3

Net Models of Distributed Systems and Workflows

3.1 INFORMAL INTRODUCTION TO PETRI NETS

In 1962 Carl Adam Petri introduced a family of graphs, called Place-Transition (P/T) nets, to model dynamic systems [25]. P/T nets are bipartite graphs populated with tokens that flow through the graph. A bipartite graph is one with two classes of nodes; arcs always connect a node in one class with one or more nodes in the other class. In the case of P/T nets the two classes of nodes are places and transitions; arcs connect one place with one or more transitions or a transition with one or more places.

To model the dynamic behavior of systems, the places of a P/T net contain tokens; firing of transitions removes tokens from some places, called input places, and adds them to other places, called output places. The distribution of tokens in the places of a P/T net at a given time is called the marking of the net and reflects the state of the system being modeled.

P/T nets are very powerful abstractions and can express both concurrency and choice. P/T nets are used to model various activities in a distributed system; a transition may model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on. The input places of a transition model the preconditions of an event, the input data for the computational task, the presence of data in an input buffer, the preconditions of a logic statement. The output places of a transition model the postconditions associated with an event, the results of the computational task, the presence of data in an output buffer, or the conclusions of a logic statement.

P/T nets, or Petri nets (PNs), as they are commonly called, provide a very useful abstraction for system analysis and for system specification, as shown in Figure 3.1.
Fig. 3.1 Applications of Petri nets. (a) PNs are often used to model complex systems that are difficult or impossible to analyze by other means. In such cases one may construct a PN model of the system, $M$, then carry out a static and/or dynamic analysis of the net model and from this analysis infer the properties of the original system $S$. If $S$ is a software system one may attempt to translate it directly into a PN rather than build a model of the system. (b) A software system could be specified using the PN language. The net description of the system can be analyzed and, if the results of the analysis are satisfactory, then the system can be built from the PN description.

To analyze a system we first construct a PN model, then the properties of the net are analyzed using one of the methods discussed in this chapter, and, finally, the results of this analysis are mapped back to the original system, see Figure 3.1(a).

Another important application of the net theory is the specification of concurrent systems, using the Petri net language, see Figure 3.1(b). In this case a concurrent
system is described as a net, then the properties of the net are investigated using PN tools, and, when satisfied that the net has a set of desirable properties, the Petri net description is translated into an imperative computer language, that, in turn, is used to generate executable code.

P/T nets are routinely used to model distributed systems, concurrent programs, communication protocols, workflows, and other complex software, or hardware or systems. Once a system is modeled as a P/T net, we can perform static and dynamic analysis of the net. The structural analysis of the net is based on the topology of the graph and allows us to draw conclusions about the static properties of the system modeled by the net, while the analysis based on the markings of the net allow us to study its dynamic properties.

High-Level Petri nets, HLPNs, introduced independently by Jensen, [13], and Genrich and Lautenbach [8] in 1981, provide a more concise, or folded, graphical representation for complex systems consisting of similar or identical components. In case of HLPNs, tokens of different colors flow through the same subnet to model the dynamic behavior of identical subsystems. An HLPN can be unfolded into an ordinary P/T net.

To use PNs for performance analysis of systems we need to modify ordinary P/T nets, where transitions fire instantaneously, and to augment them with the concept of either deterministic or random time intervals. Murata [20], Ramamoorthy [27], Sifakis [28], and Zuberek [30] have made significant contributions in the area of timed Petri nets and their application to performance analysis. The so called Stochastic Petri nets (SPNs), introduced independently by Molloy [19] and Florin and Natkin [7] in 1982 associate a random interval of time with an exponential distribution to every transition in the net. Once a transition is ready to fire, a random interval elapses before the actual transport of tokens triggered by the firing of the transition takes places. An SPN is isomorphic with a finite Markov chain. Marsan and his co-workers [18] extended SPNs by introducing two types of transitions, timed and immediate.

Applications of stochastic Petri nets to performance analysis of complex systems is generally limited by the explosion of the state space of the models. In 1988 Lin and Marinescu [16] introduced Stochastic High-Level Petri nets (SHLPNs) and showed that SHLPNs allow easy identification of classes of equivalent markings even when the corresponding aggregation of states in the Markov domain is not obvious. This aggregation could reduce the size of the state space by one or more orders of magnitude depending on the system being modeled.

This chapter is organized as follows: we first define the basic concepts in net theory, then we discuss modeling with Petri nets and cover conflict, choice, synchronization, priorities, and exclusion. We discuss briefly state machines and marked graphs, outline marking independent, as well as marking dependent properties, and survey Petri net languages. We conclude the discussion of Petri net methodologies with an introduction to state equations and other methods for net analysis. We then review applications of Petri nets to performance analysis and modeling of logic programs. Finally, we discuss the application of Petri nets to workflow modeling and enactment, and discuss several concepts and models introduced by van der Aalst and Basten [1, 2] for the study of dynamic workflow inheritance.
3.2 BASIC DEFINITIONS AND NOTATIONS

In this section we provide a formal introduction to P/T nets and illustrate the concepts with the graphs in Figures 3.2 (a)-(j). Throughout this chapter the abbreviation iff stands for if and only if.

**Definition – Bag.** A bag $\mathcal{B}(\mathcal{A})$ is a multiset of symbols from an alphabet, $\mathcal{A}$; it is a function from $\mathcal{A}$ to the set of natural numbers.

**Example.** $[x^3, y^4, z^5, w^6 \mid P(x, y, z, w)]$ is a bag consisting of three elements $x$, four elements $y$, five elements $z$, and six elements $w$ such that the $P(x, y, z, w)$ holds. $P$ is a predicate on symbols from the alphabet. $x$ is an element of a bag $\mathcal{A}$ denoted as $x \in \mathcal{A}$ if $x \in \mathcal{A}$ and if $A(x) > 0$.

The sum and the difference of two bags $A$ and $B$ are defined as:

$$A + B = [x^n \mid x \in \mathcal{A} \land n = A(x) + B(x)]$$

$$A - B = [x^n \mid x \in \mathcal{A} \land n = \max(0, (A(x) + B(x)))]$$

The empty bag is denoted as $\emptyset$.

Bag $A$ is a subbag of $B$, $A \subseteq B$ iff $\forall x \in \mathcal{A} \quad A(x) \leq B(x)$.

**Definition – Labeled P/T net.** Let $U$ be a universe of identifiers and $L$ a set of labels. An L-labeled P/T Net is a tuple $N = (p, t, f, l)$ such that:

1. $p \subseteq U$ is a finite set of places.
2. $t \subseteq U$ is a finite set of transitions.
3. $f \subseteq (p \times t) \cup (t \times p)$ is a set of directed arcs, called flow relations.
4. $l : t \rightarrow L$ is a labeling or a weight function.

The weight function describes the number of tokens necessary to enable a transition. Labeled P/T nets as defined above describe a static structure. Places may contain tokens and the distribution of tokens over places defines the state of the P/T net and is called the marking of the net. We use the term marking and state interchangeably throughout this chapter. The dynamic behavior of a P/T net is described by the structure together with the markings of the net.

**Definition – Marked P/T net.** A marked, L-labeled P/T net is a pair $(N, s)$ where $N = (p, t, f, l)$ is an L-labeled P/T net and $s$ is a bag over $p$ denoting the markings of the net.

The set of all marked P/T nets is denoted by $\mathcal{N}$.

**Definition – Preset and Postset of Transitions and Places.** The **preset** of transition $t_i$ denoted as $\bullet t_i$ is the set of input places of $t_i$ and the **postset** denoted by $t_i \bullet$ is the set of the output places of $t_i$. The **preset** of place $p_j$ denoted as $\bullet p_j$ is the set of input transitions of $p_j$ and the **postset** denoted by $p_j \bullet$ is the set of the output transitions of $p_j$.

Figure 3.2(a) shows a P/T net with three places, $p_1$, $p_2$, and $p_3$, and one transition, $t_1$. The weights of the arcs from $p_1$ and $p_2$ to $t_1$ are two and one, respectively; the weight of the arc from $t_1$ to $p_3$ is three.
Fig. 3.2 Place Transition Nets. (a) An unmarked P/T net with one transition $t_1$ with two input places, $p_1$ and $p_2$ and one output place, $p_3$. (b)-(c) The net in (a) as a Marked P/T net before and after firing of transition $t_1$. (d) Modeling choice with P/T nets. Only one of transitions $t_1$, or $t_2$ may fire. (e) Symmetric confusion; transitions $t_1$ and $t_3$ are concurrent and, at the same time, they are in conflict with $t_2$. If $t_2$ fires, then $t_1$ and $t_3$ are disabled. (f) Asymmetric confusion; transition $t_3$ is concurrent with $t_3$ and it is in conflict with $t_2$ if $t_3$ fires before $t_1$. (g) A state machine; there is the choice of firing $t_1$, or $t_2$; only one transition fires at any given time, concurrency is not possible. (h) A marked graph allows us to model concurrency but not choice; transitions $t_2$ and $t_3$ are concurrent, there is no causal relationship between them. (i) An extended P/T net used to model priorities. The arc from $p_2$ to $t_1$ is an inhibitor arc. The task modeled by transition $t_1$ is activated only after the task modeled by transition $t_2$ is activated. (j) Modeling exclusion: the net models $n$ concurrent processes in a shared memory environment. At any given time only one process may write but all $n$ may read. Transitions $t_1$ and $t_2$ model writing and respectively reading.
The preset of transition \( t_1 \) in Figure 3.2(a, b, c) consists of two places, \( \bullet t_1 = \{p_1, p_2\} \) and its postset consist of only one place, \( t_1 \bullet = \{p_3\} \). The preset of place \( p_4 \) in Figure 3.2(g) consists of transitions \( t_3 \) and \( t_4 \), \( \bullet p_4 = \{t_3, t_4\} \) and the postset of \( p_1 \) is \( p_1 \bullet = \{t_1, t_2\} \).

**Definition – Source and Sink Transitions; Self-loops.** A transition without any input place is called a source transition and one without any output place is called a sink transition. A pair consisting of a place \( p_i \) and a transition \( t_j \) is called a self-loop if \( p_i \) is both the input and output of \( t_j \).

Transition \( t_1 \) in Figure 3.2(h) is a source transition, while \( t_4 \) is a sink transition.

**Definition – Pure Net.** A net is pure if there are no self loops.

**Definition – Ordinary Net.** A net is ordinary if the weights of all arcs are 1.

The nets in Figures 3.2(c, d, e, f, g, h) are ordinary nets, the weights of all arcs are 1.

**Definition – Start and Final Places.** A place without any input transitions is called a start place and one without any output transition is called a final place. \( p_s \) is a start place iff \( \bullet p_s = \emptyset \) and \( p_f \) is a final place iff \( p_f \bullet = \emptyset \).

**Definition – Short-Circuit Net.** Given a P/T net \( N = (p, t, f, l) \) with one start place, \( p_s \) and one final place \( p_f \) the network \( \tilde{N} \) obtained by connecting \( p_f \) to \( p_s \) with an additional transition \( \tilde{t}_k \) labeled \( \tau \) is called the short-circuit net associated with \( N \).

\[
\tilde{N} = \langle p, t \cup \{\tilde{t}_k\}, f \cup \{(p_f, \tilde{t}_k), (\tilde{t}_k, p_s)\}, l \cup \{\{\tilde{t}_k, \tau\}\} \rangle
\]

**Definition – Enabled Transition.** A transition \( t_i \in t \) of the ordinary net \( (N, s) \) is enabled iff each of its input places contain a token, \( (N, s)[t_i] \Leftrightarrow \bullet t_i \in s \). Here \( s \) is the initial marking of the net. The fact that \( t_i \) is enabled is denoted as, \( (N, s)[t_i] \).

The marking of a P/T net changes as a result of transition firing. A transition must be enabled in order to fire. The following firing rule governs the firing of a transition.

**Definition – Firing Rule.** Firing of the transition \( t_i \) of the ordinary net \( (N, s) \) means that a token is removed from each of its input places and one token is added to each of its output places. Firing of transition \( t_i \) changes a marked net \( (N, s) \) into another marked net \( (N, s - \bullet t_i + t_i \bullet) \).

**Definition – Finite and Infinite Capacity Nets.** The capacity of a place is the maximum number of tokens the place may hold. A net with places that can accommodate an infinite number of tokens is called an infinite capacity net. In a finite capacity net we denote by \( K(p) \) the capacity of place \( p \).

There are two types of firing rules for finite capacity nets, strict and weak, depending on the enforcement of the capacity constraint rule.

**Definition – Strict and Weak Firing Rules.** The strict firing rule allows an enabled transition \( t_i \) to fire iff after firing the transition \( t_i \), the number of tokens in each place of its postset \( p_j \in \bullet t_i \), does not exceed the capacity of that place \( K(p_j) \). The weak firing rule does not require the firing to obey capacity constraints.

Figure 3.2(b) shows the same net as the one in Figure 3.2(a) with three token in place \( p_1 \) and one in \( p_2 \). Transition \( t_1 \) is enabled in the marked net in Figure 3.2(b);
Figure 3.2(c) shows the same net after firing of transition \( t_1 \). The net in Figure 3.2(b) models synchronization, transition \( t_1 \) can only fire if the condition associated with the presence of two tokens in \( p_1 \) and one token in \( p_2 \) are satisfied.

In addition to regular arcs, a P/T net may have inhibitor arcs that prevent transitions to be enabled.

**Definition – Extended P/T Nets.** P/T nets with inhibitor arcs are called **extended P/T nets**.

**Definition – Modified Transition Enabling Rule for Extended P/T Nets.** A transition is not enabled if one of the places in its preset is connected with the transition with an inhibitor arc and if the place holds a token.

For example, transition \( t_1 \) in the net in Figure 3.2(i) is not enabled while place \( p_2 \) holds a token.

### 3.3 MODELING WITH PLACE/TRANSITION NETS

#### 3.3.1 Conflict/Choice, Synchronization, Priorities, and Exclusion

P/T nets can be used to model concurrent activities. For example, the net in Figure 3.2(d) models conflict or choice, only one of the transitions \( t_1 \) and \( t_2 \) may fire but not both. Transition \( t_4 \) and its input places \( p_3 \) and \( p_4 \) in Figure 3.2(h) model synchronization; \( t_4 \) can only fire if the conditions associated with \( p_3 \) and \( p_4 \) are satisfied.

Two transitions are said to be concurrent if they are causally independent, as discussed in Chapter 2. Concurrent transitions may fire before, after, or in parallel with each other, as is the case of transitions \( t_2 \) and \( t_3 \) in Figure 3.2(h). The net in this figure models concurrent execution of two tasks, each one associated with one of the concurrent transitions and transition \( t_4 \) models synchronization of the two tasks.

When choice and concurrency are mixed, we end up with a situation called confusion. **Symmetric confusion** means that two or more transitions are concurrent and, at the same time, they are in conflict with another one. For example, in Figure 3.2 (e), transitions \( t_1 \) and \( t_3 \) are concurrent and in the same time they are in conflict with \( t_2 \). If \( t_2 \) fires either one or both of them will be disabled. **Asymmetric confusion** occurs when a transition \( t_1 \) is concurrent with another transition \( t_3 \) and will be in conflict with \( t_2 \) if \( t_2 \) fires before \( t_1 \) as shown in Figure 3.2 (f).

Place Transition Nets, can be used to model priorities. The net in Figure 3.2(i) models a system with two tasks \( task_1 \) and \( task_2 \); \( task_2 \) has higher priority than \( task_1 \). Indeed, if both tasks are ready to run, both places \( p_1 \) and \( p_2 \) hold tokens. When both tasks are ready, transition \( t_2 \) will fire first, modeling the activation of \( task_2 \). Only after \( t_2 \) is activated transition \( t_1 \), modeling of activation of \( task_1 \), will fire.

P/T nets are able to model exclusion. For example, the net in Figure 3.2(j), models a group of \( n \) concurrent tasks executing in a shared-memory environment. All tasks can read at the same time, but only one may write. Place \( p_3 \) models the tasks allowed to write, \( p_4 \) the ones allowed to read, \( p_2 \) the ones ready to access the shared memory and \( p_1 \) the running tasks. Transition \( t_2 \) models the initialization/selection of tasks
allowed to write and $t_1$ of those allowed to read, whereas $t_3$ models the completion of a write and $t_4$ the completion of a read.

Indeed $p_3$ may have at most one token while $p_4$ may have at most $n$. If all $n$ tasks are ready to access the shared memory all $n$ tokens in $p_2$ are consumed when transition $t_1$ fires. However, place $p_4$ may contain $n$ tokens obtained by successive firings of transition $t_2$.

### 3.3.2 State Machines and Marked Graphs

Structural properties allow us to partition the set of nets into several subclasses: (a) state machines, (b) marked graphs, (c) free-choice nets, (d) extended free-choice nets, and (e) asymmetric choice nets. This partitioning is based on the number of input and output flow relations from/to a transition or a place and by the manner in which transitions share input places as indicated in Figure 3.3.

**Fig. 3.3** Subclasses of Place Transition nets. State Machines do not model concurrency and synchronization; Marked Graphs do not model choice and conflict; Free Choice nets do not model confusion; Asymmetric-Choice Nets allow asymmetric confusion but not symmetric one.

Finite state machines can be modeled by a subclass of L-labeled P/T nets called *state machines* with the property that each transition has exactly one incoming and one outgoing arc or flow relation. This topological constraint limits the expressiveness of a state machine, no concurrency is possible. In the followings we consider marked state machines $(N, s)$ where marking $s_0 \in s$ corresponds to the initial state.

For example, in the net from Figure 3.2(g) transitions $t_1$, $t_2$, $t_3$, and $t_4$ have only one input and output arc, the cardinality of their presets and postsets is one. No concurrency is possible; once a choice was made by firing either $t_1$ or $t_2$ the evolution of the system is entirely determined.
Recall that a marking/state reflects the disposition of tokens in the places of the net. For the net in Figure 3.2 (g) with four places, the marking is a 4-tuple \((p_1, p_2, p_3, p_4)\). The markings of this net are \((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)\).

**Definition – State Machine.** Given a marked P/T net, \((N, s_0)\) with \(N = (p, t, f, l)\) we say that \(N\) is a state machine if \(\forall t_i \in t \ (|t_i| = 1 \land |t_i \cdot | = 1)\).

State machines allow us to model choice or decision, because each place may have multiple output transitions, but does not allow modeling of synchronization or concurrent activities. Concurrent activities require that several transitions be enabled concurrently. The subclass of L-labeled P/T nets called marked graphs allow us to model concurrency.

**Definition – Marked Graph.** Given a marked P/T net, \((N, s_0)\) with \(N = (p, t, f, l)\) we say that \(N\) is a marked graph if \(\forall p_i \in p \ | p_i | = 1 \land | p_i \cdot | = 1\).

In a marked graph each place has only one incoming and one outgoing flow relation; thus, marked graphs do not allow modeling of choice.

### 3.3.3 Marking Independent Properties of P/T Nets

Dependence on the initial marking partitions the set of properties of a net into two groups: structural properties, those independent of the initial marking, and behavioral or marking-dependent properties. Strong connectedness and free-choice are examples of structural properties, whereas liveness, reachability, boundeness, persistency, coverability, fairness, and synchronic distance are behavioral properties.

**Definition – Strongly Connected P/T Net.** A P/T net \(N = (p, t, f, l)\) is strongly connected if \(\forall x, y \in p \cup t \ x f^* y\).

Informally, strong connectedness means that there is a directed path from one element \(x \in p \cup t\) to any other element \(y \in p \cup t\). Strong connectedness is a static property of a net.

**Definition – Free Choice, Extended Free Choice, and Asymmetric Choice P/T Nets.** Given a marked P/T net, \((N, s_0)\) with \(N = (p, t, f, l)\) we say that \(N\) is a free-choice net if

\[
(\bullet t_i) \cap (\bullet t_j) = \emptyset \Rightarrow |\bullet t_i| = |\bullet t_j| .
\]

when \(\forall t_i, t_j \in t\).

\(N\) is an extended free-choice net if \(\forall t_i, t_j \in t\) then \((\bullet t_i) \cap (\bullet t_j) = \emptyset \Rightarrow \bullet t_i = \bullet t_j\).

\(N\) is an asymmetric choice net if \((\bullet t_i) \cap (\bullet t_j) \neq \emptyset \Rightarrow (\bullet t_i \subseteq \bullet t_j) \text{ or } (\bullet t_i \supseteq \bullet t_j)\).

In an extended free-choice net if two transition share an input place they must share all places in their presets. In an asymmetric choice net two transitions may share only a subset of their input places.
3.3.4 Marking Dependent Properties of P/T Nets

Definition – Firing Sequence. Given a marked P/T net, \( (N, s_0) \) with \( N = (p, t, f, l) \), a nonempty sequence of transitions \( \sigma \in t^* \) is called a firing sequence iff there exist markings \( s_1, s_2, \ldots, s_n \in B(p) \) and transitions \( t_1, t_2, \ldots, t_n \in t \) such that \( \sigma = t_1 t_2 \ldots t_n \) and for \( i \in (0, n) \), \( (N, s_i) t_{i+1} > \) and \( s_{i+1} = s_i - \bullet t_i + t_i \bullet \). All firing sequences that can be initiated from marking \( s_0 \) are denoted as \( \sigma(s_0) \).

Firing of a transition changes the state or marking of a P/T net, the disposition of tokens into places is modified.

Reachability is the problem of finding if marking \( s_n \) is reachable from the initial marking \( s_0, s_n \in \sigma(s_0) \). Reachability is a fundamental concern for dynamic systems. The reachability problem is decidable, but reachability algorithms require exponential time and space.

Definition – Liveness. A marked P/T net \( (N, s_0) \) is said to be live if it is possible to fire any transition starting from the initial marking, \( s_0 \). We recognize several levels of liveness of a P/T net. A transition \( t \) is

- L0-live, dead if it cannot be fired in any firing sequence in \( \sigma(s_0) \),
- L1-live, potentially firable if it can be fired at least once in \( \sigma(s_0) \),
- L2-live if given an integer \( k \) it can be fired at least \( k \) times in some firing sequence in \( \sigma(s_0) \),
- L3-live if it appears infinitely often in some firing sequence in \( \sigma(s_0) \).

The net in Figure 3.4(a) is live; in Figure 3.4(b) transition \( t_3 \) is L0-live, transition \( t_2 \) is L1-live, and transition \( t_1 \) is L3-live.

Corollary. The absence of deadlock in a system is guaranteed by the liveness of its net model.

Definition – Syphons and Traps. Given a P/T net \( N \), a nonempty subset of places \( Q \) is called a siphon/deadlock if \( \bullet Q \subseteq (Q \bullet) \) and it is called a trap if \( (Q \bullet) \subseteq (\bullet Q) \).

In Figure 3.4(c) the subnet \( Q \) is a siphon; in Figure 3.4(d), the subnet \( R \) is a trap.

Definition – Boundedness. A marked P/T net \( (N, s_0) \) is said to be \( k \)-bounded if the number of tokens in each place does not exceed the finite number \( k \) for any reachable marking from \( s_0 \).

Definition – Safety. A marked P/T net \( (N, s_0) \) is said to be safe if it is 1-bounded, for any reachable marking \( s' \in [N, s_0] > \) and any place \( p' \in p \), \( s'(p) \leq 1 \).

Definition – Reversibility. A marked P/T net \( (N, s_0) \) is reversible if for any marking \( s_n \in \sigma(s_0) \), the original marking \( s_0 \) is reachable from \( s_n \). More generally a marking \( s' \) is a home state if for every marking \( s \in \sigma(s_0) \), \( s' \) is reachable from \( s \).

Reversibility of physical systems is desirable; we often require a system to return to some special state. For example, an interrupt vector defines a set of distinguished
states of a computer system we want to return to, when an interrupt occurs. Reversibility is a property of a net necessary to model reversible physical systems; it guarantees that a net can go back to its initial marking.

**Definition – Persistence.** A marked P/T net $(N, s_0)$ is persistent if for any pair of two transitions $(t_i, t_j)$, firing of one does not disable the other.

Persistency is a property of conflict-free nets, e.g., all marked graphs are persistent because they do not allow conflicts and choice. Moreover, a safe persistent net can be transformed into a marked graph by duplicating some places and transitions.

**Definition – Synchronic Distance.** Given a marked P/T net $(N, s_0)$, the synchronic distance between two transitions $t_i$ and $t_j$ is $d_{i,j} = \max[\sigma(t_i) - \sigma(t_j)]$, with $\sigma$ a firing sequence and $\sigma(t_i)$ the number of times transition $t_i$ fires in $\sigma$.

The synchronic distance gives a measure of dependency between two transitions. For example, in Figure 3.2(j) $d(t_2, t_3) = 1$, and $d(t_1, t_2) = \infty$. Indeed, once a task is allowed to write, it will always complete the writing, while reading and writing are independent.

**Definition – Fairness.** Given a marked P/T net $(N, s_0)$, two transitions $t_i$ and $t_j$, are in a bounded-fair, B-fair, relation if the maximum number one of them is allowed to
fire while the other one not firing is bounded. If all pairs of transitions are in a B-fair relation then the P/T net is a B-fair net. A firing sequence $\sigma$ is unconditionally fair if every transition in $\sigma$ appears infinitely often.

**Definition – Coverability.** A marking $s$ of a marked P/T net $(N, s_0)$, is coverable if there exist another marking $s'$ such that for every place $p$, $s'(p) \geq s(p)$, with $s(p)$ denoting the number of tokens in $p$ under marking $s$.

Coverability is related to L1-liveness.

### 3.3.5 Petri Net Languages

Consider a finite alphabet $A = a, b, c, d, ..., w$ with $w$ the null symbol. Given a marked P/T net, $N = (p, t, f, l)$ with a start place $p_s$ and a final place $p_f$, $p_s, p_f \in p$, let us label every transition $t_i \in t$ with one symbol from $A$. Multiple transitions may have the same label.

**Definition – Petri net Language.** The set of strings generated by every possible firing sequence of the net $N$ with initial marking $M_0 = (1, 0, 0, ..., 0)$, when only the start place holds a token, and terminates when all transitions are disabled, defines a language $L(M_0)$.

**Example.** The set of strings generated by all possible firing sequences of the net in Figure 3.5 with the initial marking $M_0$, defines the Petri net language

$$L(M_0) = \{ (ef)^m(a)^n(b)^p(c)^q \mid m \geq 0, 0 \leq n < 2, p \geq 0, q \geq 0 \}.$$  

Every state machine can be modeled by a Petri net, thus every regular language is a Petri net language. Moreover it has been proved that all Petri net languages are context sensitive [24].

### 3.4 STATE EQUATIONS

**Definition – Incidence Matrix.** Given a P/T net with $n$ transitions and $m$ places, the incidence matrix $F = [f_{i,j}]$ is an integer matrix with $f_{i,j} = w(i, j) - w(j, i)$. Here $w(i, j)$ is the weight of the flow relation (arc) from transition $t_i$ to its output place $p_j$, and $w(j, i)$ is the weight of the arc from the input place $p_j$ to transition $t_i$. In this expression $w(i, j)$ represents the number of tokens added to the output place $p_j$ and $w(j, i)$ the ones removed from the input place $p_j$ when transition $t_i$ fires. $F^T$ is the transpose of the incidence matrix.

A marking $s_k$ can be written as a $m \times 1$ column vector and its $j$-th entry denotes the number of tokens in place $j$ after some transition firing.

The necessary and sufficient condition for transition $t_i$ to be enabled at a marking $s$ is that $w(j, i) \leq s(j) \ \forall s_j \in \bullet t_i$, the weight of the arc from every input place of the transition, be smaller or equal to the number of tokens in the corresponding input place.

Consider a firing sequence $\sigma$ and let the $k$-th transition in this sequence be $t_i$. In other words, the $k$-th firing in $\sigma$ will be that of transition $t_i$. Call $s_{k-1}$ and $s_k$ the
states/markings before and after the k-th firing and \( u_k \) the firing vector, an integer \( n \times 1 \) row vector with a 1 for the k-th component and 0’s elsewhere.

The dynamic behavior of the P/T net \( N \) is characterized by the state equation relating consecutive states/markings in a firing sequence:

\[
s_k = s_{k-1} + F^T u_k.\]

Reachability can be expressed using the incidence matrix. Indeed, consider a firing sequence of length \( d \), \( \sigma = u_1 u_2 \ldots u_d \) from the initial marking \( s_0 \) to the current marking \( s_q \). Then:
\[ s_q = s_0 + F^T \sum_{k=1}^{d} u_k \]

or

\[ F^T x = \Delta s \]

with \( x = \sum_{k=1}^{d} u_k \) called a firing count vector, an \( n \times 1 \) column vector of non-negative integers whose \( i \)-th component indicates how many times transition \( t_i \) must fire to transform the marking \( s_0 \) into \( s_q \) with \( \Delta s = s_q - s_0 \).

**Definition – T and S invariant.** An integer solution of the equation \( F^T x = 0 \) is called a T invariant. An integer solution of the equation \( F y = 0 \) is called an S invariant.

Intuitively, place invariants of a net with all flow relations (arcs) of weight 1 are sets of places that do not change their token count during firing of transitions; transition invariants indicate how often, starting from some marking, each transition has to fire to reproduce that marking.

### 3.5 PROPERTIES OF PLACE/TRANSITION NETS

The liveness, safeness and boundedness are orthogonal properties of a P/T net, a net may posses one of them independently of the others. For example, the net in Figure 3.4(a) is live, bounded, safe, and reversible. Transitions \( t_1 \) and \( t_2 \) are L3-live, the number of tokens in \( p_1 \) and \( p_2 \) is limited to one and marking \((1, 0)\) can be reached from \((0, 1)\). The net in Figure 3.4(b) is not live, it is bounded, safe, and not reversible.

A number of transformations, e.g., fusion of Series/Parallel Places/Transitions preserve the liveness, safeness, and boundedness of a net as seen in Figure 3.6

We now present several well-known results in net theory. The proof of the following theorems is beyond the scope of this book and can be found elsewhere.

**Theorem – Live and Safe Marked P/T Nets.** If a marked P/T net \((N, s_0)\) is live and safe then \( N \) is strongly connected. The reciprocal is not true, there are strongly connected nets that are not live and safe. The net in Figure 3.4(d) is an example of a strongly connected network that is not live.

State machines enjoy special properties revealed by the following theorem.

**Theorem – Live and Safe State Machines.** A state machine \((N, s_0)\) is live and safe iff \( N \) is strongly connected and if marking \( s_0 \) has exactly one token.

A marked graph can be represented by a directed graph with nodes corresponding to the transitions and arcs corresponding to places of the marked graph. The presence of tokens in a place is shown as a token on the corresponding arc. Firing of a transition corresponds to removing a token from each of the input arcs of a node of the directed graph and placing them on the output arcs of that node. A directed circuit in the
directed graph consists of a path starting and terminating in the same node. In this representation a marked graph consists of a number of connected directed circuits.

**Theorem – Live Marked Graph.** A marked graph \((N, s_0)\) is live iff marking \(s_0\) places at least one token on each directed circuit in \(N\).

Indeed, the number of tokens in a directed circuit is invariant under any firing. If a directed circuit contains no tokens at the initial marking, then no tokens can be injected into it at a later point in time, thus, no transitions in that directed circuit can be enabled.

**Theorem – Safe Marked Graph.** A live marked graph \((N, s_0)\) is safe iff every place belongs to a directed circuit and the total count of tokens in that directed circuit in the initial marking \(s_0\) is equal to one.

**Theorem – Live Free-Choice Net.** A free-choice net \((N, s_0)\) is live iff every syphon in \(N\) contains a marked trap.
We now present two theorems that show that a live and safe free-choice net can be seen as the interconnection of live and safe state-machines, or, equivalently, the interconnection of live and safe marked graphs.

A state machine component of a net \( N \) is a subnet constructed from places and transitions in \( N \) such that each transition has at most one incoming and one outgoing arc and the subnet includes all the input and output places of these transitions and the connecting arcs. A marked graph component of a net \( N \) is a subnet constructed from places and transitions in \( N \) such that each place has at most one incoming and one outgoing arc and the subnet includes all the input and output places of these transitions and the connecting arcs.

**Theorem – Safe Free-Choice Nets and State Machines.** A live free-choice net \( N \) is safe iff \( N \) is covered by strongly connected state machine components and each component state machine has exactly one token in the initial marking.

**Theorem – Safe Free-Choice Nets and Marked Graphs.** A live and safe free-choice net \( N \) is covered by strongly connected marked graph components.

### 3.6 COVERABILITY ANALYSIS

Given a net \( (N, s_0) \) we can identify all transitions enabled in the initial marking, \( s_0 \) and fire them individually to reach new markings; then in each of the markings reached in the previous stage we can fire, one by one, the transitions enabled and continue ad infinum.

In this manner we can construct a tree of all markings reachable from the initial one; if the net is unbounded, this tree will grow continually. To prevent this undesirable effect we use the concept of a coverable marking introduced earlier.

Recall that a marking \( s \) of a marked P/T net \( (N, s_0) \) with \( |P| \) places is a vector, \( (s(p_1), s(p_2), \ldots s(p_i), \ldots s(p_{|P|})) \); component \( s(p_i) \) gives the number of tokens in place \( p_i \). Marking \( s \) is said to be coverable if there exists another marking \( s' \) such that for every place, the number of tokens in \( p_i \) under marking \( s \) is larger, or at least equal to the one under marking \( s', s'(p_i) \geq s(p_i), \; 1 \leq i \leq |P| \).

For example, in Figure 3.7(a) the initial marking is \( (1, 0, 0) \) with one token in place \( p_1 \) and zero tokens in \( p_2 \) and \( p_3 \). In this marking two transitions are enabled, \( t_1 \) and \( t_5 \). When \( t_1 \) fires we reach the marking \( (1, 0, 1) \) and when \( t_5 \) fires we stay in the same marking, \( (1, 0, 0) \). Marking \( (1, 0, 1) \) covers \( (1, 0, 0) \).

We discuss now the formal procedure to construct the finite tree representation of the markings. First, we introduce a symbol \( \omega \) with the following properties: given any integer \( n, \omega > n, \omega \neq n = \omega, \text{and } \omega \geq \omega \). Each node will be labeled with the corresponding marking and tagged with the symbol \( \text{new} \) when it is visited for the first time, \( \text{old} \) when it is revisited, or \( \text{dead end} \) if no transitions are enabled in the marking the node is labeled with. The algorithm to construct the coverability tree is:

- Label the root of the tree with the initial marking \( s_0 \) and tag it as \( \text{new} \).
- While nodes tagged as \( \text{new} \) exist do:
Fig. 3.7  (a) A Petri net. (b) The coverability tree of the net in (a). (c) The coverability graph of the net in (a).

- Select a node tagged as new labeled with marking $s$.

- If there is another node in the tree on the path from the root to the current node with the same label $s$, then tag the current node as old and go to the first step.
– If no transitions are enabled in marking $s$ then tag the node as dead end and go to the first step.

– For all transitions $t_j$ enabled in marking $s$:
  * fire $t_j$ and determine the new marking $s'$,
  * add a new node to the graph,
  * connect the new node to the parent node by an arc labeled $t_j$,
  * tag the new node as new, and
  * determine the label of this node as follows: if on the path from the root to the parent node exists a node labeled $s''$ such that $s''(p_i) > s''(p_i)$ then identify all places $p_i$ such that $s''(p_i)$ and replace $s''(p_i) = \omega$; else label the new node $s'$.

Figure 3.7(b) illustrates the construction of the coverability graph for the net in Figure 3.7(a). As pointed out earlier, marking $(1, 0, 1)$ covers $(1, 0, 0)$ thus the node in the graph resulting after firing transition $t_1$ in marking $(1, 0, 0)$ is labeled $(1, 0, \omega)$.

From the coverability tree $T$ we can immediately construct the coverability graph $G$ of the net, as shown in Figure 3.7(c). $G$ is the state transition graph of the system modeled by the Petri net.

In our example, the net can only be in one of three states, $(1, 0, 0), (1, 0, \omega), (0, 1, \omega)$; transition $t_3$ leads to a self-loop in marking $(1, 0, 0)$, transition $t_1$ to a self-loop in state $(1, 0, \omega)$, and transition $t_3$ to a self-loop in marking $(0, 1, \omega)$. Transition $t_2$ takes the net from the marking $(1, 0, \omega)$ to $(0, 1, \omega)$ and transition $t_4$ does the opposite. Firing transition $t_2$ in marking $(1, 0, \omega)$ leads to the marking $(0, 1, \omega)$, and so on.

The coverability tree, $T$, is very useful to study the properties of a net, $(N, s_0)$. We can also identify all the markings $s'$ reachable from a given marking $s$. If a transition does not appear in the coverability tree, it means that it will never be enabled; it is a dead transition. If the symbol $\omega$ does not appear in any of the node labels of $T$, then the net is bounded. If the labels of all nodes contain only 0’s and 1’s then the net is safe.

### 3.7 APPLICATIONS OF STOCHASTIC PETRI NETS TO PERFORMANCE ANALYSIS

In this section we introduce SPNs, and SHLPNs. Then we present an application.

#### 3.7.1 Stochastic Petri Nets

SPNs are obtained by associating with each transition in a Petri net a possibly marking dependent transition rate for the exponentially distributed firing time.

**Definition.** An SPN is a quintuple:

$$SPN = (p, t, f, s, \lambda)$$
Fig. 3.8  (a) The SPN model of the philosopher system. (b) The state transition diagram of the system.

1. \( p \) is the set of places.
2. \( t \) is the set of transitions.
3. \( p \cap t = \emptyset, p \cup t \neq \emptyset \). 

4. \( f \) is the set of input and output arcs; \( f \subseteq (p \times t) \cup (t \times p) \).

5. \( s \) is the initial marking.

6. \( \lambda \) is the set of transition rates.

The SPNs are isomorphic to continuous time Markov chains due to the memoryless property of the exponential distribution of firing times. The SPN markings correspond to the states of the corresponding Markov chain so that the SPN model allows the calculation of the steady-state and transient system behavior.

In SPN analysis, as in Markov analysis, ergodic (irreducible) systems are of special interest. For ergodic SPN systems, the steady-state probability of the system being in any state always exists and is independent of the initial state. If the firing rates do not depend on time, a stationary (homogeneous) Markov chain is obtained. In particular, \( k \)-bounded SPNs are isomorphic to finite Markov chains. We consider only ergodic, stationary, and \( k \)-bounded SPNs (or SHLPNs) and Markov chains.

**Example of SPN modeling.** Consider a group of five philosophers who spend some time thinking between copious meals. There are only five forks on a circular table and there is a fork between two philosophers. Each philosopher needs the two adjacent forks. When they become free, the philosopher hesitates for a random time, exponentially distributed with average \( 1/\lambda_1 \), and then moves from the thinking phase to the eating phase where he spends an exponentially distributed time with average \( 1/\lambda_2 \). This system is described by the SPN in Figure 3.8(a). The model has 15 places and 10 transitions, all indexed on variable \( i; i \in [1, 5] \) in the following description.

- \( T_i \) is the “thinking” place. If \( T_i \) holds a token, the \( i \)th philosopher is pretending to think while waiting for forks.
- \( E_i \) is the “eating” place. If \( E_i \) holds a token, the \( i \)th philosopher is eating.
- \( F_i \) is the “free fork” place. If \( F_i \) holds a token, the \( i \)th fork is free.
- \( G_i \) is the “getting forks” transition. This transition is enabled when the hungry philosopher can get the two adjacent forks. The transition firing time is associated with \( 1/\lambda_1 \) and it is related to the time the philosopher hesitates before taking the two forks and starting to eat.
- \( R_i \) is the “releasing forks” transition. A philosopher releases the forks and returns to the thinking stage after the eating time exponentially distributed with average \( 1/\lambda_2 \).

The SPN model of the philosopher system has a state space size of 11 and its states (markings) are presented in Table 3.1. The state transition diagram of the
Table 3.1  The markings of the philosopher system in the SPN modeling example .

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$E_1$</th>
<th>$E_2$</th>
<th>$E_3$</th>
<th>$E_4$</th>
<th>$E_5$</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>$F_4$</th>
<th>$F_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_4$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_5$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_6$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_7$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_8$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_9$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The corresponding Markov chain is shown in Figure 3.8(b). The steady-state probabilities that the system is in state $i$, $p_i$, can be obtained:

$$p_i = \begin{cases} 
\frac{\lambda_2}{5 \lambda_1 (\lambda_1 + \lambda_2)} + \frac{\lambda_2^2}{\lambda_1^2} & i = 0 \\
\frac{5 \lambda_1 (\lambda_1 + \lambda_2) + \lambda_2^2}{\lambda_1^2} & i = 1, 2, 3, 4, 5 \\
\frac{5 \lambda_1 (\lambda_1 + \lambda_2) + \lambda_2^2}{\lambda_1^2} & i = 6, 7, 8, 9, 10.
\end{cases}$$

3.7.2 Informal Introduction to SHLPNs

Our objective is to model the same system using a representation that leads to a model with a smaller number of states. The following notation is used throughout this section: $a \oplus b \ (\text{mod} \ p)$ stands for addition modulo $p$. $|\{\}|$ denotes the cardinality of a set. The relations between the element and the set, $c$ and $g$, are often used in the predicates.

The SHLPNs will be introduced by means of an example that illustrates the fact that an SHLPN model is a scaled down version of an SPN model, it has a smaller number of places, transitions, and states than the original SPN model. Figure 3.9(a) presents the SHLPN model of the same philosopher system described in Figure 3.8 using an SPN. In the SHLPN model, each place and each transition stands for a set of places or transitions in the SPN model. The number of places is reduced from 15 to 3, the place $T$ stands for the set $\{T_i\}$, $E$ stands for $\{E_i\}$, and $F$ stands for $\{F_i\}$, for $i \in [1, 5]$. The number of transitions is reduced from ten to two; the transition $G$ stands for the set $\{G_i\}$ and $R$ stands for the set $\{R_i\}$ with $i \in [1, 5]$.

The three places contain two types of tokens, the first type is associated with the philosophers and the second is associated with forks, see Figure 3.9(a). The arcs are labeled by the token variables. A token has a number of attributes, the first attribute is the type and the second attribute the identity, $id$. 
The tokens residing in the place $E$, the eating place, have four attributes; the last two attributes are the *ids* of the forks currently used by the philosopher. The transition
$G$ is associated with the predicate which specifies the correct relation between a philosopher and the two forks used by him. The predicate inscribed on transition $G$, see Figure 3.9(a), as $i = j$ is a concise form of expressing that the second attribute of a $(p, i)$ token should be equal to the second attribute of the two tokens representing the forks. This means that a philosopher can eat only when two adjacent forks are free; for example, the forks $(f, 3)$ and $(f, 4)$ must be free in order to allow the philosopher $(p, 3)$ to move to the eating place.

A predicate expresses an imperative condition that must be met in order for a transition to fire. A predicate should not be used to express the results associated with the firing of a transition. There is no predicate associated with transition $R$ in Figure 3.9(a), although there is a well-defined relationship between the attributes of the tokens released when $R$ fires.

In an SHLPN model, the transition rate associated with every transition is related to the markings that enable that particular transition. To simplify the design of the model, only the transition rate of the individual markings is shown in the graph, instead of the transition rate of the corresponding compound markings. For example, in Figure 3.9(a), the transition rates are written as $\lambda_1$ for the transition $G$ and $\lambda_2$ for the transition $R$.

As shown in Figure 3.9(b) the system has three states, $S_i$ with $i \in \{0, 1, 2\}$ representing the number of philosophers in the eating place. The actual transition rates corresponding to the case when the transition $G$ fires are $5 \times \lambda_1$ and $2 \times \lambda_1$ depending on the state of the system when the transition $G$ fires. If the system is in state $S_0$, then there are five different philosophers who can go to the eating place; hence, the actual transition rate is $5 \times \lambda_1$.

The problem of determining the compound markings and the transition rates among them is discussed in the following. The markings (states) of the philosopher system based on HLPN are given in Table 3.2. The initial population of different places is five tokens in $T$, five tokens in $F$, and no token in $E$. When one or more philosophers are eating, $E$ contains one or more tokens.

In many systems, a number of different processes have a similar structure and behavior. To simplify the system model, it is desirable to treat similar processes in a uniform and succinct way. In the HLPN models, a token type may be associated with the process type and the number of tokens with the same type attribute may be associated with the number of identical processes. A process description, a subnet, can specify the behavior of a type of process and defines variables unique to each process of that type. Each process is a particular and independent instance of an execution of a process description (subnet).

The tokens present in SHLPNs have several attributes: type, identity, environment, etc. In order to introduce compound markings, such attributes are represented by variables with a domain covering the set of values of the attribute.

In the philosopher system, we can use a variable $i$ to replace the identity attribute of the philosopher and the environment variable attribute representing fork tokens to each philosopher process. The domain set of the variable $i$ is $[1, 5]$, i.e., the $(p, i)$ represents anyone among $(p, 1), (p, 2), (p, 3), (p, 4), (p, 5)$, and the $(f, i)$, represents anyone among $(f, 1), (f, 2), (f, 3), (f, 4), (f, 5)$. The compound marking (state)
Table 3.2  The states of the philosopher system with individual markings.

<table>
<thead>
<tr>
<th>State</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T$</td>
</tr>
<tr>
<td>0</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>1</td>
<td>$&lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>3</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>4</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>5</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>6</td>
<td>$&lt; p, 2 &gt;, &lt; p, 4 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>7</td>
<td>$&lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>8</td>
<td>$&lt; p, 1 &gt;, &lt; p, 3 &gt;, &lt; p, 5 &gt;$</td>
</tr>
<tr>
<td>9</td>
<td>$&lt; p, 1 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;$</td>
</tr>
<tr>
<td>10</td>
<td>$&lt; p, 1 &gt;, &lt; p, 2 &gt;, &lt; p, 3 &gt;, &lt; p, 4 &gt;$</td>
</tr>
</tbody>
</table>
Table 3.3 The states of the philosopher system with compound markings.

<table>
<thead>
<tr>
<th>State</th>
<th>Place</th>
<th>T</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt; p, i &gt;, &lt; p, i ⊗ 1 &gt;, 0</td>
<td>0</td>
<td>&lt; f, i &gt;, &lt; f, i ⊗ 1 &gt;, &lt; f, i ⊗ 2 &gt;</td>
<td>&lt; f, i ⊗ 3 &gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt; p, i ⊗ 1 &gt;, &lt; p, i ⊗ 2 &gt;, &lt; p, i ⊗ 3 &gt;, &lt; p, i ⊗ 4 &gt;</td>
<td>&lt; f, i &gt;, &lt; f, i ⊗ 1 &gt;</td>
<td>&lt; f, i ⊗ 2 &gt;, &lt; f, i ⊗ 3 &gt;</td>
<td>&lt; f, i ⊗ 4 &gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt; p, i ⊗ 1 &gt;, &lt; p, i ⊗ 2 &gt;, &lt; p, i ⊗ 3 &gt;, &lt; p, i ⊗ 4 &gt;</td>
<td>&lt; f, i &gt;, &lt; f, i ⊗ 1 &gt;</td>
<td>&lt; f, i ⊗ 2 &gt;, &lt; f, i ⊗ 3 &gt;</td>
<td>&lt; f, i ⊗ 4 &gt;</td>
</tr>
</tbody>
</table>

The markings of Table 3.3 correspond to the Markov chain states shown in Figure 3.9(b) and are obtained by grouping the states from Figure 3.8(a). The transition rates between the grouped states (compound markings) can be obtained after determining the number of possible transitions from one individual marking in each compound marking to any individual marking in another compound marking. In our case, there is one possible transition from only one individual marking of the compound marking \( S_0 \) to each individual marking of the compound marking \( S_1 \) with the same rate. So, the transition rate from \( S_0 \) to \( S_1 \) is \( 5\lambda_1 \). Using a similar argument, we can obtain the transition rate from \( S_1 \) to \( S_2 \) as \( 2\lambda_2 \), and from \( S_1 \) to \( S_0 \) as \( \lambda_2 \). The steady-state probabilities of each compound marking (grouped Markov state) can be obtained as

\[
p_0 = \frac{\lambda_2^2}{5\lambda_1(\lambda_1 + \lambda_2) + \lambda_2^2},
\]

\[
p_1 = \frac{5\lambda_1\lambda_2}{5\lambda_1(\lambda_1 + \lambda_2) + \lambda_2^2},
\]

\[
p_2 = \frac{5\lambda_2^2}{5\lambda_1(\lambda_1 + \lambda_2) + \lambda_2^2}.
\]

The probability of every individual marking of a compound marking is the same and can be easily obtained since the number of individual markings in each compound marking is known.
The previous example has presented the advantage of using high-level Petri nets augmented with exponentially distributed firing times.

### 3.7.3 Formal Definition of SHLPNs

**Definition** A high-level Petri net, HLPN consists of the following elements.

1. A directed graph \((p, t, f)\) where
   - \(p\) is the set of places
   - \(t\) is the set of transitions
   - \(f\) is the set of arcs; \(f \subseteq (p \times t) \cup (t \times p)\)

2. A structure of \(\Sigma\) consisting of some types of individual tokens \((u_i)\) together with some operations \((op_i)\) and relations \((r_i)\), i.e. \(\Sigma = (u_1, ..., u_n; op_1, ..., op_n; r_1, ..., r_k)\).

3. A labeling of arcs with a formal sum of \(n\) attributes of token variables (including the zero attributes indicating a no-argument token).

4. An inscription on some transitions being a logical formula constructed from the operation and relations of the structure \(\Sigma\) and variables occurring at the surrounding arcs.

5. A marking of places in \(p\) with \(n\) attributes of individual tokens.

6. A natural number \(k\) that assigns to the places an upper bound for the number of copies of the same token.

7. **Firing rule:** Each element of \(t\) represents a class of possible changes of markings. Such a change, also called transition firing, consists of removing tokens from a subset of places and adding them to other subsets according to the expressions labeling the arcs. A transition is enabled whenever, given an assignment of individual tokens to the variables which satisfies the predicate associated with the transition, all input places carry enough copies of proper tokens, and the capacity \(K\) of all output places will not be exceeded by adding the respective copies of tokens. The state space of the system consists of the set of all markings connected to the initial marking through such occurrences of firing.

**Definition** A continuous time stochastic high-level Petri net is an HLPN extended with the set of markings related, transition rates, \(\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_R\}\). The value of \(R\) is determined by the cardinality of the reachability set of the net.

To have an equivalence between a timed Petri net and the stochastic model of the system represented by the net, the following two elements need to be specified:

(i) the rules for choosing from the set of enabled transitions, the one that fires, and

(ii) the conditioning based on the past history.
The sojourn time in any state is given by the minimum among the exponential random variables associated with the transitions enabled by that particular state. The SHLPNs do not have immediate transitions. The predicate associated with a transition performs the selection function using the attributes of the tokens in the input places of the transition. A one-to-one correspondence between each marking of a stochastic high-level Petri net and a state of a Markov chain representing the same system can be established.

**Theorem.** Any finite place, finite transition, stochastic high-level Petri net is isomorphic to a one-dimensional, continuous-time, finite Markov chain.

As in the case of SPNs, this isomorphism is based on the marking sequence and not on the transition sequence. Any number of transitions between the same two markings is indistinguishable.

### 3.7.4 Compound Marking of an SHLPN

The compound marking concept is based on the fact that a number of entities processed by the system exhibit an identical behavior and they have a single subnet in the SHLPN model. The only distinction between such entities is the identity attribute of the token carried by the entity. If, in addition, the system consists of identical processing elements distinguished only by the identity attribute of the corresponding tokens, it is possible to lump together a number of markings in order to obtain a more compact SHLPN model of the system. Clearly, the model can be used to determine the global system performance in case of homogeneous systems when individual elements are indistinguishable.

**Definition.** A compound marking of an SHLPN is the result of partitioning an individual SHLPN marking into a number of disjoint sets such that:

(i) the individual markings in a given compound marking have the same distribution of tokens in places, except for the identity attribute of tokens of the same type,

(ii) all individual markings in the same compound marking have the same transition rates to all other compound markings.

Let us now consider a few properties of the compound marking.

(i) A compound marking enables all transitions enabled by all individual markings lumped into it.

(ii) If the individual reachability set of an SHLPN is finite, its compound reachability set is finite.

(ii) If the initial individual marking is reachable with a nonzero probability from any individual marking in the individual reachability set, the SHLPN initial compound marking is reachable with a nonzero probability from any compound marking in the compound reachability set.

We denote by $p_{ij}$ the probability of a transition from the compound marking $i$ to the compound marking $j$ and by $p_{i,x,j}$ the probability of a transition from the individual
marking \( i_n \) to the individual marking \( j_k \), where \( i_n \in i \) and \( j_k \in j \). The relation between the transition probability of individual markings is

\[
p_{ij} = \sum_k p_{i_n,j_k}.
\]

The relation between the transition rate of compound markings and the transition rate of individual markings is

\[
q_j(t) = \frac{d}{dt} \left( \sum_i p_{ij} \right) = \sum_i \frac{d}{dt} \left( \sum_k p_{i_n,j_k} \right).
\]

\[
q_j(t) = \frac{dp_{ij}}{dt} = \sum_k \frac{d(p_{i_n,j_k})}{dt}.
\]

If the system is ergodic, then the sojourn time in each compound marking is an exponentially distributed random variable with average

\[
\left[ \sum_{i \in h} (q_{j,k})_i \right]^{-1},
\]

where \( h \) is the set of transitions that are enabled by the compound marking and \( q_{j,k} \) is the transition rate associated with the transition \( i \) firing on the current compound marking \( j \).

Since there is an isomorphism between stochastic high-level Petri nets and Markov chains, any compound markings of an SHLPN correspond to grouping or lumping of states in the Markov domain.

In order to be useful, a compound marking must induce a correct grouping in the Markov domain corresponding to the original SHLPN. Otherwise, the methodology known from Markov analysis, used to establish whether the system is stable and to determine the steady-state probabilities of each compound marking, cannot be applied. The compound marking of an SHLPN induces a partition of the Markov state space that satisfies the conditions for grouping.

### 3.7.5 Modeling and Performance Analysis of a Multiprocessor System Using SHLPNs

We concentrate our attention on homogeneous systems. Informally, we define a homogeneous system as one consisting of identical processing elements that carry out identical tasks. When modeled using SHLPNs, these systems have subsets of equivalent states. Such states can be grouped together in such a way that the SHLPN model of the system with compound markings contains only one compound state for each group of individual states in the original SPN model. In this case, an equivalence relationship exists among the SHLPN model with compound markings and the original SPN model.
To assess the modeling power of SHLPNs, we consider now a multiprocessor system as shown in Figure 3.10 (a). Clearly, the performance of a multiprocessor system depends on the level of contention for the interconnection network and for the common memory modules.

There are two basic paradigms for interprocessor communication determined by the architecture of the system, namely, message passing and communication through shared memory. The analysis carried out in this section is designed for shared memory communication, but it can be extended to accommodate message passing systems. To model the system, we assume that each processor executes in a number of domains and that the execution speed of a given processor is a function of the execution domain. The model assumes that a random time is needed for the transition from one domain to another.

First, we describe the basic architecture of a multiprocessor system and the assumptions necessary for system modeling, then we present the SHLPN model of the system. The methodology to construct a model with a minimal state space is presented and the equilibrium equations of the system are solved using Markov chain techniques. Performance analysis is based on the steady-state probabilities associated with system states.

**3.7.5.1 System Description and Modeling Assumptions.** As shown in Figure 3.10(a), a multiprocessor system consists of a set of $n$ processors $P = \{P_1, P_2, \cdots, P_n\}$ interconnected by means of an interconnection network to a set of $q$ common memory modules $M = \{M_1, M_2, \cdots, M_q\}$. The simplest topology of the interconnection network is a set of $r$ buses $B = \{B_1, B_2, \cdots, B_r\}$. Each processor is also connected to a private memory module through a private bus.

As a general rule, the time to perform a given operation depends on whether the operands are in local memory or in the common one. When more than one processor is active in common memory, the time for a common memory reference increases due to contention for buses. The load factor $\rho$, is defined as the ratio between the time spent in an execution region located in the common domain and the time spent in an execution region located in the private domain.

A common measure of the multiprocessor system performance is the processing power of a system with $n$ identical processors expressed as a fraction of the maximum processing power ($n$ times the processing power of a single processor executing in its private memory). Consider an application decomposed into $n$ identical processes; in this case the actual processing power of the system depends on the ratio between local memory references and common memory ones.

The purpose of our study is to determine the resource utilization when the load factor increases. The basic assumptions of our model are:

(i) All processor exhibit identical behavior for the class of applications considered. It is assumed that the computations performed by all processors are similar and they have the same pattern of memory references. More precisely, it is assumed that each processor spends an exponentially distributed random time with mean $1/\lambda_1$, while executing in its private domain and then an exponentially distributed random time with mean $1/\lambda_3$ while executing in a common domain. We assume that after
Fig. 3.10 (a) The configuration of the multiprocessor system used in SHLPN modeling. (b) The SHLPN model of the multiprocessor system.

finishing an execution sequence in private memory, each processor draws a random number \( k \), uniformly distributed into the set \([1, q]\), which determines the module
where its next common memory reference will be. This assumption reflects the fact that common memory references are evenly spread into the set of available common memory modules.

(ii) The access time to common memory modules has the same distribution for all modules and there is no difference in access time when different buses are used.

(iii) When a processor acquires a bus it starts its execution sequence in the common memory and it releases the bus only after completing its execution sequence in the common domain.

The first assumption is justified since common mapping algorithms tend to decompose a given parallel problem into a number of identical processes, one for every processor available in the system. The second and the third assumptions are clearly realistic due to hardware considerations.

3.7.5.2 Model Description. Figure 3.10(b) presents an SHLPN model of a multiprocessor system. Although the graph representing the model is invariant to the system size, the state space of the SHLPN model clearly depends on the actual number of processors $n$, common memory modules $q$, and buses $r$. For our example, $n = 5$, $q = 3$ and $r = 2$.

The graph consists of five places and three transitions. Each place contains tokens whose type may be different. A token has a number of attributes; the first attribute is the type of the token. We recognize three different types: $p$–processor, $m$–common memory, $b$–bus. The second attribute of a token is its identity, $id$, a positive integer with values depending on the number of objects of a given type. In our example, when type = $p$, the $id$ attribute takes values in the set $[1,5]$. The tokens residing in place $P$ have a third attribute: the $id$ of the common memory module they are going to refer next.

The meaning of different places and the tokens they contain are presented in Figure 3.10 b). The notation used should be interpreted in the following way: the place $P$ contains the set of tokens of type processors with two attributes $(p, i)$, with $i \in [1,5]$. The maximum capacity of place $P$ is equal to the number of processors.

The transition $E$ corresponds to an end of execution in the private domain and it occurs with a transition rate exponentially distributed with mean $\lambda_1$. As a result of this transition, the token moves into place $Q$ where it selects the next common memory reference. A token in place $Q$ has three attributes $(p, i, j)$ with the first two as before and the third attribute describing the common memory module $j \in [1,3]$ to be accessed by processor $i$. The processor could wait to access the common memory module when either no bus is available or the memory module is busy.

Transition $G$ occurs when a processor switches to execution in common domain, and when the predicate $j = k$, see Figure 3.10 (b) is satisfied. This is a concise representation of the condition that the memory module referenced by the processor $i$ is free. Another way of expressing this condition is: the third attribute of token $(p, i, j)$ is equal to the second attribute of token $(m, k)$. The place $B$ contains tokens representing free buses and the place $M$ contains tokens representing free memory modules. The maximum capacities of these places are equal to the number of buses
Table 3.4  The states of the multiprocessor system model.

<table>
<thead>
<tr>
<th>Marking (State)</th>
<th>Place Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
</tr>
<tr>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>1</td>
</tr>
</tbody>
</table>
and memory modules. The rate of transition $G$ is $\lambda_2$ and it is related to the exponentially distributed communication delay involved in a common memory access. The place $A$ contains tokens representing processes executing in the common domain. The maximum capacity of the places in our graph are:

- Capacity ($P$) = $n$
- Capacity ($Q$) = $n$
- Capacity ($M$) = $q$
- Capacity ($B$) = $r$

- Capacity ($A$) = $\min(n, q, r)$.

The compound markings of the system are presented in Table 3.4. To simplify this table, the following convention is used: Whenever the attributes of the tokens do not have any effect on the compound marking, only the number of the tokens present in a given place is shown. When an attribute of a token is present in a predicate, only that attribute is shown in the corresponding place if no confusion about the token type is possible.

For example, the marking corresponding to state 2 has four tokens in place $P$ (the token type is $p$ according to the model description), two tokens in place $B$ (type = $b$), zero tokens in place $A$. Only the third attribute $i$ of the token present in place $Q$ (the id of the memory module of the next reference) is indicated. Also shown are the ids of the tokens present in place $M$, namely $i$, $j$, and $k$.

As a general rule, it is necessary to specify in the marking, the attributes of the tokens referred to by any predicate that may be present in the SHLPN. In our case, we have to specify the third attribute of the tokens in $Q$ and the second attribute of the tokens in $M$, since they appear in the predicate associated with transition $G$.

Table 3.4 shows the state transition table of the system. For example, state 2 can be reached from the following states, state 1 with the rate $15 \times \lambda_1$, state 18 with the rate $\lambda_3$, and state 19 with the transition rate equal to $\lambda_3$. From state 2, the system goes to state 3 with a transition rate equal to $8 \times \lambda_1$, to state 4 with rate $4 \times \lambda_1$, or to state 17 with rate $\lambda_2$.

State 2 corresponds to the situation when any four processors execute in the private domain and the fifth has selected the memory module of its next common domain reference to be module $i$. It should be pointed out that state 2 is a macrostate obtained due to the use of the compound marking concept and it corresponds to 15 atomic states. These states are distinguished only by the identity attributes of the tokens in two places, $P$ and $Q$, as shown in Table 3.5. The transition rate from the compound marking, denoted as state 1 in Table 3.4, to the one denoted by state 2 is $15 \times \lambda_1$, since there are 15 individual transitions from one individual marking of state 1 to the 15 individual markings in the compound marking corresponding to state 2.
Table 3.5  The 15 individual markings (states) for places $P$ and $Q$, corresponding to the compound marking defined as macrostate 2 in Table 3.4.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt; p, 2 &gt;$</td>
<td>$&lt; p, 1, 1 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 3 &gt;$</td>
<td>$&lt; p, 1, 2 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 4 &gt;$</td>
<td>$&lt; p, 1, 3 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 5 &gt;$</td>
<td>$&lt; p, 2, 1 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 1 &gt;$</td>
<td>$&lt; p, 2, 2 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 3 &gt;$</td>
<td>$&lt; p, 2, 3 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 4 &gt;$</td>
<td>$&lt; p, 3, 1 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 5 &gt;$</td>
<td>$&lt; p, 3, 2 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 1 &gt;$</td>
<td>$&lt; p, 3, 3 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 2 &gt;$</td>
<td>$&lt; p, 4, 1 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 3 &gt;$</td>
<td>$&lt; p, 4, 2 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 4 &gt;$</td>
<td>$&lt; p, 4, 3 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 5 &gt;$</td>
<td>$&lt; p, 5, 1 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 1 &gt;$</td>
<td>$&lt; p, 5, 2 &gt;$</td>
</tr>
<tr>
<td>$&lt; p, 2 &gt;$</td>
<td>$&lt; p, 5, 3 &gt;$</td>
</tr>
</tbody>
</table>

3.7.6  Performance Analysis

To determine the average utilization of different system resources, it is necessary to solve the equilibrium equations and then to identify the states when each resource is idle and the occupancy of that state, and the number of units of that resource that are idle. The following notation is used: Size $[B]_i$ is the occupancy of place $B$ when the system is in state $i$, and $p_i$ is the probability of the system being in state $i$. Then the average utilization of a processor $\eta_P$, a common memory module $\eta_m$, and a bus $\eta_b$ are defined as

$$\eta_P = 1 - \sum_{i \in S} \frac{p_i \times \text{size } [Q]_i}{n},$$

$$\eta_m = 1 - \sum_{i \in S} \frac{p_i \times \text{size } [M]_i}{q},$$

$$\eta_b = 1 - \sum_{i \in S} \frac{p_i \times \text{size } [B]_i}{l}.$$

The load for common resources is defined as

$$\rho = \frac{\lambda_1}{\lambda_2}.$$  

The number of original states is very high, larger than 500, and we have reduced the model to only 51 states. As mentioned earlier, the same conceptual model can
be used to model a message passing system. In such a case, $\lambda_2$ will be related to the time necessary to pass a message from one processor to another, including the processor communication overhead at the sending and at the receiving site, as well as the transmission time dependent upon the message size and the communication delay. In case of a synchronization message passing system, $\lambda_3$ will be related to the average blocking time in order to generate a reply.

3.8 MODELING HORN CLAUSES WITH PETRI NETS

The material in this section covers applications of the net theory to modeling of logic systems and follows closely Lin et al. [15]. A Horn clause of propositional logic has the form

$$ B \leftarrow A_1 \land A_2, \ldots, \land A_n. $$

This notation means that holding of all conditions $A_1$ to $A_n$ implies the conclusion $B$. Logical connectiveness is expressed using the $\leftarrow$ (implication) and $\land$ (conjunction) symbols. A Horn clause is a clause in which the conjunction of zero or more conditions implies at most one conclusion.

There are four different forms of Horn clauses. The Petri net representations of Horn clauses are:

1. The Horn clause with non-empty condition(s) and conclusion

$$ B \leftarrow A_1 \land A_2, \ldots, \land A_n \text{ with } n \geq 1. $$

For example, the clause $C \leftarrow A \land B$ is represented by the Petri net in Figure 3.11(a). When the conditions $A$ and $B$ hold tokens, transition $t$ fires, and a token is deposited in place $C$, i.e., the conclusion $C$ is true.

2. The Horn clause with empty condition(s)

$$ B \leftarrow. $$

This type of Horn clause is interpreted as an assertion of a fact. A fact $B$ can be represented in a Petri net model as a transition system with a source transition, as shown in Figure 3.11(b). The source transition $t$ is always enabled and this means that the formula $B$ is always true.

3. The Horn clause with empty conclusion

$$ \leftarrow A_1 \land A_2, \ldots, \land A_n \text{ with } n \geq 1. $$

This type of Horn clause is interpreted as the goal statement, which is in the negation form of what is to be proven. In a Petri net model a condition such as ‘$A$ and $B$’ is represented as a goal transition system with a sink transition, as shown in Figure 3.11(c).
4. The null clause, which is interpreted as a contradiction. There is no representation of such clause, the *empty net* is not defined in the net theory.

![Fig. 3.11](image-url) Modeling Horn clauses with Petri nets. (a) A Horn clause with two conditions and a conclusion. (b) A Horn clause with no condition. (c) A Horn clause with empty conclusion.

Given a set of Horn clauses consisting of \( n \) clauses and \( m \) distinct symbols, the \( n \times m \) incidence matrix \( F = [F_{ij}] \) of a Petri net corresponding to the set of clauses can be obtained by the following procedure given by Murata and Zhang [22].

**Step 1:** Denote the \( n \) clauses by \( t_1, \ldots, t_n \). The clause \( t_i \) represents the \( i^{th} \) row of \( F \).

**Step 2:** Denote the \( m \) predicate symbols by \( p_1, \ldots, p_m \). The symbol \( p_j \) represents the \( j^{th} \) column of \( F \).

**Step 3:** The \((i, j)^{th}\) entry of \( F \), \( F_{ij} \), is the sum of the arguments in the \( i^{th} \) clause and the \( j^{th} \) symbol. The sum is taken over all the \( j^{th} \) symbols appearing in the \( i^{th} \) clause. All the arguments to the left side of the \( \leftarrow \) operator are taken as positive, and all the arguments to the right side of it are taken as negative. Thus the elements \( F_{ij} \) can be either ‘0’, or ‘1’ or ‘-1’.

The following example shows the translation procedure.

**Example:** (based on Peterka and Murata [23]).

Consider the following set of Horn clauses represented in the conventional way

1) \( A \)
2) \( B \)
3) \( A \land B \rightarrow C \)
4) \( C \land B \rightarrow D \)
5) \( D \rightarrow A \)
6) \( D \rightarrow C \)

To prove that \( D \land C \) is true, one can apply the satisfiability principle. Let \( S \) be a set of first order formulae and \( G \) be a first order formula. \( G \) is a logic consequence of \( S \) iff \( S \cup \{\neg G\} \) is unsatisfiable. The following result is obtained by adding the negation of \( D \land C \) to the set of clauses
The Petri net representation of this set of Horn clauses and its incidence matrix are shown in Figure 3.12.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
-1 & -1 & 1 & 0 \\
0 & -1 & -1 & 1 \\
1 & 0 & 0 & -1 \\
0 & 0 & 1 & -1 \\
0 & 0 & -1 & -1 \\
\end{bmatrix}
\]

Fig. 3.12 The incidence matrix and the Petri net for the set of Horn clauses in the example from Section 3.8.

Sinachopoulos [29], Lautenbach [14], and Murata [23] have investigated the necessary and sufficient conditions for a set of Horn clauses to contain a contradiction based on analysis of the Petri net model of such clauses. These conditions are:

**Theorem.** A necessary net theoretical condition for a set of clauses \( C_2 \), to be unsatisfiable is that the net representation of \( C_2 \) has a non-negative \( T \)-invariant.

**Theorem.** A sufficient net theoretical condition for a set of Horn clauses \( C_2 \), to be unsatisfiable is that \( C_2 \) contains at least one source transition, at least one sink transition, and has a nonzero \( T \)-invariant.

**Theorem.** Let \( N \) be a Petri net representation of a set of Horn clauses. Let \( t_g \) be a goal transition in \( t \). There exists a firing transition sequence that reproduces the empty marking \( (M = 0) \) and fires the goal transition \( t_g \) in \( N \) iff \( N \) has a \( T \)-invariant \( X \) such that \( X \geq 0 \) and \( X(t_g) \neq 0 \). \( X \) is a vector and the value of its \( t_g \) element is given by \( X(t_g) \).
3.9 WORKFLOW MODELING WITH PETRI NETS

The idea of using Petri nets for modeling and enactment of workflows can be traced back to a paper published in 1993 by Ellis and Nutt [6]. It was soon discovered that Petri nets support modeling of dynamic changes within workflow systems [4] and the net-based workflow modeling was included in a book published in 1996 on modeling, architecture, and implementation of workflow management systems [9]. The WorkFlow nets and the concept of workflow inheritance were introduced in 1999 by van der Aalst and Basten [1].

Recall from Chapter 1 that in workflow management we handle cases, individual activations of a workflow, and for each case we execute tasks or activities in a certain order. Each task has pre-conditions that must be satisfied before the task can be executed; after the execution of a task its postconditions must hold.

3.9.1 Basic Models

The basic Petri net workflow modeling paradigm is to associate tasks with transitions, conditions with places, and cases with tokens. A workflow is modeled by a net with a start place, $p_s$, corresponding to the state when the case is accepted for processing and a final place, $p_f$, corresponding to the state when the processing of the case has completed successfully. We also require that every condition and activity contributed to the processing of the case. This requirement means that every node, be it a place or a transition, be located on a path from $p_s$ to $p_f$. These informal requirements translate into the following definition

**Definition – Workflow Net.** The P/T net $N = (p, t, f, l)$ is a Workflow net iff: (a) $N$ has one start and one finish place, $p_s$ and $p_f$ and (b) $N$, its short-circuit counterpart is strongly connected.

The initial marking, $s_{init}$, of a workflow net corresponds to the state when there is only one token in the start place, $p_s$, and the final marking $s_{final}$ corresponds to the state when there is only one token in the finish place, $p_f$.

We are interested in Workflow nets that have a set of desirable structural and behavioral properties. First, we require a net to be safe; this means that in all markings every place has at most one token. Indeed, places in the net correspond to conditions that can either be true and the place contains one token, or false and the place contains no tokens. Second, we require that it should be always possible to reach the final marking, $s_{final}$ from the initial marking $s_{init}$. This requirement simply implies that we can always complete a case successfully. Third, we require that there are no dead transitions; for each activity of the workflow there is an execution when the activity is carried out. This set of minimal requirements leads to the so-called soundness of the Workflow net.

**Definition – Sound Workflow Net.** The workflow net $[N, s_{init}]$ is sound iff: (i) it is safe, (ii) for any reachable marking $s \in [N, s_{init}]$ $s_{final} \in [N, s]$, and (iii) there are no dead transitions.
A workflow net \([N, s_{init}]\) is sound iff its associated short-circuit net, \([\tilde{N}, s_{init}]\), is live and safe [1]. Workflow definition languages used in practice lead to free-choice Workflow nets and for such nets the soundness can be decided in polynomial time.

### 3.9.2 Branching Bisimilarity

We often partition the set of objects we have to manipulate, in equivalence classes such that all objects with a set of defining properties belong to the same class. This approach allows us to structure our knowledge, to accommodate the diversity of the environment, and to formulate consistent specifications for systems with similar or identical functionality.

The basic idea of branching bisimilarity is to define classes of equivalent systems based on the states the systems traverse in their evolution and the actions causing transitions from one state to another. When defining this equivalence relationship we can insist on a stronger or weaker similarity, thus, we can define two different types of relationships.

Informally, if two systems are capable of replicating every action of each other and traversing similar states they are strongly equivalent for the corresponding set of consecutive actions. Consider two chess players; every time one makes a move, the other one is able to mirror the move. We bend the traditional chess rules and after each pair of moves of the two players, either player may move first. Clearly, this mirroring process can only be carried out for a relatively small number of moves. At some point in time one of the players will stop replicating the other’s move because of either conflicts due to the rules of the game or because it will lead to a losing position.

To define a weaker notion of equivalence we introduce the concept of silent or internal actions, actions that cannot be noticed by an external observer. For example, a casual listener of an audio news broadcast may not distinguish between a digital reception over the Internet, played back by a Real Networks player running on a laptop, and an analog broadcast played by a traditional radio receiver.

The two systems have some characteristics in common: both receive an input information stream, process this stream to generate an analog audio signal, and finally feed this signal into the loudspeakers. Yet, internally, the two systems work very differently. One is connected to the Internet and receives a digital input stream using a transport protocol, unpacks individual voice samples in the same packet, interpolates to re-construct missing samples as discussed in Chapter 5, then converts the digital samples into an analog signal. The other has an antenna and receives a high-frequency radio signal, amplifies the signal, extracts the analog audio signal from the high-frequency carrier using analog audio circuitry.

Clearly, the equivalence relationship is an ad hoc one; it only reflects the point of view of a particular observer. A more astute observer would notice differences in the quality of the sound produced by the two systems. When modeling the two processes described in this example, all actions of the digital Internet audio player that are different from those performed by the analog receiver, and vice versa, are defined as silent actions.
We now first define the concept of strong and weak bisimulation and then we introduce branching bisimilarity of Petri nets.

**Definition – Strong Bisimulation.** A binary relation $R$ over the states $s_i$ of a labeled transition system with actions $\alpha \in \text{Actions}$ is a strong bisimulation iff:

$$\forall(s_1, s_2) \in R, \forall(\alpha) \in \text{Actions}$$

$$\left( s_1 \xrightarrow{\alpha} s_1' \Rightarrow \exists s_2 \xrightarrow{\alpha} s_2' \land s_1'R s_2' \right) \land \left( s_2 \xrightarrow{\alpha} s_2' \Rightarrow \exists s_1 \xrightarrow{\alpha} s_1', s_1'R s_2' \right)$$

In this equation $s_1 \xrightarrow{\alpha} s_1'$ means that the system originally in state $s_1$ moves to state $s_1'$ as a result of action $\alpha$. Two states $s_1$ and $s_2$ are strongly bisimilar iff there is a strong bisimulation $R$ such that $s_1R s_2$. This definition can be extended to two different systems by setting them next to each other and considering them as a single system. The largest strong bisimulation, the one with the highest number of actions, is an equivalence relation called **strong bisimulation equivalence**.

If processes contain internal actions, labeled $\tau$ denote:

$$\xrightarrow{\alpha} \stackrel{\text{def.}}{=} \left( \left( \xrightarrow{\tau} \right)^* \xrightarrow{\alpha} \left( \xrightarrow{\tau} \right)^* \right), \alpha \in \text{Actions}$$

and

$$\xrightarrow{\tau} \stackrel{\text{def.}}{=} \left\{ \begin{array}{ll} \xrightarrow{\alpha} & \text{if } \alpha \neq \tau \\ \left( \xrightarrow{\tau} \right)^* & \text{if } \alpha = \tau \end{array} \right\}$$

**Definition – Weak Bisimulation.** A binary relation $R$ over the states $s_i$ of a labeled transition system with actions $\alpha \in \text{Actions}$ is a weak bisimulation if:

$$\forall(s_1, s_2) \in R, \forall(\alpha) \in \text{Actions}$$

$$\left( s_1 \xrightarrow{\alpha} s_1' \Rightarrow \exists s_2 \xrightarrow{\alpha} s_2' \land s_1'R s_2' \right) \land \left( s_2 \xrightarrow{\alpha} s_2' \Rightarrow \exists s_1 \xrightarrow{\alpha} s_1', s_1'R s_2' \right)$$

Two states $s_1$ and $s_2$ are weakly bisimilar iff there is a weak bisimulation $R$ such that $s_1R s_2$. This definition can be extended to two different systems by setting them next to each other and considering them as a single system. The largest weak bisimulation, the one with the highest number of actions, is an equivalence relation called **weak bisimulation equivalence**.

Note that in this case instead of an observable action $\alpha$ we require that when one of the systems reaches state $s_i'$ as a result of the action $\alpha$ in state $s_j$, then the other system in state $s_j$ reaches state $s_j'$ after zero or more internal or silent actions followed by $\alpha$, possibly followed by zero or more silent actions: $s_1 \xrightarrow{\alpha} s_1'$ and $s_2 \xrightarrow{\alpha} s_2'$ respectively.

Strong bisimilarity between a Petri net and a finite-state system is decidable [10] while the weak bisimilarity between a Petri net and a finite-state system is undecidable [11].

The weak bisimulation relation is used to construct classes of equivalent Petri nets. We define an equivalence relation among marked labeled P/T nets by introducing silent actions modeled as transitions with a special label $\tau$. Such transitions correspond to
internal actions that are not observable. Two marked labeled P/T nets are branching
bisimilar if one of them is able to simulate any transition $\alpha$ of the other one after
performing a sequence of zero or more silent actions. The two must satisfy an
additional requirement: both must either deadlock or terminate successfully.

**Definition – Behavioral Equivalence of Workflow Nets.** Two workflow nets,
$[N, s_{init}]$ and $[Q, q_{init}]$ are behaviorally equivalent iff a branching bisimilarity $R$
relation between them exists.

$$[N, s_{init}] \equiv [Q, q_{init}] \iff ([N, s_{init}] R [Q, q_{init}])$$

### 3.9.3 Dynamic Workflow Inheritance

The theoretical foundation for the concept of dynamic workflow inheritance discussed
now is based on work done by van Aalst and Basten [1] for workflow modeling
and analysis on the equivalence relation among labeled P/T, nets, called branching
bisimilarity. This equivalence relation is related to the concept of observable behavior.
We distinguish two types of actions, those that are observable and the silent ones,
actions we cannot observe. In this context an action is the firing of a transition.
Two P/T nets that have the same observable behavior are said to be equivalent. A
labeled P/T net can evolve into another one through a sequence of silent actions and
a predicate expressing the fact that a net can terminate successfully after executing
one or more silent actions.

The intuition behind inheritance is straightforward, given two workflows $v$ and $w$,
we say that $w$ is a subclass of $v$ or extends $v$ iff $w$ inherits "some" properties of $v$.
Conversely, we say that $v$ is a superclass of $w$. The subclass may redefine some of
the properties of its superclass.

The components of workflows are actions; hence a necessary condition for workflow
$w$ to be a subclass of $v$ is to contain all the actions of $v$ and some additional ones.
But the action subset relation between an extension and its superclass is not sufficient,
we need to relate the outcomes of the two workflows, to make them indistinguishable
from one another under some conditions imposed by actions in $w$ but not in $v$. Two
such conditions are possible: (a) block the additional actions, and (b) consider them
as unobservable, or silent.

Two basic types of dynamic inheritance, have been defined [1]. (1) *Protocol
inheritance:* if by blocking the actions in $w$ that are not present in $v$ it is not possible
to distinguish between the behavior of the two workflows we say that $w$ inherits
the protocol of $v$. (2) *Projection inheritance:* $w$ inherits the projection of $v$ if by
making the activities in $w$ that are not in $v$ unobservable, or silent it is not possible
to distinguish between the behavior of the two workflows.

Figure 3.13 inspired from van der Aalst and Basten [1] illustrates these concepts.
Each workflow is mapped into a P/T net. The P/T net transitions correspond to
workflow actions. The workflows in (b), (c), (d), and (e) are subclasses of the workflow
in (a) obtained by adding a new action D. The workflow in (b) is a subclass with respect
to projection and protocol inheritance. When either blocking or hiding action D, (b) is
Fig. 3.13  Dynamic workflow inheritance. The workflows in (b), (c), and (d) are subclasses of the workflow in (a) obtained by adding a new action D. The workflow in (b) is a subclass with respect to projection and protocol inheritance. When either blocking or hiding action D, (b) is identical to (a). The workflow in (c) is a subclass with respect to protocol inheritance but not under projection inheritance. When blocking activity D, (c) is identical to (a) but it is possible to skip activity B by executing action D. The workflow in (c) is not a subclass with respect to projection inheritance. The workflow in (d) is a subclass of the one in (a) with respect to projection inheritance. The workflow in (e) is not a subclass with respect to either projection or protocol inheritance.
tensive information related to Petri nets, including: groups working on different aspects on net theory and applications; standards; education; mailing lists; meeting announcements; and the Petri nets newsletter.

A comprehensive bibliography with more than 2500 entries is available from http://www.daimi.au.dk/PetriNets/bibl/aboutpnbibl.html.

A fair number of Petri net software tools have been developed over the years. A database containing information about more than fifty Petri net tools can be found at http://www.daimi.au.dk/PetriNets/tools/db.html.

The vast literature on Petri nets includes the original paper of Carl Adam Petri [25] and his 1986 review of the field [26]. The book by Peterson [24] is an early introduction to system modeling using Petri nets; the tutorial by Murata [21] provides an excellent introduction to the subject.

The proceedings of the conferences on Petri nets and applications, held annually since early 1980, have been published by Springer-Verlag in the Lecture Notes on Computer Science series, e.g., LNCS volumes 52, 254, 255, and so on. Conferences on Petri nets and performance models have taken place every two years since 1985 and the papers published in the proceedings of these conferences cover a wide range of topics from methodology to tools and to algorithms for analysis [17].

The book edited by Jensen and Rozenberg [12] provides a collection of papers on the theory and application of high-level nets. Timed Petri nets (TPNs), are discussed by Zuberek [30]. Stochastic Petri nets are presented by Molloy [19], Florin and Natkin [7], Marsan and his co-workers [18], Lin and Marinescu [16].

Applications to performance evaluation are discussed by Sifakis [28], Ramamoorthy [27], and Murata [20]. Applications to modeling logic systems are presented by Murata et al. [22, 23], Marinescu et al. [3, 15], and others [14, 29]. Applications of Petri nets to workflow modeling and analysis are the subject of papers by Ellis et al. [4, 6, 5], van der Aalst and Basten [1, 2], and others [9]. Work on branching bisimulation and Petri nets is reported in [10, 11].

3.11 EXERCISES AND PROBLEMS

Problem 1. A toll booth accepts one and five dollar bills, one dollar, half dollar, and quarter coins. Passenger cars pay $1.75 or $3.25, depending on the distance traveled, while trucks and buses pay $3.50 and $6.50. The machine requires exact change.

Design a Petri net representing the state transition diagram of the toll booth.

Problem 2. Translate the formal description of the home loan application process of Problem 2 in Chapter 1 into a Petri net; determine if the resulting net is a free-choice net.

Problem 3. Translate the formal description of the automatic Web benchmarking process of Problem 3 in Chapter 1 into a Petri net; determine if the resulting net is a free-choice net.
Problem 4. Translate the formal description of the grant request reviewing process of Problem 5 in Chapter 1 into a Petri net; determine if the resulting net is a free-choice net.

Problem 5. Construct the coverability graphs of the net in Figure 3.2(j) and of the nets you have constructed for Problems 1, 2, 3, and 4 above.

Problem 6. Show that if the token count on the subset of places $P' \subset P$ never changes under arbitrary transition firings, the condition $F \times y = 0$, where $y$ is an vector of integers with $|P|$ components and has nonzero components corresponding to all places in $P'$.

Problem 7. Compute the S-invariants of:
(i) the net in Figure 3.14;
(ii) the four nets you have constructed for Problems 1, 2, 3, and 4 above.

Problem 8. Compute the T-invariants of the four nets you have constructed for Problems 1, 2, 3, and 4 above.

Problem 9. Provide a definition for the Petri net language described by the net in Figure 3.15.

Problem 10. Prove that a live free-choice Petri net $\langle N, s_0 \rangle$ is safe iff it is covered by strongly connected state machines and each state machine has exactly one token in $s_0$. 

Fig. 3.14 Petri net for Problem 7.
**Fig. 3.15** A net similar to the one in Figure 3.5.

**REFERENCES**


