很容易地修改算法 3.1, 使它在遇到违反 $\Phi$ 的状态时能提供反例. 为此, 使用 (深度优先搜索) 堆栈 $U$. 当遇到违反 $\Phi$ 的 $s_n$ 时, 栈内容 (从底到顶读) 包含需要的起始路径片段. 算法 3.2 由此而来.

**算法 3.2**　使用深度优先前向搜索的不变式检验

输入: 有限迁移系统 TS 和命题公式 $\Phi$.

输出: 若 TS 满足不变式 “总是 $\Phi$” 则为 “是”, 否则为 “否” 和反例.

```
set R := ∅;                                          (* 可达状态的集合 *)
stack U := ε;                                        (* 空堆栈 *)
bool b := true;                                      (* R 中的状态都满足 Φ *)
while (I \ R ≠ ∅ ∧ b) do
  let s ∈ I \ R;                                     (* 选择任一不在 R 中的初始状态 *)
  visit(s)                                           (* 对未访问的初始状态进行深度优先前向搜索 *)
od
if b then
  return (''是'')                                    (* TS ⊨ “总是 Φ” *)
else
  return (''否'', reverse(U))                        (* 从堆栈内容得到的反例 *)
fi
```

```
procedure visit(state s)
  push(s, U);                                        (* s 入堆栈 *)
  R := R ∪ {s};                                      (* s 标记为可达的 *)
  repeat
    s' := top(U);
    if Post(s') ⊆ R then
      pop(U);
      b := b ∧ (s' ⊨ Φ);                             (* 检验 Φ 在 s 处的有效性 *)
    else
      let s'' ∈ Post(s') \ R
      push(s'', U);
      R := R ∪ {s''};                                (* 状态 s'' 是一个新可达状态 *)
    fi
  until ((U = ε) ∨ ¬b)
endproc
```

上面给出的不变性检验算法在最坏情况下的时间复杂度由访问所有可达状态的深度优先搜索算法的开销主导. 这个开销是状态 (状态图中的节点) 数和迁移 (状态图中的边) 数的线性函数, 假如在给定的状态图表示中, 在时间 $\Theta(|\mathrm{Post}(s)|)$ 内可遇到任何状态 $s$ 的直接后继 $s' \in \mathrm{Post}(s)$. 这对集合 $\mathrm{Post}(s)$ 的邻接列表表示是成立的. 在本节讨论的情境中, 必须分析复杂系统的状态图, 邻接列表的显式表示是不适当的; 相反, 一般是隐式地给出邻接列表, 例如, 利用并发进程的句法描述, 类似于程序图或像 nanoPromela 那样的带有程序图语义的高级描述语言 (参见 2.2.5节). 那么, 状态 $s$ 的直接后继由复合系统的迁移关系的公理和规则得到. 除进程的句法描述占用的空间外, 算法 3.2 占用的空间由已访问状态的集合 $R$ 的表示 (通常会用适当的哈希技术实现) 和堆栈 $U$ 主导. 因此, 不变式检验的空间复杂度是可达状态数的线性函数.

**定理 3.2**　不变式检验的时间复杂度

算法 3.2 的时间复杂度是 $O(N(1+|\Phi|)+M)$ 其中, $N$ 表示可达状态数, $M = \sum_{s \in S} |\mathrm{Post}(s)|$

表示 TS 的可达部分的迁移数.

**证明**: 状态图 $G(\mathrm{TS})$ 上的前向可达性的时间复杂度是 $O(N+M)$. 对某个状态 $s$ 验证 $s \models \Phi$ 需要的时间是 $\Phi$ 的长度的线性函数 ①. 因为对每个状态 $s$ 都要验证 $\Phi$ 是否成立, 这相当于总共 $N+M+N(1+|\Phi|)$ 个操作. ■

### 3.3.2 安全性质

如 3.3.1 节所述, 不变式可视为状态性质并可通过考虑可达状态来检验. 但是, 某些安全性质可能对有限路径片段施加要求, 不能仅通过考虑可达状态来验证. 为此, 考虑取款机的例子. 一个很自然的要求是, 只有提供正确的个人标识 (PIN) 后才可取款. 这个性质不是不变式, 因为它不是一个状态性质. 但是可以认为它是一个安全性质, 因为任何违反这一要求的无限运行都有一个有限的坏前缀, 即在输入 PIN 前取走了现金.

形式上, 安全性质 $P$ 定义为 AP 上满足下列条件的 LT 性质: 使 $P$ 不成立的任何无限单词 $\sigma$ 都包含一个坏前缀. 坏前缀的含义是坏事已经发生的有限前缀 $\hat{\sigma}$, 因此, 以该前缀开始的无限单词不可能满足 $P$.

**定义 3.10** 安全性质和坏前缀

设 $P_{\mathrm{safe}}$ 是 AP 上的 LT 性质. 如果对所有单词 $\sigma \in (2^{\mathrm{AP}})^{\omega} \setminus P_{\mathrm{safe}}$ 都存在 $\sigma$ 的一个有限前缀 $\hat{\sigma}$ 使得

$$P_{\mathrm{safe}} \cap \left\{ \sigma' \in (2^{\mathrm{AP}})^{\omega} \mid \hat{\sigma} \text{ 是 } \sigma' \text{ 的前缀} \right\} = \varnothing$$

则称 $P_{\mathrm{safe}}$ 是安全性质. 这样的有限单词 $\hat{\sigma}$ 称为 $P_{\mathrm{safe}}$ 的坏前缀. $P_{\mathrm{safe}}$ 的极小坏前缀是指这样一个坏前缀 $\hat{\sigma}$: 它的任何一个真前缀都不是 $P_{\mathrm{safe}}$ 的坏前缀. 换言之, 极小坏前缀就是最小长度的坏前缀. $P_{\mathrm{safe}}$ 的所有坏前缀的集合记为 $\mathrm{BadPref}(P_{\mathrm{safe}})$, 所有极小坏前缀的集合记为 $\mathrm{MinBadPref}(P_{\mathrm{safe}})$. ■

任何不变式都是安全性质. 对于 AP 上的命题公式 $\Phi$ 和它的不变式 $P_{\mathrm{inv}}$, 形为

$$A_0 A_1 \cdots A_n \in (2^{\mathrm{AP}})^{+}$$

且 $A_0 \models \Phi, A_1 \models \Phi, \cdots, A_{n-1} \models \Phi$ 而 $A_n \not\models \Phi$ 的所有有限单词组成 $P_{\mathrm{inv}}$ 的极小坏前缀的集合. 下面的两个例子说明某些安全性质并非不变式.

**例 3.10** 交通灯的一个安全性质

考虑通常的具有红、绿、黄 3 个阶段的交通灯. 红灯之前必黄灯, 这个要求是安全性质但并非不变式. 证明如下.

令 red、yellow 和 green 是原子命题. 直观地, 它们用于表示红灯、黄灯和绿灯阶段的状态. 性质 “永远至少一灯亮” 描述为

$$\{\sigma = A_0 A_1 A_2 \cdots \mid A_j \subseteq \mathrm{AP} \wedge A_j \neq \varnothing\}$$

坏前缀就是那些包含 $\varnothing$ 的有限单词. 极小坏前缀以 $\varnothing$ 结束. 性质 “两灯永远不会同时亮” 描述为

$$\{\sigma = A_0 A_1 A_2 \cdots \mid A_j \subseteq \mathrm{AP} \wedge |A_j| \leqslant 1\}$$

① 为了涵盖 $\Phi$ 是原子命题的情况, 此时设 $|\Phi| = 0$; 验证 $\Phi$ 对状态 $s$ 是否成立的耗时用 $1 + |\Phi|$ 处理.

该性质的坏前缀是含有 {red, green}、{red, yellow} 等类似集合的单词. 极小坏前缀以这样的集合结束.

现在, 令 $\mathrm{AP}' = \{\mathrm{red}, \mathrm{yellow}\}$. 性质 “红灯之前必为黄灯” 可描述为无限单词 $\sigma = A_0A_1A_2\cdots$ 的集合, 其中 $A_i \subseteq \{\mathrm{red}, \mathrm{yellow}\}$, 并且对所有 $i \geqslant 0$ 都有

$$\mathrm{red} \in A_i \text{ 蕴涵 } i > 0 \text{ 且 } \mathrm{yellow} \in A_{i-1}$$

坏前缀就是违反这一条件的有限单词. 极小坏前缀的例子有

$$\varnothing\varnothing\{\mathrm{red}\} \text{ 和 } \varnothing\{\mathrm{red}\}$$

坏前缀

$$\{\mathrm{yellow}\}\{\mathrm{yellow}\}\{\mathrm{red}\}\{\mathrm{red}\}\varnothing\{\mathrm{red}\}$$

不是极小的, 因为它的真前缀 $\{\mathrm{yellow}\}\{\mathrm{yellow}\}\{\mathrm{red}\}\{\mathrm{red}\}$ 也是坏前缀.

在能够构成正则语言的意义上, 该安全性质的极小坏前缀是正则的. 图 3.9 中的有限自动机恰好接收上述安全性质的极小坏前缀. [①] 此处, ¬yellow 应理解为 $\varnothing$ 或 {red}. 注意, 本例给出的其他性质也是正则的. ■

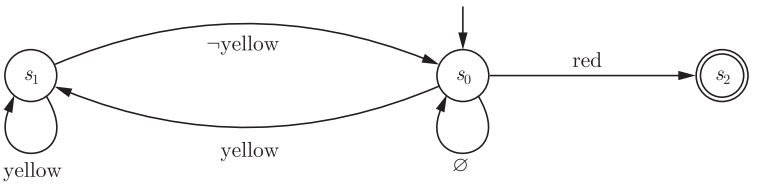


图 3.9　正则安全性质的极小坏前缀的有限自动机

**例 3.11**　饮料售货机的安全性质

对于饮料售货机, 一个自然的安全性质是

“投币数至少是提交饮料数”

通过使用命题集合 $\{\mathrm{pay}, \mathrm{drink}\}$ 以及标记函数, 这一性质可用无限单词 $A_0A_1A_2\cdots$ 的集合形式化. 其中, 对于所有 $i \geqslant 0$ 都有

$$|\{0 \leqslant j \leqslant i \mid \mathrm{pay} \in A_j\}| \geqslant |\{0 \leqslant j \leqslant i \mid \mathrm{drink} \in A_j\}|$$

该安全性质的坏前缀的例子有

$$\varnothing\{\mathrm{pay}\}\{\mathrm{drink}\}\{\mathrm{drink}\}$$
$$\varnothing\{\mathrm{pay}\}\{\mathrm{drink}\}\varnothing\{\mathrm{pay}\}\{\mathrm{drink}\}\{\mathrm{drink}\}$$

请读者证明, 图 3.8 中的两台饮料售货机都满足上述安全性质. ■

安全性质是对有限迹的要求. 这在形式上用以下引理表述.

① 4.1 节将简要介绍作为有限单词上的语言接收器的有限自动机的主要概念.

**引理 3.1** 安全性质的满足关系

对于无终止状态的迁移系统 TS 和安全性质 $P_{\text{safe}}$:

$$\text{TS} \models P_{\text{safe}} \text{ 当且仅当 } \text{Traces}_{\text{fin}}(\text{TS}) \cap \text{BadPref}(P_{\text{safe}}) = \varnothing$$

**证明**: 先证充分性, 用反证法. 令 $\text{Traces}_{\text{fin}}(\text{TS}) \cap \text{BadPref}(P_{\text{safe}}) = \varnothing$ 并假设 $\text{TS} \not\models P_{\text{safe}}$. 那么, 对 TS 中的某个路径 $\pi$ 有 $\text{trace}(\pi) \notin P_{\text{safe}}$. 因此, $\text{trace}(\pi)$ 从 $P_{\text{safe}}$ 的一个坏前缀 $\widehat{\sigma}$ 开始. 然而, 这样就得到 $\widehat{\sigma} \in \text{Traces}_{\text{fin}}(\text{TS}) \cap \text{BadPref}(P_{\text{safe}})$. 矛盾.

再证必要性, 用反证法. 假设 $\text{TS} \models P_{\text{safe}}$ 且 $\widehat{\sigma} \in \text{Traces}_{\text{fin}}(\text{TS}) \cap \text{BadPref}(P_{\text{safe}})$. 可把有限迹 $\widehat{\sigma} = A_1A_2\cdots A_n \in \text{Traces}_{\text{fin}}(\text{TS})$ 扩充为无限迹 $\sigma = A_1A_2\cdots A_nA_{n+1}A_{n+2}\cdots \in \text{Traces}(\text{TS})$. 那么, $\sigma \notin P_{\text{safe}}$. 因此, $\text{TS} \not\models P_{\text{safe}}$. ■

最后给出用闭包刻画的安全性质的另一种特征.

**定义 3.11** 前缀与闭包

对于迹 $\sigma \in (2^{\text{AP}})^{\omega}$, 令 $\text{pref}(\sigma)$ 表示 $\sigma$ 的有限前缀的集合, 即

$$\text{pref}(\sigma) = \{\widehat{\sigma} \in (2^{\text{AP}})^{*} \mid \widehat{\sigma} \text{ 是 } \sigma \text{ 的有限前缀}\}$$

即, 如果 $\sigma = A_0A_1A_2\cdots$, 那么 $\text{pref}(\sigma) = \{\varepsilon, A_0, A_0A_1, A_0A_1A_2, \cdots\}$ 是有限单词的集合. 此概念可用常规方法提升到迹集. 对于 AP 上的性质 $P$:

$$\text{pref}(P) = \bigcup_{\sigma \in P} \text{pref}(\sigma)$$

LT 性质 $P$ 的闭包定义为

$$\text{closure}(P) = \{\sigma \in (2^{\text{AP}})^{\omega} \mid \text{pref}(\sigma) \subseteq \text{pref}(P)\}$$ ■

例如, 对于无限迹 $\sigma = ABABAB\cdots$ (其中, $A, B \subseteq \text{AP}$), 我们有 $\text{pref}(\sigma) = \{\varepsilon, A, AB, ABA, ABAB, \cdots\}$, 此即由正则表达式 $(AB)^{*}(A + \varepsilon)$ 给出的正则语言.

LT 性质 $P$ 的闭包是一些无限迹的集合, 这些迹的有限前缀必须是 $P$ 的有限前缀. 换言之, $P$ 的闭包中的无限迹不会有不是 $P$ 的前缀的前缀. 正如将要看到的那样, 闭包是刻画安全与活性性质的核心概念.

**引理 3.2** 安全性质的替代特征

令 $P$ 是 AP 上的线性性质. 那么, $P$ 是安全性质当且仅当 $\text{closure}(P) = P$.

**证明**: 先证充分性. 设 $\text{closure}(P) = P$. 为证明 $P$ 是安全性质, 取一个元素 $\sigma \in (2^{\text{AP}})^{\omega} \setminus P$ 并证明 $\sigma$ 从 $P$ 的坏前缀开始. 因为 $\sigma \notin P = \text{closure}(P)$, 所以存在 $\sigma$ 的有限前缀 $\widehat{\sigma} \notin \text{pref}(P)$. 由 $\text{pref}(P)$ 的定义, 满足 $\widehat{\sigma} \in \text{pref}(\sigma')$ 的 $\sigma' \in (2^{\text{AP}})^{\omega}$ 不属于 $P$. 因此, $\widehat{\sigma}$ 是 $P$ 的坏前缀, 并由定义可知 $P$ 是安全性质.

再证必要性. 假设 $P$ 是安全性质. 需要证明 $P = \text{closure}(P)$. 包含关系 $P \subseteq \text{closure}(P)$ 对任何 LT 性质都成立. 还要证明 $\text{closure}(P) \subseteq P$. 用反证法证明. 假设有某个 $\sigma = A_1A_2A_3\cdots \in \text{closure}(P) \setminus P$. 因为 $P$ 是安全性质并且 $\sigma \notin P$, 所以 $\sigma$ 有一个有限前缀

$$\widehat{\sigma} = A_1A_2\cdots A_n \in \text{BadPref}(P)$$

由于 $\sigma \in \text{closure}(P)$, 有 $\widehat{\sigma} \in \text{pref}(\sigma) \subseteq \text{pref}(P)$. 因此, 存在单词 $\sigma' \in P$, 其形式为

$$\sigma' = \underbrace{A_1 A_2 \cdots A_n}_{\text{坏前缀}} B_{n+1} B_{n+2} \cdots$$

这与 $P$ 是安全性质矛盾. ■

### 3.3.3　迹等价与安全性质

迁移系统的迹包含与 LT 性质的满足性之间存在紧密联系 (见定理 3.1): 对于没有终止状态的迁移系统 TS 和 TS′,

$$\text{Traces}(\text{TS}) \subseteq \text{Traces}(\text{TS}') \quad \text{当且仅当} \quad \begin{array}{l}\text{对所有 LT 性质 } P \\ \text{TS}' \models P \text{ 蕴涵 } \text{TS} \models P\end{array}$$

注意, 这一结果考虑了所有无限迹. 因此, 它阐述了迁移系统的无限迹与 LT 性质的有效性之间的关系. 当替换无限迹只考虑有限迹时, 可建立安全性质的有效性的类似联系, 如定理 3.3 所述.

**定理 3.3**　有限迹包含与安全性质

令 TS 与 TS′ 是没有终止状态并有相同命题集合 AP 的迁移系统. 那么, 以下命题等价:

(a) $\text{Traces}_{\text{fin}}(\text{TS}) \subseteq \text{Traces}_{\text{fin}}(\text{TS}')$.

(b) 对于任何安全性质 $P_{\text{safe}}$, $\text{TS}' \models P_{\text{safe}}$ 蕴涵 $\text{TS} \models P_{\text{safe}}$.

**证明**:

先证 (a) ⇒ (b). 假设 $\text{Traces}_{\text{fin}}(\text{TS}) \subseteq \text{Traces}_{\text{fin}}(\text{TS}')$, 同时令 $P_{\text{safe}}$ 是安全性质并且 $\text{TS}' \models P_{\text{safe}}$. 由引理 3.1 有 $\text{Traces}_{\text{fin}}(\text{TS}') \cap \text{BadPref}(P_{\text{safe}}) = \varnothing$, 由此可知, $\text{Traces}_{\text{fin}}(\text{TS}) \cap \text{BadPref}(P_{\text{safe}}) = \varnothing$. 再次使用引理 3.1 得 $\text{TS} \models P_{\text{safe}}$.

再证 (b) ⇒ (a). 假设 (b) 成立. 令 $P_{\text{safe}} = \text{closure}(\text{Traces}(\text{TS}'))$. 那么, $P_{\text{safe}}$ 是安全性质并且有 $\text{TS}' \models P_{\text{safe}}$ (见习题 3.9). 因此, 由 (b) 推出 $\text{TS} \models P_{\text{safe}}$, 即

$$\text{Traces}(\text{TS}) \subseteq \text{closure}(\text{Traces}(\text{TS}'))$$

可由此推导出

$$\begin{aligned}\text{Traces}_{\text{fin}}(\text{TS}) &= \text{pref}(\text{Traces}(\text{TS})) \\ &\subseteq \text{pref}(\text{closure}(\text{Traces}(\text{TS}')) \\ &= \text{pref}(\text{Traces}(\text{TS}')) \\ &= \text{Traces}_{\text{fin}}(\text{TS}')\end{aligned}$$

这里使用了如下性质: 对于任意 $P$ 都有 $\text{pref}(\text{closure}(P)) = \text{pref}(P)$ (见习题 3.10). ■

定理 3.3 与并发系统的分步设计有关. 如果初步设计 (即迁移系统) TS′ 细化为设计 TS, 使得

$$\text{Traces}(\text{TS}) \not\subseteq \text{Traces}(\text{TS}')$$

那么 TS′ 的 LT 性质不能带到 TS. 然而, 如果 TS 的有限迹是 TS′ 的有限迹 (这是一个比 TS 和 TS′ 的完整迹包含弱一些的要求), 即

$$\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) \subseteq \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$$

那么, 所有已为 TS′ 确认的安全性质同样也对 TS 成立. 对 TS 的其他要求, 即安全性质之外的 LT 性质, 要用不同的技术进行检验.

**推论** 有限迹等价与安全性质

令 TS 和 TS′ 是没有终止状态且有相同原子命题集合 AP 的迁移系统. 那么, 下列命题等价:

(a) $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) = \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$.

(b) 对于 AP 上的任何安全性质 $P_{\mathrm{safe}}$, $\mathrm{TS} \models P_{\mathrm{safe}} \iff \mathrm{TS}' \models P_{\mathrm{safe}}$.

下面对有限迹包含与迹包含之间的差异作出几点说明. 因为假设迁移系统没有终止状态, 所以在迹包含与有限迹包含之间仅有细微的区别. 对于没有终止状态的有限迁移系统 TS 与 TS′, 迹包含与有限迹包含是一致的. 这可由定理 3.4 推出.

**定理 3.4** 有限迹包含与迹包含的关系

令 TS 和 TS′ 是具有相同原子命题集合 AP 的迁移系统, 并且 TS 没有终止状态, TS′ 是有限的. 那么,

$$\mathrm{Traces}(\mathrm{TS}) \subseteq \mathrm{Traces}(\mathrm{TS}') \iff \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) \subseteq \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$$

**证明**: 由 $\mathrm{pref}(\cdot)$ 的单调性及 $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) = \mathrm{pref}(\mathrm{Traces}(\mathrm{TS}))$ 对任何迁移系统 TS 成立, 可从左推出右.

余下的就是证明可从右推出左. 假设 $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) \subseteq \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$. 因为 TS 没有终止状态, 它的所有迹都是无限的. 令 $A_0A_1A_2\cdots \in \mathrm{Traces}(\mathrm{TS})$. 为证明 $A_0A_1A_2\cdots \in \mathrm{Traces}(\mathrm{TS}')$, 需要说明 TS′ 存在能生成这个迹的路径, 例如 $s_0s_1s_2\cdots$, 使得 $\mathrm{trace}(s_0s_1s_2\cdots) = A_0A_1A_2$.

无限迹 $A_0A_1A_2\cdots$ 的任何有限前缀 $A_0A_1\cdots A_m$ 都在 $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS})$ 中, 且因为 $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}) \subseteq \mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$, 所以, 它也在 $\mathrm{Traces}_{\mathrm{fin}}(\mathrm{TS}')$ 中. 因而, 对任何自然数 $m$, 在 TS′ 中都存在有限路径 $\pi^m = s_0^m s_1^m \cdots s_m^m$ 使得

$$\mathrm{trace}(\pi^m) = L(s_0^m)L(s_1^m)\cdots L(s_m^m) = A_0A_1\cdots A_m$$

其中, $L$ 表示 TS′ 的标记函数. 因此, 对于所有 $0 \leqslant j \leqslant m$ 都有 $L(s_j^m) = A_j$.

尽管 $A_0A_1\cdots A_m$ 是 $A_0A_1\cdots A_{m+1}$ 的前缀, 但并不能保证 $\pi^m$ 是 $\pi^{m+1}$ 的前缀. 但是, 由于 TS′ 的有限性, $\pi^0\pi^1\pi^2\cdots$ 必存在子列 $\pi^{m_0}\pi^{m_1}\pi^{m_2}\cdots$ 使得 $\pi^{m_i}$ 和 $\pi^{m_{i+1}}$ 的前 $i$ 个状态相同. 因此, $\pi^{m_0}\pi^{m_1}\pi^{m_2}\cdots$ 可产生 TS 中的具有所需性质的无限路径 $\pi$.

用所谓的对角化技术可形式化地证明这一点. 过程如下. 构造 TS′ 中的状态 $s_0, s_1, s_2, \cdots$ 以及下标 (即自然数) 构成的无穷集合的无限序列 $I_0, I_1, I_2, \cdots$, 其中 $I_n \subseteq \{m \in \mathbb{N} \mid m \geqslant n\}$, 使它们满足以下性质:

(1) 若 $n \geqslant 1$, 则 $I_{n-1} \supseteq I_n$.