# Petri Nets*

JAMES L. PETERSON

*Department of Computer Sciences, The University of Texas, Austin, Texas 78712*

Over the last decade, the Petri net has gained increased usage and acceptance as a basic model of systems of asynchronous concurrent computation. This paper surveys the basic concepts and uses of Petri nets. The structure of Petri nets, their markings and execution, several examples of Petri net models of computer hardware and software, and research into the analysis of Petri nets are presented, as are the use of the reachability tree and the decidability and complexity of some Petri net problems. Petri net languages, models of computation related to Petri nets. and some extensions and subclasses of the Petri net model are also briefly discussed.

*Keywords and Phrases:* Petri nets, system models, asynchronous concurrent events.

*CR Categories*: 1.3, 5.29, 8.1

## INTRODUCTION

A *Petri net* is an abstract, formal model of information flow. The properties, concepts, and techniques of Petri nets are being developed in a search for natural, simple, and powerful methods for describing and analyzing the flow of information and control in systems, particularly systems that may exhibit asynchronous and concurrent activities. The major use of Petri nets has been the modeling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the concurrence, precedence, or frequency of these occurrences.

Since many readers may be unfamiliar with Petri nets, we first present a very brief and informal introduction to their fundamentals and history. Then we consider several aspects of Petri nets in more detail. We begin, in Section 2, by considering the use of Petri nets for modeling systems of parallel or concurrent activities. Section 3 presents a more formal definition and discussion of the fundamental concepts and notations of Petri nets. Section 4 considers the extensive body of research dealing with the analysis of Petri nets, their advantages, and their limitations. Petri net languages are presented in Section 5. Finally, in Section 6, we consider some of the many variations of Petri nets that have been defined, both as generalizations of Petri nets and as subclasses of the general model; the more general models have certain advantages for modeling, while the more restricted models have certain advantages for analysis.

## 1. OVERVIEW

Figure 1 shows a simple Petri net. The pictorial representation of a Petri net as a graph used in this illustration is common practice in Petri net research. The Petri net graph models the static properties of a system, much as a flowchart represents the static properties of a computer program.

* This work was supported, in part, by the National Science Foundation, under Grant Number MCS75-16425.



Pet77

CONTENTS



The graph contains two types of nodes: circles (called *places*) and bars (called *transitions*). These nodes, places and transitions, are connected by directed arcs from places to transitions and from transitions to places. If an arc is directed from node $i$ to node $j$ (either from a place to a transition or a transition to a place), then $i$ is an *input* to $j$, and $j$ is an *output* of $i$. In Fig. 1, for example, place $p_1$ is an input to transition $t_2$, while places $p_2$ and $p_3$ are outputs of transition $t_2$.

In addition to the static properties represented by the graph, a Petri net has dynamic properties that result from its execution. Assume that the execution of a computer program represented by a flowchart is exhibited by placing a marker on the flowchart to mark the instruction being executed, and that as the execution progresses, the marker moves around the flowchart. Similarly, the execution of a Petri net is controlled by the position and movement of markers (called *tokens*) in the Petri net. Tokens, indicated by black dots, reside in the circles representing the places of the net. A Petri net with tokens is a *marked Petri net*.

The use of the tokens rather resembles a board game. These are the rules: Tokens are moved by the *firing* of the transitions of the net. A transition must be *enabled* in order to fire. (A transition is enabled when all of its input places have a token in them.) The transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output places of the transition. In the marked Petri net of Fig. 2, for example, the transition $t_2$ is enabled since it has a token in its input place ($p_1$). Transition $t_5$, on the other hand, is not enabled since one of its inputs ($p_3$) does not have a token. If $t_2$ fires, the marked Petri net of Fig. 3 results. The firing of transition $t_2$ removes the enabling token from place $p_1$ and puts tokens in $p_2$ and $p_3$, the two outputs of $t_2$.

The distribution of tokens in a marked Petri net defines the state of the net and is called its *marking*. The marking may change as a result of the firing of transitions. In different markings, different transitions may be enabled. For example, in the marked net of Fig. 3 three transitions are enabled: $t_1$, $t_3$, and $t_5$, none of which were enabled in the marking of Fig.

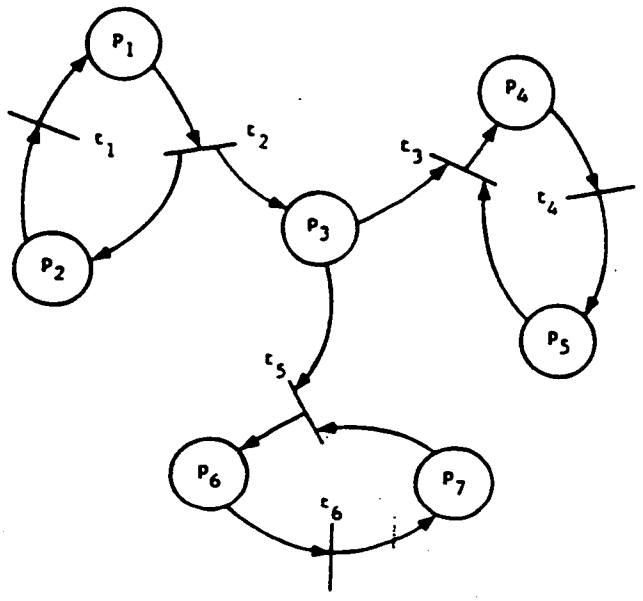



FIGURE 1. A simple graph representation of a Petri net.

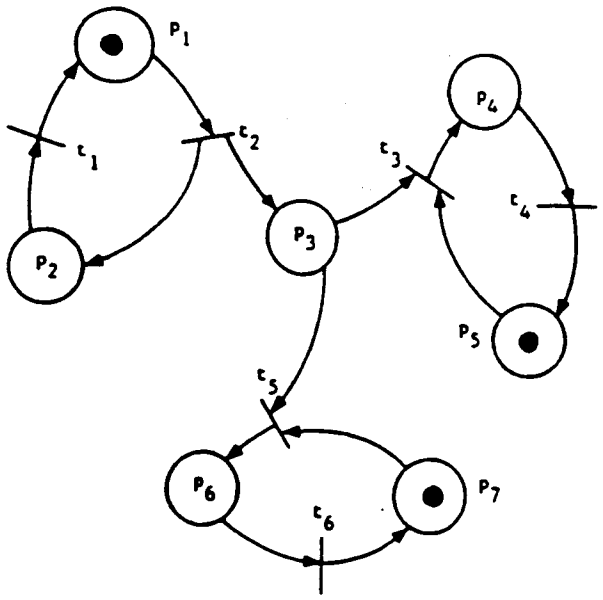



FIGURE 2. A marked Petri net.

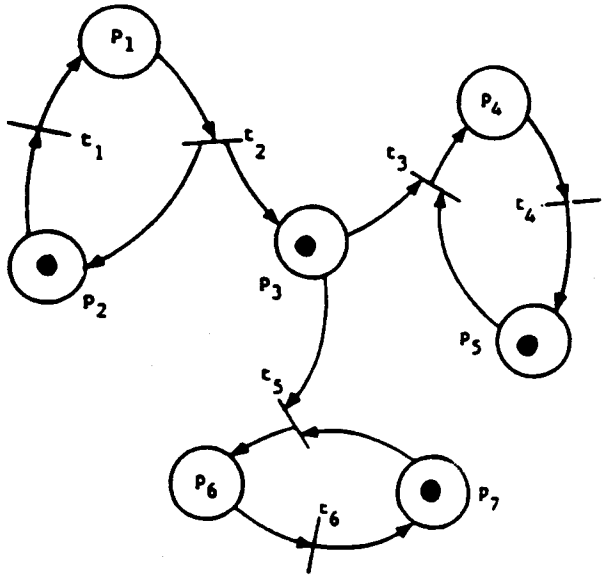



FIGURE 3. The marking resulting from firing transition $t_2$ in Fig. 2. Note that the token in $p_1$ was removed and tokens were added to $p_2$ and $p_3$.

2. In this situation, we have a choice as to which transition will fire next. Figure 4 shows the three possible resultant markings; from each of these, other markings may then be reached since transition firings may continue as long as there is an enabled transition.

Note that in the marking of Fig. 4a, transitions $t_3$ and $t_5$ remain enabled, and transition $t_2$ is also enabled; if transition $t_2$ fires, the resulting marking will have two tokens in place $p_3$. In the marking of Fig. 4b, transition $t_1$ remains enabled but transition $t_5$ has been disabled since there is no longer a token in place $p_3$. In the marking of Fig. 4c, transition $t_3$ has become disabled. Firing either of transitions $t_3$ or $t_5$ disables the other; they are said to be *in conflict*.

This simple, vague, and incomplete example of a Petri net is meant to give a brief introduction to the basic concepts of Petri nets. It also raises some questions for further consideration. For example, the markings of Figs. 3, 4a, 4b, and 4c were generated from the marking of Fig. 2 by firing transitions. Can we characterize the class of markings that may be reached from a given marked Petri net? Can we characterize the class of sequences of transition firings that are possible from a marked Petri net? What interesting properties can Petri nets have and how can these properties be tested for?

Some of these questions are of interest for Petri nets as abstract formal entities. Other questions relate to Petri nets in their function as models of other systems, existing or proposed. For example, the net of Fig. 2 can represent a producer–consumer problem [25] with one producer (places $p_1$ and $p_2$) and two consumers (places $p_4$, $p_5$ and $p_6$, $p_7$). The items produced by the producer are passed to the consumers. This is modeled by place $p_3$ and the tokens "produced" by transition $t_2$ and "consumed" by transitions $t_3$ and $t_5$. For this interpretation of the net, we may be interested in how far the producer can get ahead of the consumers (the maximum number of tokens in $p_3$), whether the consumers could get ahead of the producer or consume the same item twice, and so on. The use of Petri nets in modeling is discussed in Section 2.

Although Petri nets are basically very simple, they may be approached and utilized in a large number of ways. Petri nets can be considered as formal automata and investigated either as automata or as generators of formal languages [37, 79]. Questions dealing with the theory of computational complexity have been raised [64, 49]. Petri nets have associations with the study of linear algebra [69], Presburger arithmetic [52], and graph theory. They

are a major model of concurrent systems [6], particularly computer systems. They are of interest in some areas of hardware design, description and construction, software systems, and the interactions between design and implementation.

Because of the breadth of application and depth of research into Petri nets, we can only touch here on many of the results. We refer the interested reader to the original works cited in the Bibliography for the proofs and details of much of the research.

### History

The theory of Petri nets has developed from the work of Carl Adam Petri, A. W. Holt, Jack Dennis, and many others. Petri nets originated in the early work of Petri, in Germany, who in his thesis [80], developed a new model of information flow in systems. This model was based on the concepts of asynchronous and concurrent operation by the parts of a system and the realization that relationships between the parts could be represented by a graph, or net.

The ideas of Petri came to the attention of a group of researchers at Applied Data Research, Inc., working on the Information Systems Theory Project [43]. This group, led by Anatol Holt, developed the theory of "systemics" [44] which was concerned with the representation and analysis of systems and their behavior. It was this work which provided the early theory, notation, and representation of Petri nets, and showed how Petri nets could be applied to the modeling and analysis of systems of concurrent processes.

Applied Data Research's associations with Project MAC at MIT, and particularly the Computation Structures Group under the direction of Jack Dennis, introduced the concepts of Petri nets to this latter group. The Computation Structures Group has been a most productive source of research and literature in this field, publishing several PhD theses and numerous reports and memos on Petri nets [73, 32, 34, 84, 38, 28]. Two pertinent conferences have been held by the Computation Structures Group: the Project MAC Conference on Concurrent Systems and Parallel Computation at Woods Hole in 1970, [23] and the Conference on Petri Nets and Related Methods at MIT in 1975.

From the work at Applied Data Research and MIT, the use of Petri nets has spread widely. A large amount of research has been done on both the nature and the application of Petri nets, and their use seems to be expanding. The simplicity and power of Petri nets make them excellent tools for working with asynchronous concurrent systems. Unfortunately much of the work on Petri nets is in the form of theses, dissertations, reports, and memos that are not readily available nor in wide circulation. This paper is an attempt to remedy this situation; it is intended as both a survey and a tutorial on Petri nets.

It should be noted that Petri nets can be viewed in many different ways; we cannot present here all such views. In part due to the difficulty of obtaining literature on Petri nets and the newness of the theory, the terminology, notation, and emphasis have varied widely in research on this subject. This problem is also caused by the power of Petri nets and the resultant diversity of applications.

Petri has expanded upon his original theory, continuing work on the basic concepts of information flow and the structure of concurrent systems. This has resulted in a form of general systems theory called net theory [81, 82] which is related to topology. This research, involving the fundamental nature of information and its control, has stimulated a wealth of further research in Europe, particularly at the Institut für Informationssystemforschung of the Gesellschaft für Mathematik und Datenverarbeitung in Bonn. While starting with the same fundamental concepts as the work in the United States, this work developed in a different direction, evolving into a more general and abstract theory.

Holt also has continued to develop new concepts from the original work on Petri nets, concentrating on the development of tools for the representation and analysis of systems. His work has centered mainly on research into the fundamental aspects of concurrency and conflict in systems with multiple parts.

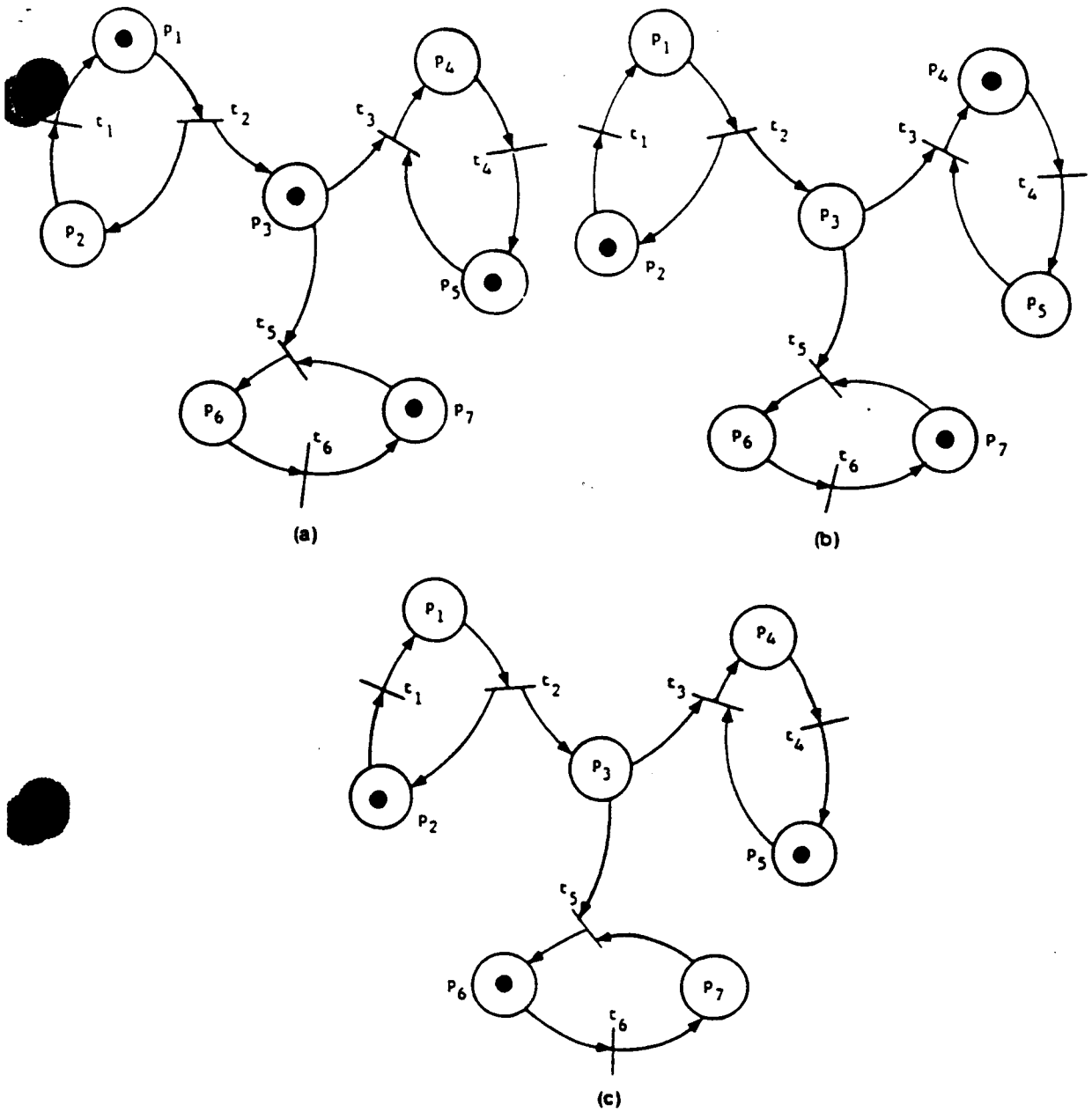



FIGURE 4. Markings resulting from the firing of different transitions in the net of Fig. 3. (a) Result of firing transition $t_1$; (b) Result of firing transition $t_3$; (c) Result of firing transition $t_5$.

In contrast to the work of Petri, Holt, and many European researchers, which emphasizes the fundamental concepts of systems, the work at MIT and many other American research centers concentrates on those mathematical aspects of Petri nets that are more closely related to automata theory. (This paper is written from the latter point of view.) This approach is motivated by a desire to analyze systems by modeling them as Petri nets, and then manipulating the Petri nets to derive properties of the modeled systems. This requires the development of techniques for analyzing Petri nets in order to answer questions similar to those raised earlier (e.g., what markings are reachable in a given Petri net? What sequences of transition firings are possible? etc.). This mechanistic approach is quite different in orientation from the more philosophical approaches of Holt and Petri.

## 2. MODELING WITH PETRI NETS

In many sciences, a phenomenon is studied by examining not the actual phenomenon itself but rather a *model* of the phe-

nomenon. A model is a representation, often in mathematical terms, of what are felt to be the important features of the object under study. By the manipulation of the representation, it is hoped that new knowledge about the modeled phenomenon, and the model itself, will be obtained without the cost, inconvenience, or danger of manipulating the real phenomenon itself. For example, much work on atomic energy has been done by modeling because of the expense and danger of handling radioactive materials.

Most modeling uses mathematics. The important features of many physical phenomena can be described numerically and the relations between these features described by equations or inequalities. Particularly in physics and chemistry, properties such as mass, momentum, acceleration, position, and forces, are describable by mathematical equations. To successfully utilize the modeling approach, however, requires a knowledge of both the modeled phenomena and the modeling techniques. Thus, mathematics has developed as a science in part because of its usefulness in modeling phenomena in other sciences. For example, the differential calculus was developed in direct response to the need for a means to model continuously changing properties such as position, velocity, and acceleration in physics.

Petri nets are also a modeling tool. They were devised for use in the modeling of a specific class of problems, the class of discrete-event systems with concurrent or parallel events. Petri nets model systems, and particularly two aspects of systems, *events* and *conditions*, and the relationships among them [44]. In this view, in a system, at any given time, certain conditions will hold. The fact that these conditions hold may cause the occurrence of certain events. The occurrence of these events may change the state of the system, causing some of the previous conditions to cease holding, and causing other conditions to begin to hold.

A simple example might be that the simultaneous holding of both the condition 'A card reader is needed' and the condition

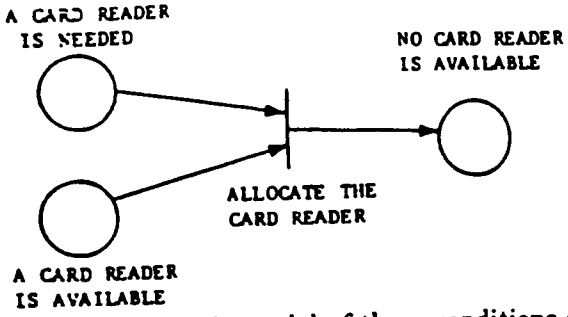



FIGURE 5. A simple model of three conditions and an event.

'A card reader is available' might cause the event 'Allocate the card reader' to occur. The occurrence of this event results in the ceasing of the conditions 'A card reader is needed' and 'A card reader is available', while causing the condition 'No card reader is available' to become true. These events and conditions, and their relationships, may be modeled as in Fig. 5, where we are using places to represent conditions and transitions to represent events. Note that other conditions, such as 'The card reader is allocated', may also hold in the system even though they are not modeled.

More complicated systems may also be modeled in this manner. Consider for example the following description of a computer system:

- Jobs appear and are put on an input list. When the processor is free, and there is a job on the input list, the processor starts to process the job.
- When the job is complete, it is placed on an output list, and if there are more jobs on the input list, the processor continues with another job; otherwise it waits for another job.

This is a very simple system composed of several elements: the processor, the input list, the output list, and the jobs. We can identify several conditions of interest:

- The processor is idle;
- A job is on the input list;
- A job is being processed;
- A job is on the output list;

and several events:

- A new job enters the system;
- Job processing is started;
- Job processing is completed;
- A job leaves the system.

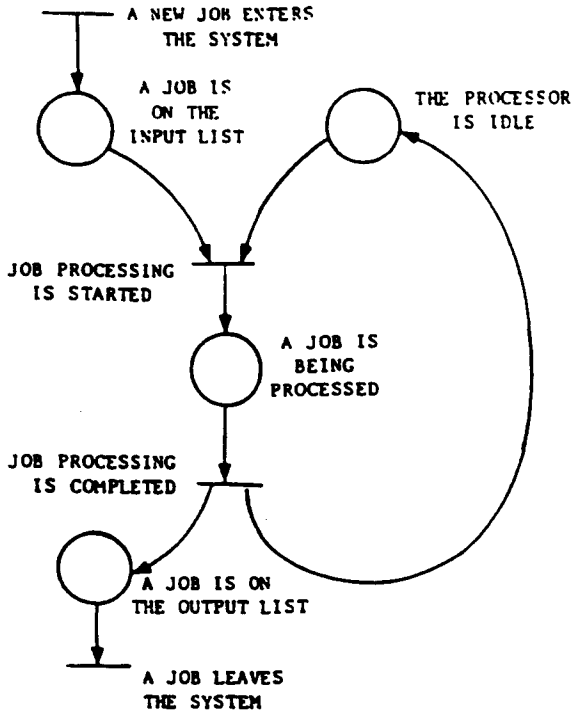



FIGURE 6. Modeling of a simple computer system.

The Petri net of Fig. 6 illustrates the modeling of this system. The "job enters" transition in this illustration is a *source*; the "job leaves" transition is a *sink*.

### Properties of Petri Nets Useful in Modeling

The example above illustrates several points about Petri nets and the systems they can model. One is inherent *concurrency* or *parallelism*. There are two main kinds of independent entities in the system: the job and the processor. In the Petri net model, the events which relate solely to one or the other can occur independently; there is no need to synchronize the actions of the jobs and the processor. Thus jobs may enter or leave the system at any time independent of the action of the processor. However, when synchronization is necessary, for instance when both a job and an idle processor must be available for processing to start, the situation is also easily modeled. Thus a Petri net would seem to be ideal for modeling systems of distributed control with multiple processes occurring concurrently.

Another major feature of Petri nets is their asynchronous nature. There is no inherent measure of time or the flow of time in a Petri net. This reflects a philosophy of time which states that the only important property of time, from a logical point of view, is in defining a partial ordering of the occurrence of events. Events take variable amounts of time in real life; the Petri net model reflects this variability by not depending upon a notion of time to control the sequence of events. Therefore, the Petri net structure itself must contain all necessary information to define the possible sequences of events of a modeled system.

Thus, in the net of Fig. 6 the event 'Job processing is completed' must follow the corresponding event 'Job processing is started' because of the structure of the net although no information at all is given or considered concerning the amount of time required to process a job. On the other hand, events which need not be constrained in terms of their relative order of occurrence are not constrained; thus while a job is being processed the event 'A new job enters the system' may occur, before, after, or simultaneously with the occurrence of the event 'Job processing is completed'.

A Petri net, like the system which it models, is viewed as a sequence of discrete events whose order of occurrence is one of possibly many allowed by the basic structure. This leads to a *nondeterminism* in Petri net execution. If at any time more than one transition is enabled, then any of the several enabled transitions may fire. The choice as to which transition fires is made in a nondeterministic manner, i.e., randomly or by forces that are not modeled. This feature of Petri nets reflects the fact that in real-life situations where several things are happening concurrently, the order of occurrence of events is not unique, so that any of a set of sequences may occur. While nondeterminism is advantageous from a modeling point of view, it introduces considerable complexity into the analysis of Petri nets.

To reduce this complexity, one limitation is generally accepted in the modeling of systems by Petri nets. The firing of a transition (occurrence of an event) is considered to be *instantaneous*, i.e., to take

zero time. Since time is a continuous variable, then, the probability of any two or more events happening simultaneously is zero, and two transitions cannot fire simultaneously. The events being modeled are considered *primitive* events. Note that this need cause no problems in the modeling of events. For example, in Fig. 6 the event 'Process a job' was modeled. But since this event is not a primitive one (it takes nonzero time and other events, such as the entering and leaving of the system by other jobs, may occur at the same time), it is decomposed into a beginning and an ending, which are instantaneous events, plus the noninstantaneous occurrence. This is shown in Fig. 7. Since this technique can be used for any nonprimitive event, the modeling power of Petri nets is not reduced.

The nondeterministic and nonsimultaneous firing of transitions in the modeling of concurrent systems takes two forms. One of these is shown in Fig. 8, which depicts "simultaneous" events that may occur in either order. In this situation the two enabled events do not affect each other in any way and the possible sequences of events include some in which one event occurs first and some in which the other occurs first.

The other type of situation, where simultaneity causes difficulties in modeling, is handled by defining events to occur nonsimultaneously. This is illustrated in Fig. 9. Here the two enabled transitions $t_j$ and $t_k$ are in *conflict*. Only one transition can fire, since in so doing it removes the token from $p_i$ and disables the other transition. To accurately model a system using Petri nets requires careful attention to assure that in cases such as the above the Petri net reflects all, and only those, event sequences which are possible in real life.

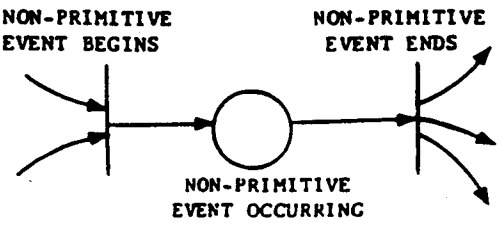



FIGURE 7. Modeling of a nonprimitive event.

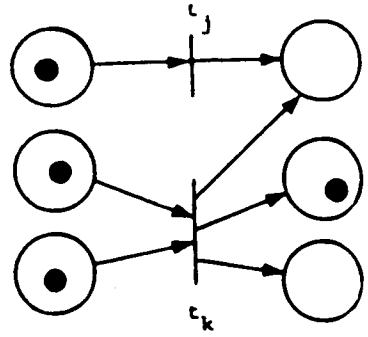



FIGURE 8. Modeling of "simultaneous" events which may occur in either order.

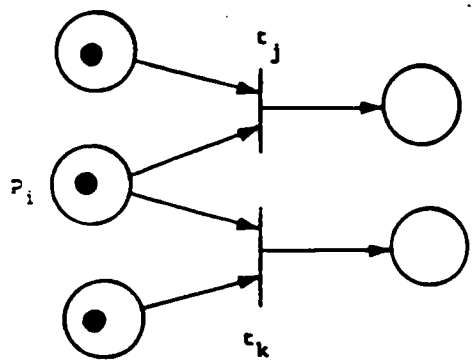



FIGURE 9. Illustration of conflicting transitions. Transitions $t_j$ and $t_k$ conflict since the firing of one will disable the other.

The two concepts illustrated by Figs. 8 and 9, *concurrency* and *conflict*, are basic to an understanding of Petri nets. At the same time, the usefulness of Petri nets as models of information flow derives from the natural way in which they can be used to express and analyze concurrency and conflict.

An important aspect of Petri nets is that they are uninterpreted models. The net of Fig. 6 has been labeled with statements that indicate to the human observer the intent of the model, but these labels do not, in any way, affect the execution of the net. The net of Fig. 10 is identical to that of Fig. 6 in that it has an identical structure. However, no meaning is attached to the places and transitions in this uninterpreted net; we deal only with the abstract properties inherent in the structure of the net. Since we are interested in the properties of Petri nets per se, in this paper we concern ourselves only with uninterpreted Petri nets.

Another valuable feature of Petri nets is their ability to model a system hierarchi-

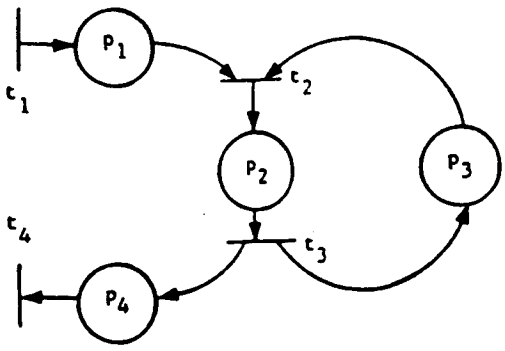



FIGURE 10. An uninterpreted Petri net.

cally. An entire net may be replaced by a single place or transition for modeling at a more abstract level (abstraction) or places and transitions may be replaced by subnets to provide more detailed modeling (refinement). Figure 11 illustrates this hierarchical modeling capability.

Most of the work on Petri nets has been in the investigation of the properties of a given net or class of nets. Little explicit attention has been paid to developing modeling techniques specifically for Petri nets. However, there are certain areas in which Petri nets would seem to be the perfect tool for modeling: those areas in which events occur asynchronously and independently. There are many examples, some of which we present here.

### Modeling of Hardware

Large, powerful computer systems often use asynchronous parallel activities in an effort to achieve maximum parallelism and hence increase effective processing speed. In computers such as the CDC 6600 [91] and the IBM 360/91 [4], for example, multiple functional units are provided to perform computations on multiple registers. The control unit of the machine attempts to keep several of these units in operation simultaneously.

However, the introduction of parallelism in this manner must be controlled so that the results of executing the program with and without parallelism are the same. Certain operations in the program will require that the results of previous operations have been successfully computed before the following instructions can proceed. A system which introduces parallelism into a sequential program in such a way as to maintain correct results is called *determinate*. The conditions for maintaining determinancy have been considered by Bernstein [10]. They are the following: For two operations $a$ and $b$ such that $a$ precedes $b$ in the linear precedence of the program, $b$ can be started before $a$ is done if and only if $b$ does not need the result of $a$ as an input and the results of $b$ do not change either the inputs or outputs of $a$ [15].

One method of applying these constraints to the construction of the computer control unit that is to issue instructions is to use a reservation table. An instruction for functional unit $u$ using registers $i, j$, and $k$ can be issued only if all four of these components are not reserved; if the instruction is issued, all four of them become reserved. If the instruction cannot be issued at this time, the control unit waits until the instruction can be issued before continuing to the next instruction.

This sort of scheme can be modeled as a Petri net. To each functional unit and each register we associate a place. If the unit or register is free, a token will be in the place; if it is not, no token will be in the place. Figure 12 shows a portion of a Petri net which could be used to model the execution of an instruction using unit $u$ and regis-

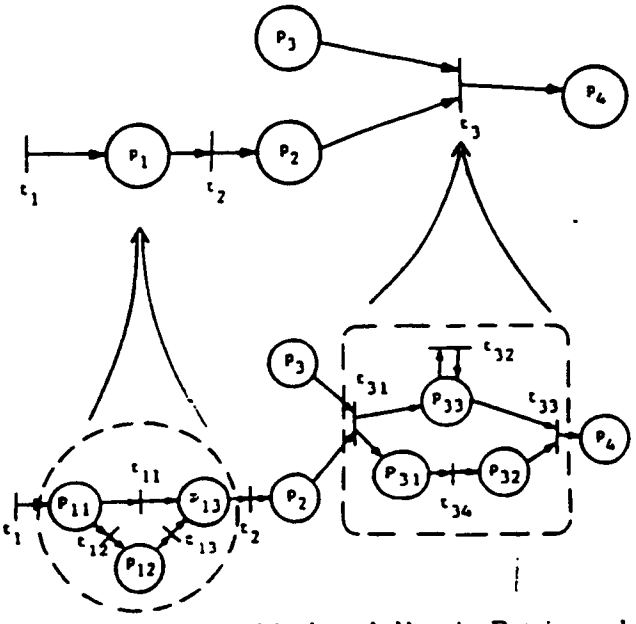



FIGURE 11. Hierarchical modeling in Petri nets by replacing places or transitions by subnets (or vice versa)

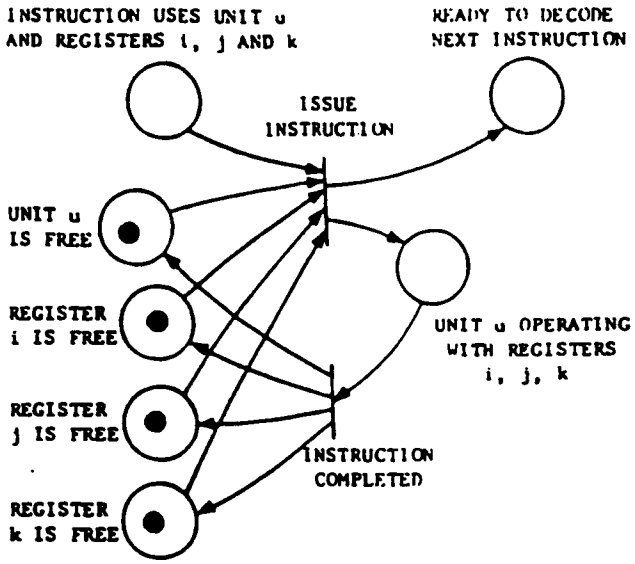



FIGURE 12. A portion of a Petri net modeling a control unit for a computer with multiple registers and multiple functional units.

ters $i, j$, and $k$. Modeling the entire control unit would of course require a much larger Petri net.

The scheme described above is a very simple method of introducing parallelism and does not consider, for example, the fact that multiple functional units can use the same register as an input simultaneously. Thus, this scheme may not produce schedules with maximum parallelism [55]. However, there are other schemes which can do so. These (more complicated) schemes can also be modeled by (more complicated) Petri nets. For example, the CPU of a CDC 6600 has been modeled by a Petri net [90]. This model was used to determine how object code should be generated to minimize execution time by maximizing parallelism between the various functional units.

Another approach to the construction of a high-performance computer is the use of pipelines [13]. This technique is useful, particularly for vector and array processing, and is similar to the operation of an assembly line. The pipeline is composed of a number of stages, which may be in execution simultaneously. When stage $k$ finishes, it passes on its results to stage $(k + 1)$ and looks to $(k - 1)$ for new work. If each stage takes $t$ time units and there are $n$ stages, then the complete operation for one operand takes $nt$ time units. However, if the pipe is kept supplied with new operands, it can turn out results at the rate of one every $t$ time units.

As an example, consider the addition of two floating-point numbers. The gross steps involved are:

a) Extract the exponents of the two numbers;
b) Compare the exponents, and interchange if necessary to properly order the larger and smaller of the exponents;
c) Shift the smaller fraction to equalize exponents;
d) Add fractions;
e) Post-normalize;
f) Consider exponent overflow or underflow, and pack the exponent and fraction of the result.

Each of these steps can be performed by a separate computational unit, with a particular operand being passed from unit to unit for the complete addition operation.

The coordination of the different units can be handled in several ways. Typically, the pipeline control is synchronous with the time allowed for each step of the pipe, being some fixed constant time $t$; every $t$ time units, the result of each unit is shifted down the pipe to become the input for the next unit. However, this can unnecessarily hold up processing, as the time needed may vary from stage to stage and may also vary for different inputs. For example, the post-normalization step in the floating-point addition above may take different amounts of time depending on how long the normalization shift should be and whether it should be to the left or to the right. Thus processing might be speeded up by an asynchronous pipeline in which results from stage $k$ are sent on to stage $(k + 1)$ as soon as stage $k$ is done and stage $(k + 1)$ is free. This scheme can be easily modeled by a Petri net.

Consider an arbitrary stage in the pipeline. Operations at this stage require certain inputs and produce certain outputs. Obviously, there has to be a place to put the inputs and outputs. Typically, this involves registers: the unit uses the values in its input register to produce values in its output register. It must then wait until 1) a new input is available in its input regis-

ter and 2) its output register has been emp-ed by being copied into the input register the next stage. Thus the control for the pipeline needs to know when the following conditions hold:

- input register full;
- input register empty;
- output register full;
- output register empty;
- unit busy;
- unit idle;
- copying taking place.

Figure 13 shows the Petri net which models the operation of an asynchronous pipeline of this kind. Other forms of pipeline control units can also be defined in terms of Petri nets.

The above examples show some of the uses to which Petri nets can be put in the modeling of hardware. Petri nets have also been associated with the description of general modular asynchronous systems [22, 75] and macromodules [14]. At Honeywell, Petri nets have been used for investigating the fault-tolerant properties of designs [48].

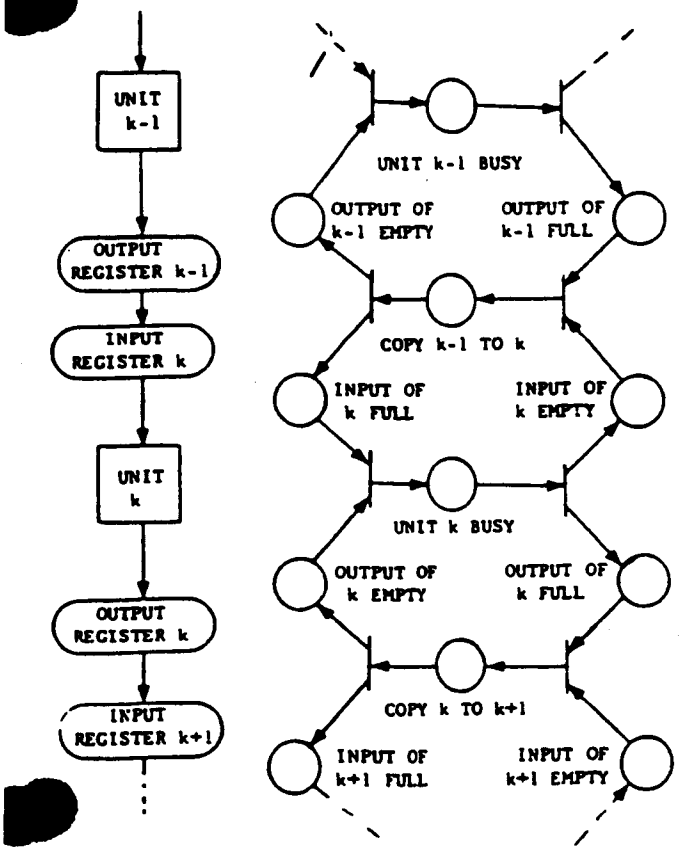
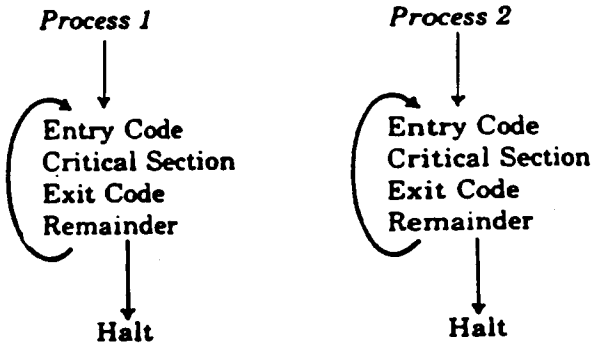



FIGURE 13. Representation of an asynchronous pipelined control unit. The block diagram on the left is modeled by the Petri net on the right.

### Modeling of Software

On a more abstract level, Petri nets can also model software concepts [24, 77, 61]. Resource allocation, deadlock, and process coordination in an operating system can be modeled. A process can be modeled by a Petri net in the same way that it can be modeled by a flowchart, and then the interactions between processes can be modeled as additional places, arcs and transitions.

For example, consider the mutual exclusion problem [25]. This is a problem of enforcing coordination of processes in such a way that particular sections of code called critical sections, one in each process, are mutually excluded in time. That is, if Process 1 is executing its critical section, then Process 2 may not begin *its* critical section until Process 1 has left its own critical section. At its highest level of abstraction, the situation is illustrated as follows:




The problem is to define appropriate entry code and exit code to assure mutual exclusion.

The mutual exclusion problem can be easily solved by using $P$ and $V$ operations as defined in [25] for process synchronization and coordination. The $P$ and $V$ operations operate on semaphores, and only $P$ and $V$ instructions may be executed on a semaphore. (A semaphore $S$ is a variable with integer values.) These operations can be defined as follows:

$P(S)$: As soon as $S > 0$, set $S := S - 1$;
$V(S)$: $S := S + 1$

These are indivisible operations. A process executing a $P$ operation must wait until the semaphore is positive before it can decrement it and continue. A $V$ operation simply adds one to the semaphore (perhaps allowing some other process to execute a $P$ operation). Two processes cannot execute $P$ or $V$ operations on the same semaphore concurrently. For example,

| *Process 1* | *Process 2* |
|---|---|
| P(mutex); | P(mutex); |
| "Critical Section"; | "Critical Section"; |
| V(mutex); | V(mutex); |

is a solution of the mutual exclusion problem diagrammed above using $P$ and $V$ operations, which are primitive, and the semaphore "mutex" which is global to the two processes and has an initial value of one.

$P$ and $V$ operations have been used widely. Systems of processes that use these operations can be modeled by Petri nets (see [74]). A semaphore is modeled by a place; the number of tokens in the place models the value of the semaphore. A $V$ operation on the semaphore places a token in the semaphore; a $P$ operation removes a token (it waits until there is one to remove, if necessary). This is illustrated in Fig. 14. The mutual exclusion problem can then be modeled as shown in Fig. 15, in which the place $S$ models the semaphore. Note that places $p_2$ and $p_4$ are mutually excluded.

The modeling of $P$ and $V$ operations by Petri nets has resulted in the discovery and proof of some limitations of these nets. Patil showed that certain process synchronization problems (such as the Cigarette Smokers Problem) cannot be solved by $P$ and $V$ operations [74]. Other work following up this technique of using Petri nets to prove properties of semaphore systems is reported in [57], [3], and [56].

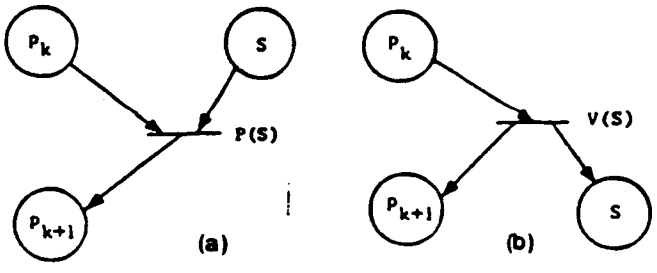



FIGURE 14. Examples of modeling with semaphores. (a) Modeling of a $P$ operation; (b) Model-

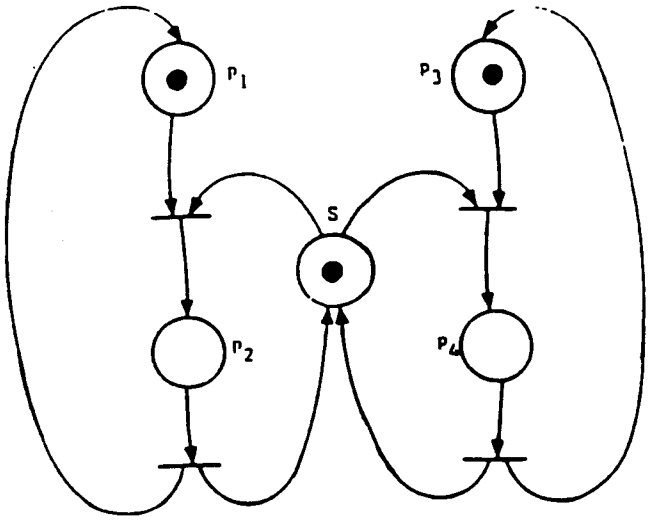



FIGURE 15. A Petri net model of a $P/V$ solution to the mutual exclusion problem.

These simple examples can only provide a slight indication of the modeling power of Petri nets and the many diverse systems that can be modeled by them. Many other areas of study have been mentioned as possible subjects of Petri net modeling, including resource allocation, operating systems [71], queueing networks, traffic control, distributed computer systems, legal systems [65], proofs in mathematics [30], and brain modeling. While much work in developing modeling techniques remains to be done, Petri nets are a very powerful modeling tool that can be applied to a large variety of systems.

## 3. STRUCTURE OF PETRI NETS

The use of Petri nets for the modeling of concurrent systems requires a careful understanding of the properties of such nets. The development of an appropriate theory has motivated most of the research on Petri nets. This basic theory is presented in the following sections. Since this paper is tutorial in nature, we have tried to limit the formality of the presentation; however, all of the concepts presented here have been rigorously defined and formalized in the literature. The reader who is interested in a more formal treatment should consult the references.

components: a set of *places*, $P$, and a set of *transitions*, $T$. To complete the definition, it is necessary to define the relationship between the places and the transitions. This can be done by specifying two functions connecting transitions to places: $I$, the *input* function, and $O$, the *output* function. The input function $I$ defines, for each transition $t_j$, the set of *input places* for the transition $I(t_j)$. The output function $O$ defines, for each transition $t_j$, the set of *output places* for the transition $O(t_j)$.

These four items define the *structure* of a Petri net. Places and transitions are the fundamental undefined concepts of Petri net theory; other concepts are defined in terms of these concepts. Formally, a Petri net $C$ is defined as the four-tuple $C = (P, T, I, O)$.

Consider the following example Petri net structure, defined as a four-tuple. Each component of the structure is given:

$$C = (P, T, I, O)$$

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1\} \qquad O(t_1) = \{p_2, p_3, p_5\}$$

$$I(t_2) = \{p_2, p_3, p_5\} \qquad O(t_2) = \{p_5\}$$

$$I(t_3) = \{p_3\} \qquad O(t_3) = \{p_4\}$$

$$I(t_4) = \{p_4\} \qquad O(t_4) = \{p_2, p_3\}$$

### The Petri Net Graph

Although the definition given above is useful and appropriate for formal work with Petri nets, it is ill-suited for the illustration of many of the concepts of Petri nets in an informal and intuitive, yet concise, manner. For this purpose a different representation of a Petri net is more useful: the *Petri net graph*. Figure 16 shows the Petri net graph corresponding to the structure described above.

A Petri net structure consists of places, transitions, and the input and output functions. In a Petri net graph there are two types of nodes corresponding to the places and transitions of the Petri net structure: a circle represents a place, and a bar represents a transition. The input and output functions are represented by directed arcs from the places to the transitions and from the transitions to the places. An arc is directed from a place $p_i$ to a transition $t_j$ if

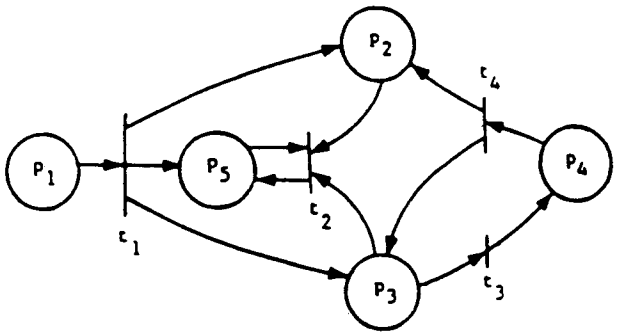



FIGURE 16. A Petri net graph.

Similarly, an arc is directed from a transition $t_j$ to a place $p_i$ if the place is an output of the transition.

A Petri net graph is a *directed* graph since the arcs are directed. In addition, since its nodes can be partitioned into two sets (places and transitions) such that each arc is directed from an element of one set (place or transition) to an element of the other set (transition or place), it is a *bipartite* directed graph.

The correspondence between Petri net graphs and Petri net structures is so natural that in most work they are considered not merely as different representations for the same concept, but rather as the same concept. Thus we refer to either Petri net graphs or Petri net structures as Petri nets. In this paper we give our examples as Petri net graphs, but our discussion and techniques are defined in terms of Petri net structures.

### Markings

A marking $\mu$ of a Petri net is an assignment of *tokens* to the places in that net. ("Token" is a primitive concept for Petri nets.) Tokens reside in the places of the net. The number and position of tokens in a net may change during its execution. The vector $\mu = (\mu_1, \mu_2, \cdots, \mu_n)$ gives, for each place in the Petri net, the number of tokens in that place. The number of tokens in place $p_i$ is $\mu_i$, $i = 1, \cdots, n$. We may also define a marking function $\mu: P \rightarrow N$ from the set of places to the natural numbers, $N = \{0, 1, 2, \cdots\}$. This allows us to use the notation $\mu(p_i)$ to specify the number of tokens in place $p_i$. For a marking $\mu$, $\mu(p_i) = \mu_i$.

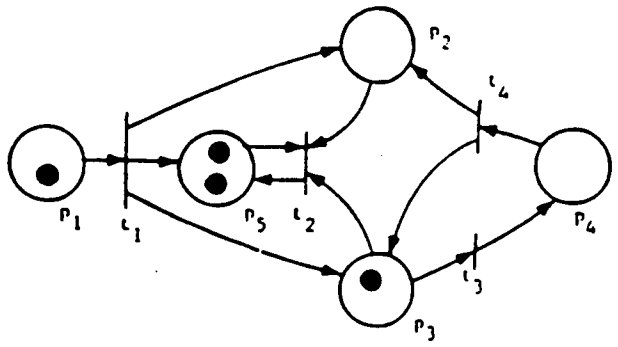



FIGURE 17. A marked Petri net.

sented by small solid dots inside the circles representing the places of the net. Figure 17 is an example of a Petri net graph with a marking. It represents the structure described in the previous section with the marking $\mu = (1, 0, 1, 0, 2)$. A Petri net $C = (P, T, I, O)$ with a marking $\mu$ becomes the *marked Petri net*, $M = (P, T, I, O, \mu)$.

Since the number of tokens in a place is unbounded over the set of all markings, there is an infinite number of markings for a Petri net. It is, of course, a denumerable infinity.

### Execution Rules for Marked Petri Nets

Having presented the definitions and representations of Petri nets and their markings, we now present the execution rules for marked Petri nets.

A Petri net executes by *firing* transitions. A transition may fire if it is *enabled*. A transition is enabled if each of its input places has at least one token in it. In Figure 17, for example, since the inputs to transition $t_2$ are places $p_2$, $p_3$, and $p_5$, transition $t_2$ is enabled if $p_2$ has at least one token, $p_3$ has at least one token, and $p_5$ has at least one token.

A transition fires by removing one token from each of its input places and then depositing one token into each of its output places. Transition $t_3$ in Fig. 17, with $I(t_3) = \{p_3\}$ and $O(t_3) = \{p_4\}$, is enabled whenever there is at least one token in place $p_3$. Transition $t_3$ fires by removing one token from $p_3$ (its input) and placing one token in $p_4$ (its output). Extra tokens in $p_3$ are not affected by firing $t_3$ although they may enable additional firings of $t_3$ later. Transition $t_2$ with $I(t_2) = \{p_2, p_3, p_5\}$ and $O(t_2) = \{p_5\}$ fires by removing one token from each of $p_2$, $p_3$, and $p_5$ and then puts one token in $p_5$.

Firing a transition will in general change the marking of the Petri net, $\mu$, to a new marking $\mu'$. Note that since only enabled transitions may fire, the number of tokens in each place always remains nonnegative when a transition is fired. Firing a transition can never remove tokens that are not there: if any one of the input places of a transition contains no tokens, then the transition cannot fire. Figure 18 summarizes the possible results of firing a transition. If a place is an input to the transition, one token is removed; if it is an output, one token is added. No net change occurs if the place is neither an input nor an output, or is both an input and an output. In the latter case, it is necessary for a token to be in the input place even though no change in marking occurs for this place.

### The State Space of a Petri Net

The *state* of a Petri net is defined by its marking. Thus the firing of a transition represents a change in the state of the net. The *state space* of a Petri net with $n$ places is the set of all markings, i.e., $N^n$. The

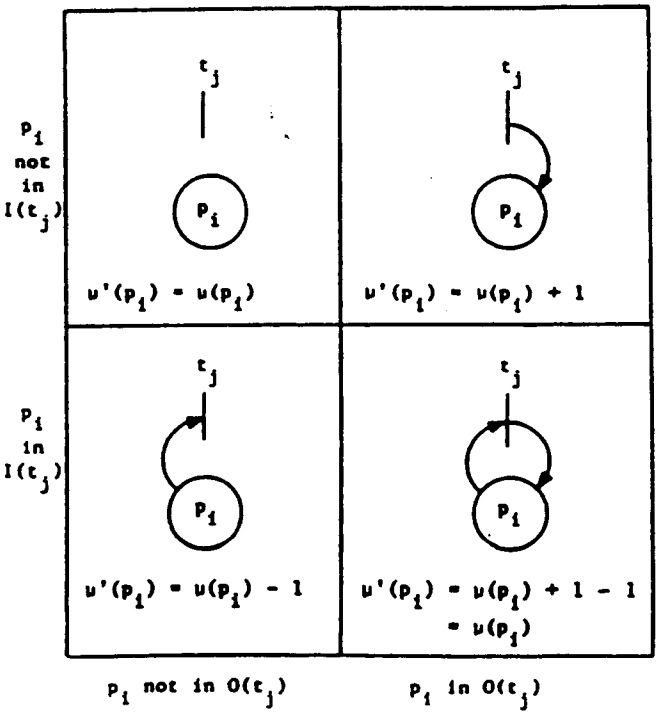



FIGURE 18. The changes in the marking of a place $p_i$ which result from firing a transition $t_j$.

change in state caused by firing a transition is defined by a partial function $\delta$, called the *next-state function*. Application of this function to a marking $\mu$ and a transition $t_j$ yields the value of the marking that results from the firing of transition $t_j$ in marking $\mu$. Since $t_j$ can fire only if it is enabled, $\delta(\mu, t_j)$ is undefined if $t_j$ is not enabled in marking $\mu$. If $t_j$ is enabled, then $\delta(\mu, t_j) = \mu'$, where $\mu'$ is the marking that results from removing tokens from the inputs of $t_j$ and adding tokens to the outputs of $t_j$.

Given a Petri net and an initial marking $\mu^0$, we can execute the Petri net by successive transition firings. Firing a transition $t_j$ in the initial marking produces a new marking $\mu^1 = \delta(\mu^0, t_j)$. In this new marking, we can fire any new enabled transition, say $t_k$, resulting in a new marking $\mu^2 = \delta(\mu^1, t_k)$. This can continue as long as there is at least one enabled transition in each marking. If we reach a marking in which no transition is enabled, then no transition can fire and the execution of the Petri net must stop.

As an example of Petri net execution, consider the execution of the marked Petri net of Fig. 17. In the marking $\mu^0 = (1, 0, 1, 0, 2)$, two transitions, $t_1$ and $t_3$, are enabled. Choosing one arbitrarily, we can fire $t_3$ producing the marking $\delta(\mu^0, t_3) = (1, 0, 0, 1, 2) = \mu^1$. In this marking, transitions $t_1$ and $t_4$ are enabled. Firing $t_4$ changes the marking to $\delta(\mu^1, t_4) = (1, 1, 1, 0, 2) = \mu^2$. In $\mu^2$, $t_1$, $t_2$, and $t_3$ are enabled. Firing $t_1$ results in $\mu^3 = \delta(\mu^2, t_1) = (0, 2, 2, 0, 3)$. This process can continue as long as at least one transition is enabled.

Two sequences result from the execution of the Petri net: a sequence of markings $(\mu^0, \mu^1, \mu^2, \cdots)$, and a sequence of transitions $(t_{j0}, t_{j1}, t_{j2}, \cdots)$ such that $\delta(\mu^k, t_{jk}) = \mu^{k+1}$ for $k = 0, 1, 2, \cdots$. In the example above the transition sequence was $t_3, t_4, t_1, \cdots$; therefore $j_0 = 3$, $j_1 = 4$, $j_2 = 1$, $\cdots$. Given the transition sequence and $\mu^0$, we can easily derive the marking sequence for the execution of the Petri net and, except for a few degenerate nets, given the marking sequence we can derive the transition sequence. Both of these sequences thus provide a record of the execution of the net.

### The Reachability Set of a Petri Net

From a marking $\mu$, a set of transition firings is possible. The result of firing a transition in a marking $\mu$ is a new marking $\mu'$. We say that $\mu'$ is *immediately reachable* from $\mu$ if we can fire some enabled transition in the marking $\mu$ resulting in the marking $\mu'$. A marking $\mu'$ is *reachable* from $\mu$ if it is immediately reachable from $\mu$ or is reachable from any marking which is immediately reachable from $\mu$. We then define the reachability set $R(M)$ for a marked Petri net $M = (P, T, I, O, \mu)$ as the set of all markings which can be reached from $\mu$. This is the reflexive transitive closure of the "immediately reachable" relationship.

The reachability set of a marked Petri net is the set of all states into which the Petri net can enter by any possible execution. Hence many analysis questions deal with properties of the reachability set of a Petri net. (This is discussed in more detail in Section 4.)

Considering a Petri net in terms of states and state changes may obscure some of the more important concepts relating to concurrent systems that can be modeled by Petri nets. One of these is the concept of *local* changes in state, as modeled by transitions. In a complex system composed of independent asynchronously operating subparts, each part can be modeled by a Petri net. The enabling and firing of transitions are then affected by, and in turn affect only, local changes in the marking of the Petri net. Separate parts of the total system may operate independently and concurrently. The view of Petri nets presented here, with a *global* state and a global sequence of transitions, can hide the inherent modularity and concurrency in the Petri net model. However, despite some important objections to this automata-theory-related conception [44], most research in the United States has been based on this approach.

## 4. ANALYSIS OF PETRI NETS

Why should systems be modeled as Petri nets? Originally, the purpose was mainly *descriptive*. Petri nets with their uniform

and simple execution rules, can be used to describe a system in terms of simple concepts which provide a natural way to depict systems of asynchronous concurrent processes [24]. After a short time, however, it became obvious that another use of Petri nets was to take the description of the system, as a Petri net, and *analyze* it for the presence of desirable or undesirable properties. A body of work is being developed which is aimed at deriving, from a Petri net, properties of the net, and from these, the properties of the system which the net models. Some of the analytic questions that one would like to ask about a Petri net are quite difficult; hence, restricted subclasses of Petri nets have been defined to make analysis easier in specific situations. (This will be discussed in Section 6.)

Following this train of thought, another use of Petri nets would seem to be in the *design* of concurrent systems. One method of design would consist in first creating a design in a traditional representation, for example, a logic circuit or a program. Then the design would be converted into a Petri net and the Petri net analyzed. If no design errors were discovered, the design could then be implemented in the traditional manner. If there were errors, however, it would be necessary to determine how the error which was found in the Petri net representation manifests itself in the original design, modify the design, and repeat the entire process of conversion to a Petri net and analysis. This process could be simplified if the design process could be carried out directly in Petri nets and the resultant Petri net implemented directly. This approach requires that both the necessary design techniques and the methods for implementing Petri net designs, in hardware or software, be developed.

Although some work on design with Petri nets [67] and implementation of Petri nets [73, 27, 75] has been done, it has been limited in scope, presumably because its success hinges on the existence of effective analysis techniques.

### Analysis Questions

The first task in developing analysis techniques is to define the types of questions that the analysis procedures are to answer, and the properties to be studied. Obviously, the analysis techniques should be oriented towards the solution of those problems that most need to be solved rather than towards areas that are only of academic interest. We discuss now some of the properties that have been investigated for Petri nets.

One property of Petri nets derives from their original definition in terms of events and conditions. A condition is represented by a place. The fact that the condition holds is indicated by a token in the place. Consider, however, that either a condition holds or it does not hold. Hence, a token should either be present or it should be absent. Also, no more than one token should ever be present in one place at one time, as it seems pointless to have multiple tokens when one is sufficient. Petri nets which are constructed such that no more than one token can ever be in any place of the net at the same time are *safe* nets [44].

Another definition of a safe net is the statement that there is a bound on the number of tokens in any place of the net, and that bound is 1. A natural generalization of this is to allow multiple tokens in a place but only to the extent that there are no more than $k$ tokens in any given place at the same time. Nets in which the number of tokens in any place is bounded by $k$ are called *$k$-bounded* nets. (Thus a safe net is a 1-bounded net.) If a net is $k$-bounded for some $k$ but we do not know the value of $k$, it is simply called *bounded*.

Boundedness is a very important practical property of Petri nets. If we wish to implement a design modeled by a Petri net, then since the capacity of any given hardware is bounded, the Petri net must be bounded if construction is to be possible. In other words, if, for example, places are to be implemented as counters, then since every physically realizable counter can only hold a bounded number, the net must also be bounded.

Another property that might be important is conservation of tokens. If tokens are used to represent resources, then it follows that since resources are neither created nor destroyed, tokens should also

be neither created nor destroyed. A Petri net is *conservative* if the number of tokens in the net is conserved. This implies that each transition in a conservative net is conservative, in the sense that the number of inputs of each firable transition is equal to the number of outputs of that transition. More generally, weights can be defined for each place allowing the number of tokens to change as long as the weighted sum is constant [62].

Notice that the above statement was qualified to restrict it to firable transitions. Consider the Petri net of Fig. 19, which depicts a nonfirable transition. Transitions $t_1$ and $t_2$ are conservative. Transition $t_3$ is not conservative, in the sense that if it ever fired it would decrease the number of tokens by two. However, for any initial marking in which the number of tokens in $p_3$ is zero, this transition is not firable and hence the number of tokens in the net is conserved. This means, then, that the transition $t_3$ and place $p_3$ can be deleted from the net, simplifying it, with no change in its behavior. This would allow a simpler and cheaper implementation.

The notion of transitions that cannot fire seems strange, and we want to be able to identify such transitions. Note that a transition which cannot fire is not simply a transition which is not enabled, but rather a transition which cannot become enabled. A transition is *dead* in a marking if there is no sequence of transition firings that can enable it. A transition is *potentially firable* if there exists some sequence that enables it [35]. A transition of a Petri net is *live* if it is potentially firable in all reachable markings.

The importance of the concepts of liveness and deadness of transitions comes from considerations in the modeling of operating systems. Liveness is tied to the concept of deadlocks and deadlock-freeness [45]. Thus it may be important not only that a transition be firable in a given marking, but that it stay potentially firable in all markings reachable from that marking. If this is not true, then it is possible to reach a state in which the transition is dead, perhaps signifying a possible deadlock.

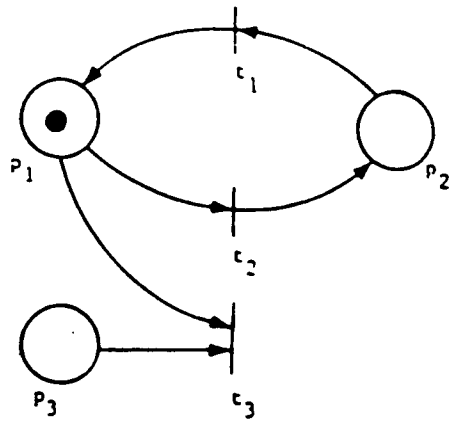



FIGURE 19. A Petri net with a nonfirable transition. Transition $t_3$ is dead in this marking.

A number of different definitions of liveness have been considered. Commoner [16] defines four subtly different forms of liveness for a transition $t_j$ and a marking $\mu$:

$L_1$: If there exists a $\mu'$ in $R(M)$ such that $\delta(\mu', t_j)$ is defined (i.e., $t_j$ is potentially firable);

$L_2$: If for every positive integer $n$ there exists a transition sequence $\sigma$ such that $\delta(\mu, \sigma)$ is defined and $t_j$ appears at least $n$ times in $\sigma$;

$L_3$: If there exists an infinite sequence of transition firings such that $\delta(\mu, \sigma)$ is defined and $t_j$ appears infinitely often in $\sigma$;

$L_4$: If $t_j$ is live (i.e., potentially firable in all reachable markings).

Note that the implications of the four definitions of liveness are quite different (see [53]). Thus, whenever the property of liveness in a Petri net is discussed, it is important to state carefully the definition being used, since we do not yet have a single commonly accepted definition.

As mentioned earlier, the concept of liveness was developed to deal with deadlock problems in operating systems. Other problems in operating systems can also be posed in terms of Petri nets. The actual statement of these questions depends upon the manner in which the system is modeled. For example, access to a resource may be modeled by a transition or a place. The mutual exclusion problem is to assure that at most one of perhaps several processes tries to access the resource at the same time. Depending on the modeling used, this will be expressed as a question concerning whether or not two transitions

can be enabled simultaneously or whether or not two places may have tokens simultaneously.

Notice, however, that both of these questions can be stated in terms of the reachability of any of a set of undesirable states. In fact, many questions can often be reduced to the *reachability problem*. The reachability problem is simply the following: Given a marked Petri net (with marking $\mu$) and a marking $\mu'$, is $\mu'$ reachable from $\mu$? This problem is very important to the analysis of Petri nets. It can be considered a special case of the *set reachability problem*, which is to determine if a set of markings, $S = \{\mu^1, \mu^2, \ldots \mu^k\}$, is a subset of the reachability set $R(M)$ of a marked Petri net.

There are many other interesting questions that might be studied with Petri nets. Furthermore, since the questions designers want to ask about their designs depend on the projected use of the designs, there will always be new questions. Thus it is important to develop general techniques that allow new types of questions to be answered. The basis for the importance of the reachability problem is that many questions about the correctness of systems modeled as Petri nets can be translated into instances of this problem. For instance, Hack has shown that the liveness problem (are all transitions live?) is reducible to the reachability problem and that in fact the two problems are equivalent, since reachability is also reducible to liveness [35].

### Solution Techniques

While several approaches to the analysis of Petri nets have been considered, almost all work in this area eventually uses one basic technique. This technique involves finding a finite representation for the reachability set of a Petri net, in recognition of the fact that many of the properties of a Petri net are based on properties of its reachability set. The representation used is known as the *reachability tree*. It consists of a tree whose nodes represent markings of the Petri net and whose arcs represent the possible changes in state resulting from the firing of transitions [51, 53].

Notice, however, that the reachability set of a marked Petri net is often infinite. Thus, to form a finite representation of an infinite set we must map many markings into the same node of the tree. This many-to-one mapping is accomplished by collapsing a set of states into a node by ignoring the number of tokens in a place of the net when this number becomes "too large." This is represented by using a special symbol, $\omega$, for the number of tokens in this place.

The symbol $\omega$ represents a value which can be arbitrarily large. Because of this we must interpret the operations of addition, subtraction, and comparison as

$$\begin{aligned} \omega + a &= \omega \\ \omega - a &= \omega \\ a &< \omega \end{aligned}$$

for any natural number $a$. Thus, $\omega$ might be thought of as a symbol for infinity.

Each node in the reachability tree is labeled with a marking; arcs are labeled with transitions. The initial node (root of the reachability tree) is labeled with the initial marking. Given a node $x$ in the tree, additional nodes are added to the tree for all markings that are directly reachable from the marking of the node $x$. For each transition $t_j$ which is enabled in the marking for node $x$, a new node with marking $\delta(x, t_j)$ is created, and an arc labeled $t_j$ is directed from the node $x$ to this new node. This process is repeated for all new nodes.

Continuing this process will obviously create the entire state-space. A path from the initial marking (root) to a node in the tree corresponds to an execution sequence. Since the state-space may be infinite, two special steps are taken to define a finite reachability tree. First, if a new marking is generated which is equal to an existing marking on the path from the root node to the new marking, the new (duplicate) marking becomes a terminal node. Since the new marking is equal to the previous marking, all markings reachable from it have already been added to the reachability tree by the earlier identical marking.

Second, if any new marking $x$ is generated which is greater than a marking $y$ on

the path from the root node to the marking $x$, then those components of marking $x$ which are strictly greater than the corresponding components of marking $y$ are replaced by the symbol $\omega$. Since marking $x$ is greater than marking $y$, any sequence of transition firings which is possible from marking $y$ is also possible from marking $x$. In particular, the sequence that transformed marking $y$ into marking $x$ can be repeated indefinitely, each time increasing the number of tokens in those places which have a $\omega$. Thus the number of tokens in these places can be made arbitrarily large.

As an example of this construction, consider the marked Petri net of Fig. 20. We begin with (1, 0, 1, 0) as the root of the tree. In this marking, we have only one enabled transition. Thus we have a new node corresponding to firing $t_3$, (1, 0, 0, 1) and an arc from (1, 0, 1, 0) to (1, 0, 0, 1). From this marking we can fire $t_2$, resulting in (1, 1, 1, 0). Now, since (1, 1, 1, 0) $\geq$ (1, 0, 1, 0), we replace the second component by $\omega$. This reflects the fact that we can fire the sequence $t_3t_2$ an arbitrary number of times and make the number of tokens in place $p_2$ as large as desired. In the marking (1, $\omega$, 1, 0), two transitions are enabled, $t_1$ and $t_3$. Firing these two would give us two new nodes, (1, $\omega$, 0, 0) and (1, $\omega$, 0, 1). The first of these has no successors since $\delta((1, \omega, 0, 0), t_j)$ is undefined for all $t_j$. The second enables $t_2$, which fires to give (1, $\omega$, 1, 0) which is identical to an earlier node. Thus, the complete reachability tree is as shown in Fig. 21.

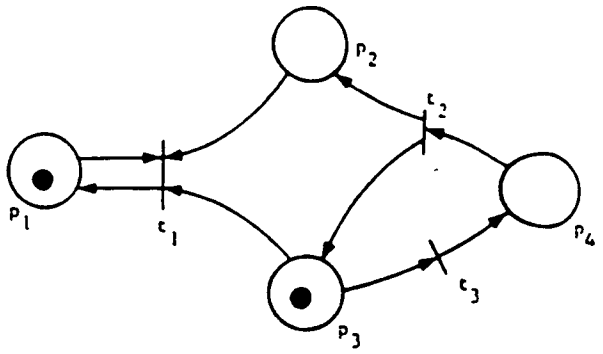



FIGURE 20. A Petri net with marking (1,0,1,0) and infinite reachable state-space.

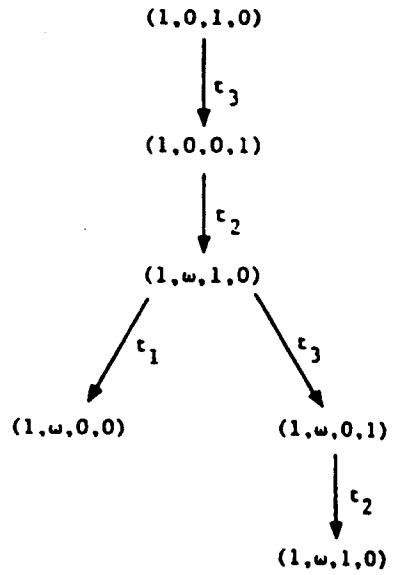



FIGURE 21. The reachability tree of the Petri net of Fig. 19.

### Analysis Using the Reachability Tree

How is the reachability tree used for analysis? Let us consider some of the questions raised in the previous section.

On the questions of safeness and boundedness, notice that if a Petri net is $k$-bounded, then, by definition, no more than $k$ tokens are ever in any place. Thus the possible values for each place are drawn from the set $\{0, 1, \ldots, k\}$ and there are only $(k + 1)^n$ possible reachable markings. Therefore, the reachable state-space is finite.

In the same way, consider a conservative Petri net. If we let $k$ be the number of tokens in the net, then we must always have $k$ tokens in the net. Since there are only a finite number of ways to partition $k$ tokens among $n$ places, we must again have a finite reachability set.

Now consider the reachability tree. If any node in the reachability tree contains the symbol $\omega$, then that component can become arbitrarily large, i.e., it is not bounded. Thus, if the symbol $\omega$ is anywhere in the reachability tree, the reachability set is not finite and hence neither bounded nor conservative. On the other hand, if the $\omega$ symbol does not occur anywhere in the tree, then the reachability tree *is* the reachability set and both are finite. This means that the reachability set is bounded, and the bound can be established by inspection. Similarly, if the

reachability set is finite, conservation can be determined by inspection. In fact, for a finite reachability set, any analysis question can be solved by inspection.

Other problems can also be solved using the reachability tree. For example, the *coverability problem* can be solved by inspection of the reachability tree [51, 35]. The coverability problem is the following: Given a marked Petri net $M$ and a marking $\mu$, does there exist a marking $\mu'$ in $R(M)$ such that $\mu' \geq \mu$? This problem is useful in determining whether violations of mutual exclusion occur in a system, and in testing transitions for liveness (deadlock).

### The Reachability Problem

The more general questions of liveness and reachability are not answerable by the reachability tree. Because of the fundamental nature of the reachability problem in the analysis of Petri nets and vector addition systems—an equivalent modeling system [51, 34]—it has been the object of a considerable amount of research. It has been shown that the general reachability problem is equivalent to several special cases such as the zero reachability problem (is the zero vector an element of the reachability set?) [70] and the subset reachability problem (given a nonempty subset of places and a marking $\mu$, is any reachable marking equal to $\mu$ for the specified subset of places, with all other places allowed to have any value?) [34]. These problems are equivalent in that if an algorithm can be found to solve any one of them it can be modified to solve any of the others.

Such an algorithm has recently been found [88]. The algorithm is very difficult to follow and depends upon both a search through a finite tree of possible solutions and the recursive solution of reachability problems for lower-dimensional state-spaces. However, regardless of the complexity of the algorithm, its existence shows that the reachability problem is solvable (although possibly at a high cost).

Since the liveness problem is equivalent to the reachability problem [35], it also is solvable.

### Unsolvable Problems

Some Petri-net problems are not solvable despite their apparent similarity to the reachability problem. The first such problem studied was the subset problem: given two marked Petri nets, is the reachability set of one net a subset of the reachability set of the other net? Rabin showed this problem to be undecidable [9, 33]. Later Hack showed that the equality problem—given two marked Petri nets, is the reachability set of one net equal to the reachability set of the other net?—is also undecidable [36].

These problems are important for applications in which one might want the Petri nets to be optimized, but the set of reachable markings not to be changed. Unfortunately, it has been shown that both of these problems are *undecidable* in the sense that there exists no general algorithm which can decide, for two arbitrary marked Petri nets, if their reachability sets are equal or one is the subset of the other. This proof is quite complicated. It is based on the construction of a Petri net which (weakly) computes the value of a polynomial in such a way that if the equality problem is decidable then Hilbert's tenth problem is solvable. Since Hilbert's tenth problem is known to be unsolvable, the equality problem is undecidable. This in turn implies the undecidability of the subset problem.

### Complexity

While much attention has been focused on the decidability of the reachability problem and similar problems, other aspects of Petri net analysis procedures have also been investigated. One aspect that has recently come under investigation is the computational complexity of the problem. While it is not yet possible to determine the complexity exactly, it is possible to give its lower bounds. Lipton has shown that the reachability problem is exponential time-hard and exponential space-hard [64]. That is, the amount of time and memory space needed to solve the reachability problem must be at least an exponential function of the length of the input descrip-

tion of the Petri net (in the worst case). This is a lower bound; the actual complexity could be much worse. Lipton also shows that the coverability problem has an exponential space lower bound. Rackoff [83] has obtained an algorithm for solving coverability in exponential space, showing this to be a tight lower bound. The complexity of some other problems in Petri nets has also been considered [49].

These complexity analyses are very important in determining the usefulness of Petri nets for the modeling and analysis of systems. The recent discovery that the reachability problem is decidable marked a significant advance in the search for analysis techniques; however, the complexity bounds as well as the large number of places and transitions needed to model even simple systems tend to indicate that, although analysis questions may be decidable using Petri nets, in the worse case the cost of answering even simple questions may make such analysis unfeasible.

## 5. PETRI NET LANGUAGES

Another area in which Petri nets have been used is the study of formal languages (see [79, 37]). Here Petri nets are used to model the flow of information and control of actions in a system. The firing of a transition models the occurrence of an operation in the modeled system. A Petri net properly models a system if every sequence of actions in the modeled system is possible in the Petri net and every sequence of actions in the Petri net represents a possible sequence in the modeled system.

To represent these concepts, we label the transitions of a Petri net; with each transition we associate a symbol naming the transition. Since there are only a finite number of transitions, we can define a finite alphabet $\Sigma$ which is the set of all these symbols. A labeling function $\sigma$ maps transitions to symbols, i.e., $\sigma: T \rightarrow \Sigma$. A labeled marked Petri net defines a set of strings over $\Sigma$, each string corresponding to a possible execution of the net. The set of all possible strings corresponding to the possible executions of a marked labeled Petri net defines a *Petri net language*.

Several varieties of Petri net languages result from slightly different approaches to defining the languages of a Petri net. One entire group of languages results from the use of different labeling policies, since restrictions on the allowable labeling functions create restricted classes of Petri net languages.

The *free* languages are those obtained when one introduces the requirement that all transition labels be distinct and nonnull, i.e., $\sigma(t_i) \neq \sigma(t_j)$ for $t_i \neq t_j$. This requirement reflects the view that since distinct transitions model distinct events, they should be distinctly labeled.

A more general class of languages results if one allows a more general label function in which many transitions may be labeled with the same symbol. This reflects the view that the same action can result from different circumstances, and hence may be modeled by different transitions. The modeling process may even introduce some "extra" transitions that are necessary for proper token movement but do not correspond to actions of interest in the modeled (real) system.

A third class of labeling functions allows transitions to be labeled with the *null* label $\lambda$. A null label is defined as a label which does not show up in the string resulting from an execution of the Petri net.

As an example of the differences of these labeling policies, consider the Petri net of Fig. 22. Let the language under consideration be the set of sequences whose net result is to move the token in place $p_1$ to place $p_4$ (i.e., the set of sequences $\{t \in T^* \mid \delta((1, 0, 0, 0), t) = (0, 0, 0, 1)\}$). If we label the transitions with the free labeling

$$\sigma_1(t_1) = a \qquad \sigma_1(t_3) = b$$

$$\sigma_1(t_2) = c \qquad \sigma_1(t_4) = d,$$

then the language is $\{a^n c b^n d \mid n \geq 0\}$. A nonfree but $\lambda$-free labeling such as

$$\sigma_2(t_1) = a \qquad \sigma_2(t_3) = b$$

$$\sigma_2(t_2) = a \qquad \sigma_2(t_4) = b$$

yields the language $\{a^n b^n \mid n > 0\}$. If transition $t_1$ is assigned a null label, then the language which results could be the regular language $\{ab^*c\}$.

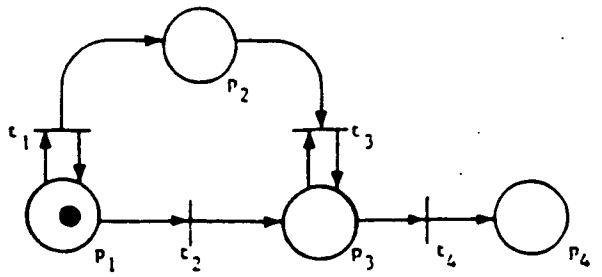



FIGURE 22. A Petri net with marking (1,0,0,0).

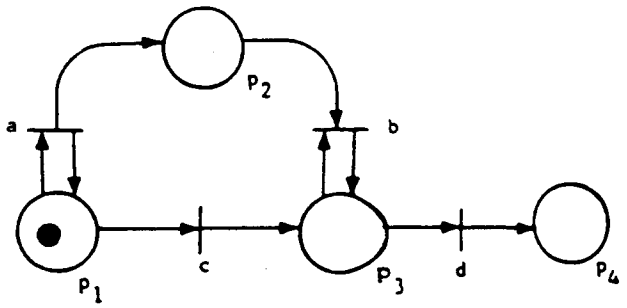



FIGURE 23. Labeled marked Petri net corresponding to the net of Fig. 22.

The class of labeling functions is only one of the determinates of Petri net languages. Another is the definition of the set of final states. Remember that a language is the set of all possible sequences resulting from the execution of a labeled Petri net starting in an initial marking (or one of a finite set of initial markings) and terminating in any element of a set of *final markings*. Different classes of languages correspond to different definitions of the set of final markings.

Four types of Petri net languages have been defined in terms of the definition of final markings [39]:

*L-type*: The set of final markings is defined by a finite final marking set $F$;

*G-type*: Given a finite marking set $F$, a final marking is any marking which is greater than or equal to any element of $F$;

*T-type*: A final marking is any terminal marking (a marking in which no transition is enabled);

*P-type*: All reachable markings are final markings.

As an example of the differences between these different language classes, consider the Petri net of Fig. 23, which is a labeled version of the net of Fig. 22. (The labeling shown is free, but that is not important.) For a final state set $F = \{(0, 0, 1, 0)\}$, the L-type language is $\{a^n cb^n \mid n \geq 0\}$, the G-type language is $\{a^m cb^n \mid m \geq n \geq 0\}$, the T-type language is $\{a^m cb^n d \mid m \geq n \geq 0\}$, and the P-type language is $\{a^m \mid m \geq 0\} \cup \{a^m cb^n \mid m \geq n \geq 0\} \cup \{a^m cb^n d \mid m \geq n \geq 0\}$.

With three different kinds of labeling functions and four different kinds of final-state sets, twelve different classes of Petri net languages can be defined. Despite their differences, these classes are closely related. Preliminary investigations have shown that a number of containment relationships hold between the classes. For example, since the labeling functions are successively more general, all Petri net languages with free labelings are also Petri net languages with λ-free labelings which in turn are also Petri net languages with arbitrary labelings. It can be shown that all P-type languages are also G-type languages, that all G-type and T-type languages with arbitrary or λ-free labelings are L-type languages with the same type of labeling, and that L-type languages with arbitrary labelings are also T-type languages. These relationships are shown in Fig. 24, in which an arc between classes is used to indicate that one class is contained within the other. It is not known whether other arcs might also exist or which containments are proper.

The L-type and P-type languages have been investigated in greater depth. L-type languages have been shown to be closed under union, intersection, concatenation, concurrency, reversal and λ-free homo-

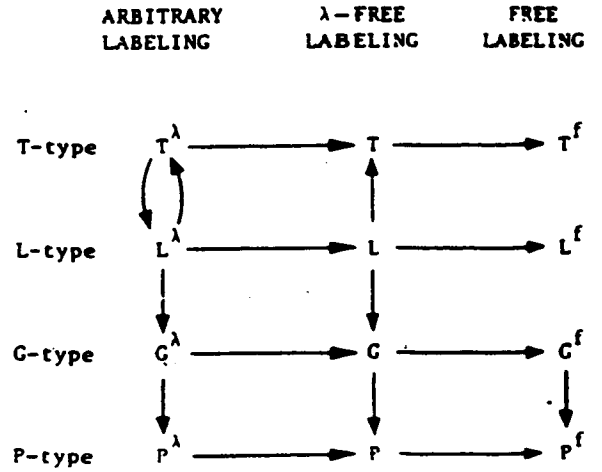



FIGURE 24. Relationships among the different classes of Petri net languages.

morphism [77, 37]. P-type languages are more restrictive but are still closed under union, intersection, concatenation and concurrency [37]. Hack has developed a characterization theorem for L-type languages showing that the class of L-type languages is the smallest of languages that includes a finite language and the complete parenthesis language and is closed under inverse homomorphism, concurrency, intersection and λ-free homomorphism. (The complete parenthesis language is the context-free language over two symbols {(,)} whose strings are properly nested parenthesis strings [37].)

The relation of Petri net languages (of the L-type) and other classes of languages has also been examined. All regular languages are Petri net languages. Some context-free languages are Petri net languages and some Petri net languages are context-free, but neither class includes the other. Their common intersection includes regular languages and bounded context-free languages, among others. Surprisingly, the complement of a free Petri net language is context-free [18]. All λ-free Petri net languages are context-sensitive [77]. These relationships are illustrated in Fig. 25. In this diagram, an arrow between two classes of languages indicates proper containment. Note that Petri net languages appear to be roughly equivalent to context-free languages in complexity (and interest).

The original impetus for studying Petri net languages was to try to settle some of the decidability questions for Petri nets. It has been shown [37] that the membership problem for Petri nets with λ-free or free labelings is decidable, but the inclusion and equivalence problems for P, $P^{\lambda}$, L and $L^{\lambda}$ languages are undecidable. Many decidability problems are equivalent to the reachability problem and thus decidable.

A different approach to the study of Petri nets by the use of formal language theory has been considered by Crespi-Reghizzi and Mandrioli [19]. They noticed the similarity between the firing of a transition and the application of a production in a derivation in which places are nonterminals and tokens are separate instances of the nonterminals. The major difference of this approach is the lack of ordering information in the Petri net contained in the sentential form of the derivation. To accommodate it, Crespi-Reghizzi and Mandrioli defined the *commutative grammars*, which are isomorphic to (generalized) Petri nets. In addition, they considered the relationship of Petri nets to matrix, scattered-context, non-terminal-bounded, derivation-bounded, equal-matrix, and Szilard languages. For example, it is not difficult to see that the class $L^{I}$ is the set of Szilard languages of matrix context-free languages [21]. Similar work by Keller considered the class of commutative semi-Thue systems [53]. Keller has also pointed out that λ-free languages are a subset of real-time counter languages [26].

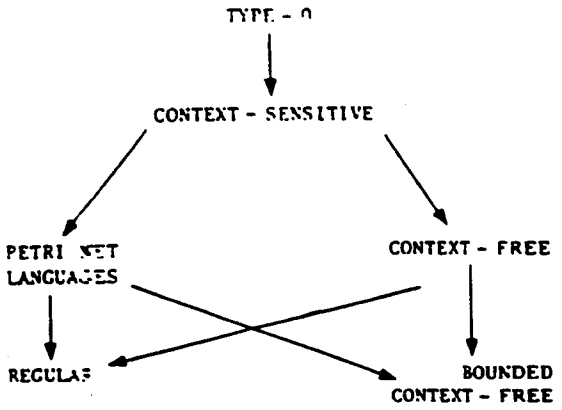



FIGURE 25. Relationships among Petri net languages and the classical language classes.

It should be pointed out that this entire approach to Petri nets and languages may represent an approach from the wrong direction: Petri nets were designed to represent concurrent activity, yet the representation of a Petri net execution by a string forces all activity to be represented serially, incorrectly implying a total ordering between events. Some work has considered other representations of the partial orderings resulting from concurrent activities [43, 85], but further research is needed in this area.

## 6. EXTENSIONS AND SUBCLASSES

The success of any model is due to two factors: its modeling power and its decision power. Modeling power refers to the abil-

ity to correctly represent the system to be modeled so that the model will be a faithful representation of the modeled system. Decision power refers to the ability to analyze specific versions of the model and determine properties of the modeled system [54].

These two factors generally work at cross purposes. Consider, for example, finite-state systems. Since the set of reachable states is finite, it is possible to answer almost any question about a finite-state model; hence such a model has very high decision power. On the other hand, the class of systems which can be modeled is severely limited, which means that such a model has very low modeling power. Turing machines, by contrast, have good modeling power but, since most general questions are undecidable, have poor decision power. When we increase modeling power (and hence the complexity of the models and the modeled systems), our ability to algorithmically determine the properties of the models is generally decreased.

Petri net models represent an attempt to compromise between these two factors. They have better modeling power than finite-state models while (one hopes) retaining most of the latter's decision power. As a matter of fact, Petri nets were originally defined in answer to the limited modeling power of finite-state models.

Not all researchers have been satisfied with the modeling power of Petri nets, however. It is difficult to model some events or conditions in systems by Petri nets, and it has been shown that the correct modeling of other relatively reasonable systems is impossible [3, 57]. Thus several proposals have been put forth for extending the modeling power of Petri nets.

### Extended Petri Nets

One of the first extensions is to remove the constraint that a place may contribute or receive only one token from the firing of a transition. Consider the modeling of chemical reactions. Here a token in a place represents the availability of a certain molecule or atom. Chemical reactions are modeled by transitions and may occur whenever tokens indicate the availability of the reactants. The firing of the transition models the reaction, which consumes inputs (reactants) and produces outputs (products). Notice that a chemical reaction may well require more than one unit of a particular reactant. This is modeled by allowing multiple arcs between transitions and places, signifying the number of tokens needed. Figure 26 illustrates a Petri net model of a reaction that needs three $Cl_2$ and two P to produce two $PCl_3$. In order for the transition to fire, at least three $Cl_2$ and two P must be available. The firing of the transition absorbs these tokens and produces two tokens in its output place.

Petri nets that allow multiple arcs have been called *generalized Petri nets* [34, 54]. Hack has shown that these nets are equivalent to ordinary Petri nets (at most one arc between a place and a transition). Hence although this change may increase the convenience of use, it does not change the fundamental modeling power or decision power of Petri nets. Most researchers thus use generalized Petri nets in their work, often ignoring the distinction between them and what we have defined as Petri nets in this paper.

A more fundamental extension of Petri nets was undertaken by a number of authors [1, 5, 73] in response to difficulties in the modeling of priority systems with Petri nets. This extension involves so-called zero-testing [53]: the introduction of arcs from a place $p_i$ to a transition $t_j$ which allow the transition to fire only if the place

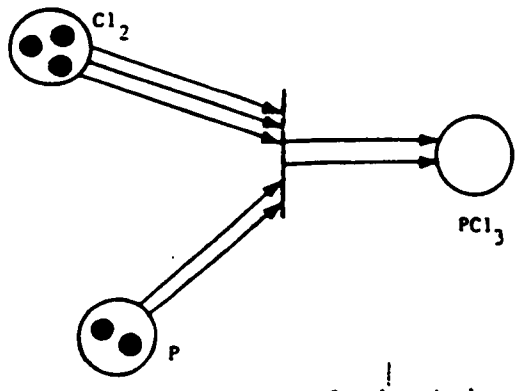



FIGURE 26. Petri net model of a chemical reaction, illustrating the concept of multiple input and output arcs between a transition and a place.

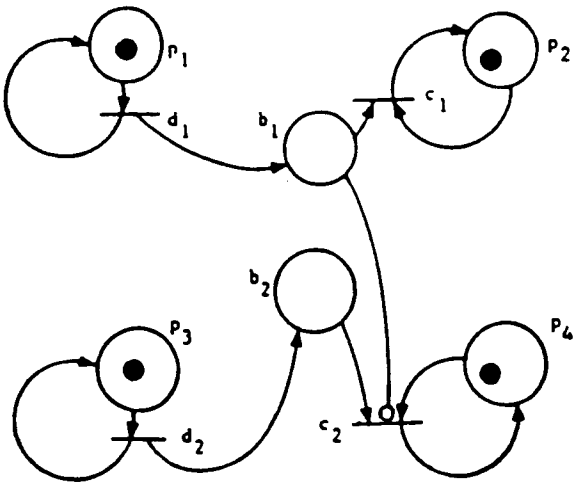



FIGURE 27. An extended Petri net which has no equivalent narrowly defined Petri net.

$p_i$ has zero tokens in it. These special arcs have been drawn in several ways. We represent them as shown in Fig. 27. Note that transition $c_2$ can fire only if places $p_4$ and $b_2$ each have at least one token in them and place $b_1$ has exactly zero tokens. The arc from $b_1$ to $c_2$ is called an *inhibitor* arc; it gives transition $c_1$ priority over transition $c_2$.

The addition of inhibitor arcs is a major extension of the concept of Petri nets. Agerwala has shown that Petri nets extended in this manner have the modeling power of a Turing machine and hence can also be used to show that many decision problems are undecidable [1]. Many other extensions of Petri nets including the introduction of priorities between transitions, time bounds on transition firings [66], or constraint sets that prohibit tokens residing simultaneously in two places [73] are equivalent to Petri nets with inhibitor arcs and hence to Turing machines. In terms of modeling power Petri nets seem to be just below Turing machines, so that any significant extension results in Turing-machine equivalence [78].

### Subclasses of Petri Nets

It was hoped that the limitations on the modeling power of Petri nets relative to Turing machines would be balanced by a compensating increase in decision power. This appears to be the case, since for Petri nets many decision problems are equivalent to the reachability problem, which has been shown to be decidable. However, research on the complexity of the reachability problem has shown that even though it is decidable, it is very difficult to solve. Thus, from a practical point of view, Petri nets may be too powerful to be analyzed.

The result of this has been the definition of a number of subclasses of Petri nets, in hopes of finding a subclass with (known) decision power and still adequate modeling power for practical purposes. These subclasses are defined by restrictions on their structure intended to improve their analyzability.

Two subclasses are most commonly considered, state machines and marked graphs [44]. State machines are Petri nets which are restricted so that each transition has exactly one input and one output. These nets are obviously conservative and hence finite-state. In fact, they are exactly the class of finite-state machines. This is clearly shown by considering the state graph of a finite-state machine, as in Fig. 28a. The nodes of this graph represent the states of the finite-state machine. An arc from state $i$ to state $j$ labeled $x$ indicates that there is a transition from state $i$ to state $j$ with input $x$. Note that this state graph is nondeterministic. The graph of Fig. 28a can be converted to an equivalent Petri net by simply making each state a place, and making each arc between two places a transition. This is illustrated in Fig. 28b. Note that this Petri net is conservative. If the state graph had been nondeterministic, then the Petri net would also have this characteristic. Finite-state machines, being finite, have very high decision power, but they are of limited usefulness in modeling systems which are not finite.

Marked graphs, the dual of state machines, have also been studied extensively [17, 44]. A *marked graph* is a Petri net in which each place has exactly one input transition and one output transition. Algorithms are known for showing that a marked graph is live and safe, and for solving the reachability problem for marked graphs. Thus, marked graphs have high decision power. They have limited modeling power, however, since they

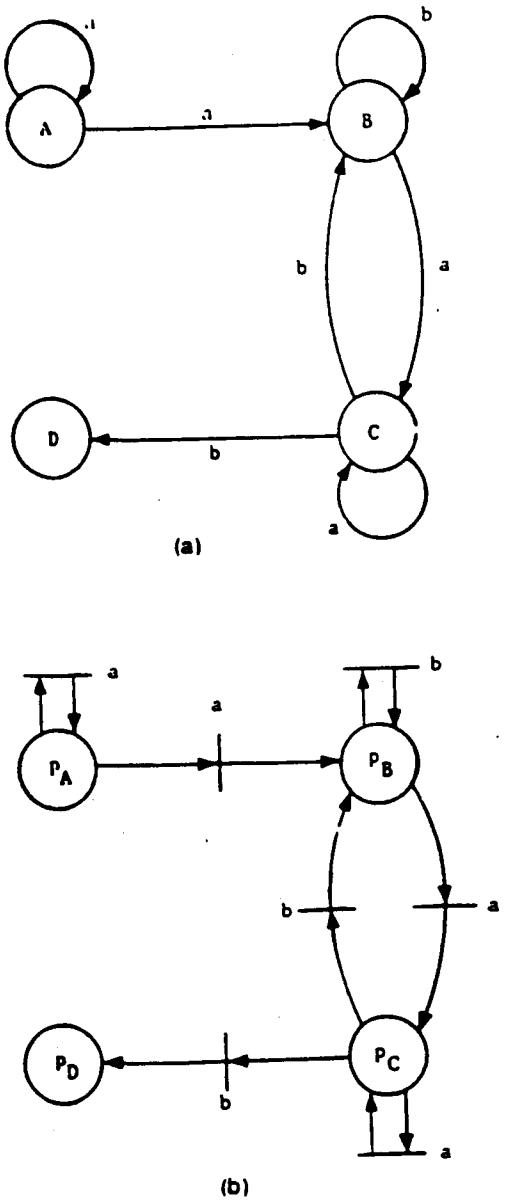



FIGURE 28. Equivalent models of a finite-state machine. (a) State graph model. (b) Petri net model.

are able to model only those systems whose control flow has no branches. In other words, parallel activities can be easily modeled, but not alternative activities.

The problem with modeling data-dependent decisions (branches) in a Petri net is that conflicts may arise, and nets with conflicts seem to be difficult to analyze. Hack has investigated the class of *free-choice Petri nets* [32] in which each arc from a place is either the unique output of the place, or the unique input to a transition. This restriction means that if there is a token in a place then either the token will remain in that place until its unique output transition fires or, if there are multiple outputs for the place, then there is a free choice as to which of the transitions is fired.

Hack and Commoner have shown that liveness and safeness for free-choice Petri nets are decidable and have given necessary and sufficient conditions for these properties. Hack has also shown that free-choice nets can model a class of systems called *production schemata* which are similar to assembly-line systems.

Other subclasses of Petri nets have been defined [32], for example simple Petri nets, but little analysis of them has been done to date. Figure 29 shows allowed and disallowed situations for three subclasses. Landweber and Robertson [58] have studied the classes of conflict-free and persistent Petri nets.

### Related Models

Any discussion of Petri nets would be incomplete without a mention of vector addition systems. These systems were defined by Karp and Miller [51] and are equivalent

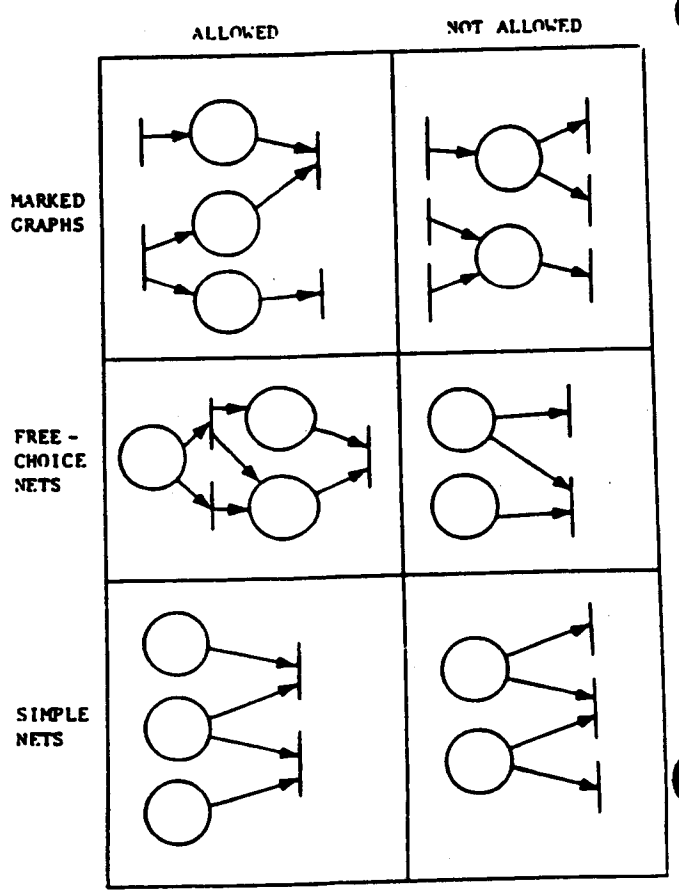



FIGURE 29. Differences between the subclasses of Petri nets.

to Petri nets [34]. A vector addition system is essentially a mathematical formulation, in terms of vectors, of the markings and transitions of a Petri net. Since the mathematical formulation is more convenient for formal manipulation than Petri net graphs, many results are given in terms of vector addition systems, although they apply equally to Petri nets. Vector replacement systems [53] are a related (and equivalent) model based on a generalization of the vector addition systems.

It should also be pointed out that Petri nets are far from the only model of concurrent systems to have been developed. The many other models developed to date include program graphs [87], computation graphs [50], message transmission systems [85], flow graph schemata [89], and complex bilogic directed graphs [31]. Baer has published a survey of some of these models [6]. A comparison of the properties of many of these models [78] has shown that most of them are either subclasses of Petri nets or equivalent to Petri nets. These results have been reinforced by the comparisons of Agerwala [2] which arrive at much the same conclusion concerning the relative modeling power of the various models. The definition of equivalence must be carefully considered, however. Lipton, Snyder, and Zalcstein [63] have compared models using a definition of equivalence considerably different but no less valid than those of Peterson and Bredt or Agerwala, and arrived at important differences in the modeling power of the various models of concurrent systems.

## CONCLUSIONS

The Petri net has been defined as a model for systems exhibiting concurrent asynchronous activities. The major factors that might affect its acceptance are concerns regarding the modeling power and decision power of the model. Although Petri nets are not the only models of asynchronous concurrent systems, they are equivalent to or include most other models. In addition they have a certain clearness and cleanness which permits a simple and natural representation of many systems. Thus they have gained increasing acceptance in the last decade, and their use is growing.

A major modeling system must provide more than simply a convenient representation system, however. It must also provide analysis procedures that can be used to determine properties of the modeled system through the model. Some such analysis procedures for Petri nets do exist, allowing the analysis of systems for boundedness, conservation, coverability, and reachability of a marking. However, other properties, such as inclusion or equivalence of two Petri nets, have been shown to be undecidable. Even though problems such as reachability may be decidable, complexity results tend to indicate that these problems may be intractable, requiring too much computational time and space to be practical. Any significant extension of the Petri net model tends to be equivalent to a Turing machine, and hence analysis of these extensions is not possible due to decidability problems. The subclasses which have been examined have good decision properties, but may be too limited for useful modeling. On this topic as on many others relating to Petri nets, much work remains to be done.

## ACKNOWLEDGMENTS

I am grateful to R. M. Keller, M. Hack, L. H. Landweber, D. Mandrioli, E. L. Robertson, the referees, and the editors for their suggestions and comments.

## BIBLIOGRAPHY

[1] AGERWALA, T. *A complete model for representing the coordination of asynchronous processes*. Hopkins Computer Research Report No. 32. Computer Science Program, Johns Hopkins Univ., Baltimore, Md., July 1974, 58 pp.

[2] AGERWALA, T. *An analysis of controlling agents for asynchronous processes*, Hopkins Computer Research Report No. 35, Computer Science Program, Johns Hopkins Univ., Baltimore, Md., Aug. 1974, 85 pp.

[3] AGERWALA, T.; AND FLYNN M. "Comments on capabilities, limitations and 'correctness' of Petri nets," in *Proc. 1st Annual Symp. Computer Architecture*, G. J. Lipovski, and S. A. Szygenda (Eds.), ACM, N.Y., 1973, pp. 81–86.

[4] ANDERSON, D. W.; SPARACIO, F. J.; AND TOMASULO, R. M. "The IBM System/360 Model 91: Machine philosophy and instruction han-

dling," *IBM J. R. & D.* 11, 1 (Jan. 1967), 8-24.

[5] BAER, J. L. "Modelling for parallel computation: a case study," in *Proc. 1973 Sagamore Computer Conf. Parallel Processing*, Springer-Verlag, N.Y., 1973.

[6] BAER, J. L. "A survey of some theoretical aspects of multiprocessing," *Computing Surveys* 5, 1 (March 1973), 31-80.

[7] BAKER, H. G. *Petri nets and languages*, Computation Structures Group Memo 68, Project MAC, MIT, Cambridge, Mass., May 1972, 6 pp.

[8] BAKER, H. G. "Equivalence problems of Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., June 1973, 53 pp.

[9] BAKER, H. G. *Rabin's proof of the undecidability of the reachability set inclusion problem of vector addition systems*, Computation Structures Group Memo 79, Project MAC, MIT, Cambridge, Mass., July 1973, 18 pp.

[10] BERNSTEIN, A. J. "Program analysis for parallel processing," *IEEE Trans. Electronic Comp.* EC-15, (Oct. 1966), 757-762.

[11] BERNSTEIN, P. A. *Description problems in the modeling of asynchronous computer systems*, Tech. Rep. 48, Dept. Computer Science, Univ. Toronto, Toronto, Canada, Jan. 1973.

[12] CERF, V. G. "Multiprocessors, semaphores, and a graph model of computation," PhD Thesis, Computer Science Dept., Univ. Calif., Los Angeles, April 1972.

[13] CHEN, T. C. "Overlap and pipeline processing," in *Introduction to computer architecture*, H. S. Stone (Ed.), Science Research Associates, Chicago, Ill., 1975, pp. 375-431.

[14] CLARK, W. A. "Macromodular computer systems," in *Proc. 1967 Spring Jt. Comp. Conf.*, Thompson Book Co., Washington, D.C., 1967, pp. 335-336.

[15] COFFMAN, E. G.; AND DENNING, P. J. *Operating systems theory*, Ch. 2, Prentice-Hall, Englewood Cliffs, N. J., 1973, pp. 31-82.

[16] COMMONER, F. G. *Deadlocks in Petri nets*, CA-7206-2311, Applied Data Research, Wakefield, Mass., June 1972, 50 pp.

[17] COMMONER, F.; HOLT, A. W.; EVEN, S.; AND PNUELI, A. "Marked directed graphs," *J. Computer and Systems Science* 5, (Oct. 1971), 511-523.

[18] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "Properties of firing sequences," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975.

[19] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. *Petri nets and commutative grammars*, Internal Report No. 74-5, Laboratorio di Calcolatori, Instituto di Elettrotecnica ed Elettronica del Politecnico di Milano, Italy, March 1974.

[20] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "A decidability theorem for a class of vector-addition systems," *Information Processing Letters* 3, 3 (Jan. 1975), 78-80.

[21] CRESPI-REGHIZZI, S.; AND MANDRIOLI, D. "Petri nets and Szilard languages," *Information and Control* 33, 2 (Feb. 1977), 177-192.

[22] DENNIS, J. B. "Modular asynchronous control structures for a high performance processor," in *Record of the Project MAC Conf. Concurrent and Parallel Computation*, ACM, N. Y., 1970, pp. 55-80.

[23] DENNIS, J. B. (Ed.), *Record of the project MAC conf. concurrent systems and parallel computation*, ACM, N. Y., 1970, 199 pp.

[24] DENNIS, J. B., *Concurrency in software systems*, Computation Structures Group Memo 65-1, Project MAC, MIT, June 1972, 18 pp.; also in *Advanced course in software engineering*, F. L. Bauer (Ed.), Springer-Verlag, Berlin, W. Germany, 1973, pp. 111-127.

[25] DIJKSTRA, E. W., "Cooperating sequential processes," in *Programming languages*, F. Genuys (Ed.), Academic Press, N. Y., 1968, pp. 43-112.

[26] FISCHER, P. C.; MEYER, A. R.; AND ROSENBERG, A. L. "Counter machines and counter languages," *Mathematical Systems Theory* 2, 3 (1968), 265-283.

[27] FURTEK, F. "Modular implementation of Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1971.

[28] FURTEK, F. C. "The logic of systems," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., May 1976; also Tech. Rep. 170, MIT Laboratory Computer Science, June 1976.

[29] GENRICH, H. J. *Einfache nicht-sequentielle Prozesse* (Simple nonsequential processes), Gesellschaft für Mathematik und Datenverarbeitung, Birlinghoven, W. Germany, 1970.

[30] GENRICH, H. J. *The Petri net representation of mathematical knowledge*, GMD-ISF Internal Report 75-06, Institut für Informationssystemforschung, Gesellschaft für Mathematik und Datenverarbeitung, Birlinghoven, W. Germany, 1975.

[31] GOSTELOW, K. P. "Flow of control, resource allocation and the proper termination of programs," PhD Thesis, Computer Science Dept., Univ. Calif., Los Angeles, Dec. 1971, 219 pp.

[32] HACK, M. "Analysis of production schemata by Petri nets," MS Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass.; also: MAC TR-94, Project MAC, MIT, Feb. 1972, 119 pp; Errata: Hack, M. *Corrections to 'Analysis of production schemata by Petri nets'*, Computation Structures Note 17, Project MAC, MIT, June 1974, 11 pp.

[33] HACK, M. *A Petri net version of Rabin's undecidability proof for vector addition systems*, Computation Structures Group Memo 94, Project MAC, MIT, Cambridge, Mass., Dec. 1973, 12 pp.

[34] HACK, M. *Decision problems for Petri nets and vector addition systems*, Computation Structures Group Memo 95, Project MAC, MIT, Cambridge, Mass., March 1974; also Technical Memo 59, Project MAC, MIT, March, 1975.

[35] HACK, M. *The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems*, Computation Structures Group Memo 107, Project MAC, MIT, Cambridge, Mass., Aug. 1974, 9 pp.; also in *Proc. 15th Annual Symp. Switching and Automata*, IEEE, N. Y., 1974.

[36] HACK, M. *The equality problem for vector addition systems is undecidable*, Computation Structures Group Memo 121, Project MAC, MIT, Cambridge, Mass., April 1975, 32 pp.; also in *Theoretical Computer Science* 2, 1 (June 1976).

[37] HACK, M. *Petri net languages*, Computation Structures Group Memo 124, Project MAC, MIT, Cambridge, Mass., June 1975, 128 pp.; also TR 159, Laboratory Computer Science,

MIT, March 1976.

[38] HACK, M. "Decidability questions for Petri nets," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Dec. 1975; also TR-161, Laboratory Computer Science, MIT, June 1976, 194 pp.

[39] HACK, M.; AND PETERSON, J. L. "Petri nets and languages," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975.

[40] HANSAL, A.; AND SCHWAB, G. M. *On marked graphs III*, Report LN 25.6.038, IBM Vienna Laboratories, Vienna, Austria, Sept. 1972.

[41] HENHAPL, W. *Firing sequences of marked graphs*, Report LN 25.6.023, IBM Vienna Laboratories, Vienna, Austria, June 1972.

[42] HENHAPL, W. *Firing sequences of marked graphs II*, Report LN 25.6.036, IBM Vienna Laboratories, Vienna, Austria, June 1972.

[43] HOLT, A. W.; SAINT, H.; SHAPIRO, R. M.; AND WARSHALL, S. *Final report of the information system theory project*, Tech. Rep. RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, N. Y., Sept. 1968.

[44] HOLT, A. W.; AND COMMONER, F. *Events and condition*, Applied Data Research N.Y., 1970; also in *Record Project MAC Conf. Concurrent Systems and Parallel Computation*, (Chapters I, II, IV, and VI) ACM, N.Y., 1970, pp. 3–52.

[45] HOLT, R. C. "On deadlock in computer systems," PhD Thesis, Dept. Computer Science, Cornell Univ., Ithaca, N.Y., Jan. 1971; also TR 71-91, Dept. Computer Science, Cornell Univ.; and TR CSRG-6, Computer Science Research Group, Univ. Toronto, Toronto, Canada, July 1972.

[46] IZBICKI, H. *On marked graphs*, Report LR 25.6.023, IBM Vienna Laboratories, Vienna, Austria, Sept. 1971.

[47] IZBICKI, H. *On marked graphs II*, Report LN 25.6.029, IBM Vienna Laboratories, Vienna, Austria, Jan. 1972.

[48] JACK, L. "Graphical representation for fault tolerant phenomena," presented at Seminar, Dept. Electrical Engineering, Univ. Texas, Austin, Jan. 1976.

[49] JONES, N. D.; LANDWEBER, L. H.; AND LIEN, Y. E. *Complexity of some problems in Petri nets*, TR-276, Comp. Science Dept., Univ. Wisconsin-Madison, Sept. 1976, 43 pp.; to appear in *Theor. Comp. Sci.*

[50] KARP, R. M.; AND MILLER, R. E. "Properties of a model for parallel computation: determinacy, termination, queueing," *SIAM J. Appl. Math.* 14, 6 (Nov. 1966) 1390–1411.

[51] KARP, R. M.; AND MILLER, R. E. "Parallel program schemata," *J. Computer and Systems Science* 3, 4 (May 1969), 167–195.

[52] KASAMI, T.; TOKURA, N.; AND PETERSON, W. W. "Vector addition systems and synchronization problems of concurrent processes," draft manuscript, 1974.

[53] KELLER, R. M. *Vector replacement systems: a formalism for modelling asynchronous systems*, Tech. Rep. 117, Computer Science Laboratory, Princeton Univ., Princeton, N.J., Dec. 1972; Revised: Jan. 1974, 57 pp.

[54] KELLER, R. M. *Generalized Petri nets as models for system verification*, Tech. Rep. 202, Dept. Electrical Engineering, Princeton Univ., Princeton, N.J., Aug. 1975.

[55] KELLER, R. M. "Look-ahead processors," *Computing Surveys* 7, 4 (Dec. 1975), 177–196.

[56] KELLER, R. M. "Formal verification of parallel programs," *Comm. ACM* 19, 7 (July 1976), 371–384.

[57] KOSARAJU, S. R. *Limitations of Dijkstra's semaphore primitives and Petri nets*, Tech. Rep. 25, Johns Hopkins Univ., Baltimore, Md. May 1973, 5 pp.; also in *Operating Systems Review* 7, 4 (Oct. 1973), 122–126.

[58] LANDWEBER, L. H.; AND ROBERTSON, E. L. *Properties of conflict free and persistent Petri nets*, Tech. Rep. 264, Computer Sciences Dept., Univ. Wisconsin-Madison, Madison, Wisc., Dec. 1975, 30 pp.

[59] LAUER, P. E.; AND CAMPBELL, R. H. *A description of path expressions by Petri nets*, Tech. Rep. 64, Computing Laboratory, Univ. Newcastle Upon Tyne, England, May 1974, 39 pp.

[60] LAUER, P. E. *Path expressions as Petri nets, or Petri nets with fewer tears*, MRM 70, Computing Laboratory, Univ. Newcastle Upon Tyne, England, Jan. 1974, 61 pp.

[61] LAUTENBACH, K.; AND SCHMID, H. A. "Use of Petri nets for proving correctness of concurrent process systems," in *Proc. IFIP Congress 74*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974, pp. 187–191.

[62] LIEN, Y. E. "Termination properties of generalized Petri nets," *SIAM J. Computing* 5, 2 (June 1976), 251–265.

[63] LIPTON, R. J.; SNYDER, L.; AND ZALCSTEIN, Y. "A comparative study of models of parallel computation," in *Proc. 15th Annual Symp. Switching and Automata*, IEEE, N.Y., 1974, pp. 145–155.

[64] LIPTON, R. "The reachability problem and the boundedness problem for Petri nets are exponential-space hard," presented at *MIT Conf. Petri Nets and Related Methods*, MIT, Cambridge, Mass., July 1975; also TR-62, Dept. Computer Science, Yale Univ., New Haven, Conn., Jan. 1976.

[65] MELDMAN, J. A.; AND HOLT, A. W. "Petri nets and legal systems," *Jurimetrics J.* 12, 2 (Dec. 1971).

[66] MERLIN, P. A. "A study of recoverability of computing systems," PhD. Thesis, Dept. Information and Computer Science, Univ. Calif. Irvine, 1974, 165 pp.

[67] MISUNAS, D. "Petri nets and speed independent design," *Comm. ACM* 16, 8 (Aug. 1973), 474–481.

[68] MILLER, R. E. *A comparison of some theoretical models of parallel computation*, RC 4230, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.; also *IEEE Trans. Comp.* C-22, 8 (Aug. 1973), 710–717.

[69] MURATA, T.; AND CHURCH, R. W. *Analysis of marked graphs and Petri nets by matrix equations*, Research Report MDC 1.1.8, Dept. Information Engineering, Univ. Illinois, Chicago Circle, Nov. 1975, 25 pp.

[70] NASH, B. O. "Reachability problems in vector addition systems," *American Math. Monthly* 80, 3 (March 1973), 292–295.

[71] NOE, J. D. *A Petri net model of the CDC 6400*, Report 71-04-03, Computer Science Dept., Univ. Washington, 1971; also in *Proc. ACM SIGOPS Workshop on System Performance Evaluation*, ACM, N.Y., 1971, pp. 362–378.

[72] NOE, J. D.; AND NUTT, G. J. "Macro E-Nets for representation of parallel systems," *IEEE*

*Trans. Comp.* C-22, 8 (Aug. 1973), 718–727.

[73] PATIL, S. S. "Coordination of asynchronous events," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., May 1970; also MAC TR-72, Project MAC, MIT, June 1970, 236 pp.

[74] PATIL, S. S. *Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes*, Computation Structures Group Memo 57, Project MAC, MIT, Cambridge, Mass., Feb. 1971.

[75] PATIL, S. S. *Circuit implementation of Petri nets*, Computation Structures Group Memo 73, Project MAC, MIT, Cambridge, Mass., Dec. 1972, 14 pp.

[76] PATIL, S. S.; AND DENNIS, J. B. *The description and realization of digital systems*, Computation Structures Group Memo 71, Project MAC, MIT, Cambridge, Mass., Oct. 1972; also in *Proc. Sixth Annual IEEE Computer Society Internatl. Conf. Digest of Papers*, IEEE, N.Y., 1972.

[77] PETERSON, J. L. "Modelling of parallel systems," PhD Thesis, Dept. Electrical Engineering, Stanford Univ., Stanford, Calif., Dec. 1973, 241 pp.

[78] PETERSON, J. L.; AND BREDT, T. H. "A comparison of models of parallel computation," in *Proc. IFIP Congress 74*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974, pp. 466–470.

[79] PETERSON, J. L. "Computation sequence sets," *J. Computer and System Sciences* 13, 1 (Aug. 1976), 1–24.

[80] PETRI, C. A. "Kommunikation mit Automaten," *Schriften des Rheinisch-Westfalischen Institutes für Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany 1962; translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, N.Y., 1965, 89 pp.

[81] PETRI, C. A. "Concepts of net theory," in *Proc. Symp. and Summer School on Mathematical Foundations of Computer Science*, High Tatras, Sept. 3–8, 1973, Math. Inst. Slovak Academy of Science, 1973, pp. 137–146.

[82] PETRI, C. A. *Interpretations of net theory*, Interner Bericht 75-07, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, W. Germany, July 1975, 34 pp.

[83] RACKOFF, C. *The covering and boundedness problems for vector addition systems*, Tech. Rep. 97, Dept. Computer Science, Univ. Toronto, Toronto, Canada, July 1976, 14 pp.

[84] RAMCHANDANI, C. "Analysis of asynchronous concurrent systems by timed Petri nets," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., 1974; also MAC-TR-120, Project MAC, MIT, Feb. 1974.

[85] RIDDLE, W. E. "The modelling and analysis of supervisory systems," PhD Thesis, Computer Science Dept., Stanford Univ., Stanford, Calif., March 1972, 173 pp.

[86] RIDDLE, W. E. *The equivalence of Petri nets and message transmission models*, SRM 97, Univ. Newcastle Upon Tyne, England, Aug. 1974, 11 pp.

[87] RODRIGUEZ, J. E. "A graph model for parallel computation," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1967, 120 pp.

[88] SACERDOTE, G.; AND TENNEY, R. L. "The decidability of the reachability problem for vector addition systems," (submitted for publication), Nov. 1976, 8 pp.

[89] SLUTZ, D. R. "The flow graph schemata model of parallel computation," PhD Thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., Sept. 1968.

[90] SHAPIRO, R. M.; AND SAINT, H. "A new approach to optimization of sequencing decisions," *Annual Review of Automatic Programming* 6, 5, (1970), 257–288.

[91] THORNTON, J. E. *Design of a computer: the Control Data 6600*, Scott, Foresman and Co., Glenview, Ill., 1970, 181 pp.

[92] TSICHRITZIS, D. *Modular system description*, Tech. Rep. 33, Dept. Computer Science, Univ. Toronto, Toronto, Canada, Oct. 1971, 20 pp.

[93] VAN LEEUWEN, J. "A partial solution to the reachability-problem for vector addition systems," in *Proc. 6th Annual ACM Symp. Theory of Computing*, ACM, N.Y., 1974, pp. 303–309.