8.5 PETRI NETS

Consider the computer program shown in Figure 8.5.1. Normally, the instructions would be processed sequentially—first, $A = 1$, then $B = 2$, and so on. However, notice that there is no logical reason that prevents the first three instructions—$A = 1$; $B = 2$; $C = 3$—from being processed in any order or concurrently. With the continuing decline of the cost of computer hardware, and processors in particular, there is increasing interest in concurrent processing to achieve greater speed and efficiency. The use of **Petri nets**, graph models of concurrent processing, is one method of modeling and studying concurrent processing.

$A = 1$
$B = 2$
$C = 3$
$A = A+1$
$C = B+C$
$B = A+C$

FIGURE 8.5.1
A computer program.

DEFINITION 8.5.1

A *Petri net* is a directed graph $G = (V, E)$, where $V = P \cup T$ and $P \cap T = \emptyset$. Any edge e in E is incident on one member of P and one member of T. The set P is called the set of *places* and the set T is called the set of *transitions*.

Less formally, a Petri net is a directed, bipartite graph where the two classes of vertices are called places and transitions. In general, parallel edges are allowed in Petri nets; however, for simplicity, we will not permit parallel edges.

EXAMPLE 8.5.2

An example of a Petri net is given in Figure 8.5.2. Places are typically drawn as circles and transitions as bars or rectangular boxes.

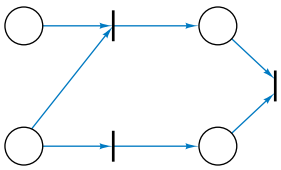


FIGURE 8.5.2 A Petri net. Circles are *places* and bars are *transitions*. □

DEFINITION 8.5.3

A *marking* of a Petri net assigns each place a nonnegative integer. A Petri net with a marking is called a *marked Petri net* (or sometimes just a Petri net).

If a marking assigns the nonnegative integer n to place p, we say that there are n *tokens* on p. The tokens are represented as black dots.

EXAMPLE 8.5.4

An example of a marked Petri net is given in Figure 8.5.3.

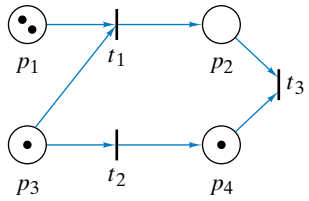


FIGURE 8.5.3 A marked Petri net.

□

In modeling, the places represent **conditions**, the transitions represent **events**, and the presence of at least one token in a place (condition) indicates that that condition is met.

EXAMPLE 8.5.5 ***Petri Net Model of a Computer Program***

In Figure 8.5.4 we have modeled the computer program of Figure 8.5.1. Here the events (transitions) are the instructions, and the places represent the conditions under which an instruction can be executed. □

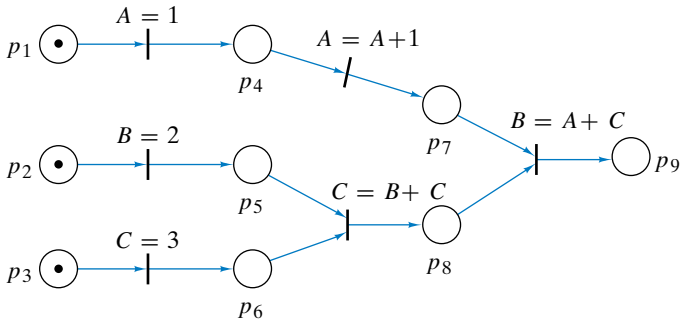


FIGURE 8.5.4 The program of Figure 8.5.1 as a Petri net. The tokens indicate that the conditions for executing $A = 1$, $B = 2$, and $C = 3$ are met.

DEFINITION 8.5.6

In a Petri net, if an edge is directed from place p to transition t, we say that p is an *input place* for transition t. An *output place* is defined similarly. If every input place for a transition t has at least one token, we say that t is *enabled*. A *firing* of an enabled transition removes one token from each input place and adds one token to each output place.

EXAMPLE 8.5.7

In the Petri net of Figure 8.5.3, places p_1 and p_3 are input places for transition t_1. Transitions t_1 and t_2 are enabled, but transition t_3 is not enabled. If we fire transition t_1, we obtain the marked Petri net of Figure 8.5.5. Transition t_3 is

now enabled. If we then fire transition t_3, we obtain the net shown. At this point no transition is enabled and thus none may be fired. □

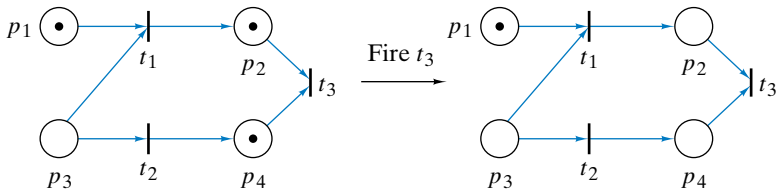


FIGURE 8.5.5 Firing transition t_3.

DEFINITION 8.5.8

If a sequence of firings transforms a marking M to a marking M', we say that M' is *reachable* from M.

EXAMPLE 8.5.9

In Figure 8.5.6, M'' is reachable from M by first firing transition t_1 and then firing t_2. □

In modeling, the firing of a transition simulates the occurrence of that event. Of course, an event can take place only if all of the conditions for its execution have been met; that is, the transition can be fired only if it is enabled. By putting tokens in places p_1, p_2, and p_3 in Figure 8.5.4, we show that the conditions for executing the instructions $A = 1$, $B = 2$, and $C = 3$ are met. The program is ready to be executed. Since the transitions $A = 1$, $B = 2$, and $C = 3$ are enabled, they can be fired in any order or concurrently. Transition $C = B + C$ is enabled only if places p_5 and p_6 have tokens. But these places will have tokens only if transitions $B = 2$ and $C = 3$ have been fired. In other words, the condition under which the event $C = B + C$ can occur is that $B = 2$ and $C = 3$ must have been executed. In this way we model the legal execution sequences of Figure 8.5.1 and the implicit concurrency within this program.

Among the most important properties studied in Petri net theory are **liveness** and **safeness**. Liveness is related to the absence of deadlocks and safeness is related to bounded memory capacity.

EXAMPLE 8.5.10 ***Petri Net Model of a Shared Computer System***

Two persons are sharing a computer system that has a disk drive D and a printer P. Each person needs both D and P. A possible Petri net model of this situation is shown in Figure 8.5.7. The marking indicates that both D and P are available.

Now suppose that person 1 requests D and then P (while person 2 requests neither). The occurrences of these events are simulated by first firing transition "request D" and then firing transition "request P" for person 1. The

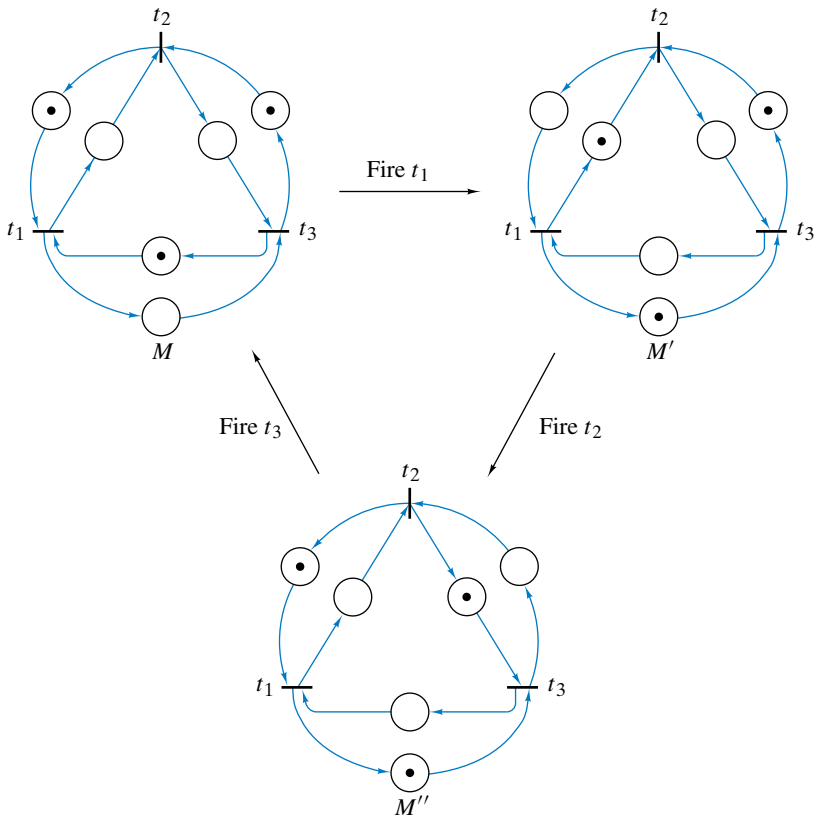


FIGURE 8.5.6 Marking M'' is reachable from M by firing t_1 and then t_2.

resulting Petri net is shown in Figure 8.5.8. When person 1 finishes the processing and releases D and P, simulated by firing the transitions "process" and then "release D and P," we return to the Petri net of Figure 8.5.7. If person 2 requests D and then P (while person 1 requests neither), we obtain a similar firing sequence.

Again, assume that we have the situation of Figure 8.5.7. Now suppose that person 1 requests D and then person 2 requests P. After the appropriate transitions are fired to simulate the occurrences of these events, we obtain the Petri net of Figure 8.5.9. Notice that at this point, no transition can fire. Person 1 is waiting for person 2 to release P and person 2 is waiting for person 1 to release D. Activity within the system stops. We say that a deadlock occurs. Formally, we say that a marked Petri net is **deadlocked** if no transition can fire. Prevention of deadlocks within concurrent processing environments is a major practical concern. □

Example 8.5.10 motivates the following definition.

DEFINITION 8.5.11

A marking M for a Petri net is *live* if, beginning from M, no matter what sequence of firings has occurred, it is possible to fire any given transition by proceeding through some additional firing sequence.

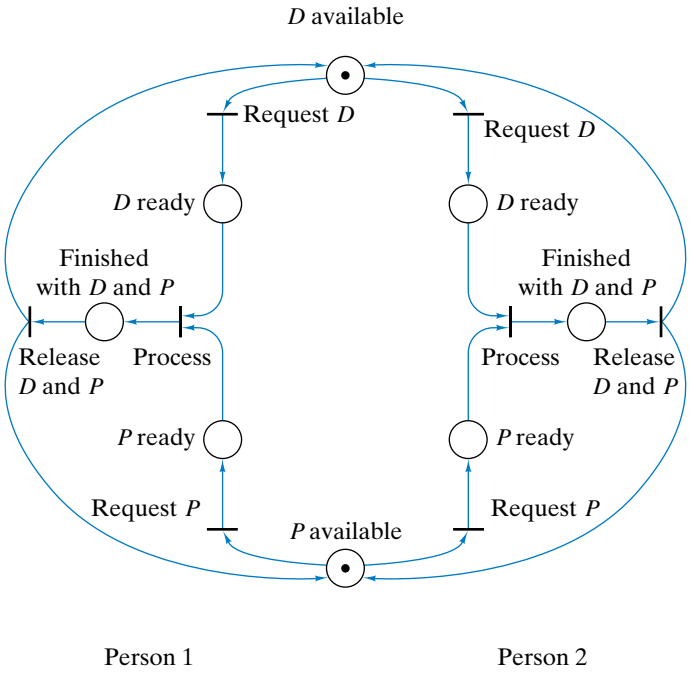


FIGURE 8.5.7 A Petri net model of a shared computer system. Each person needs disk drive D and printer P. The marking indicates that D and P are available.

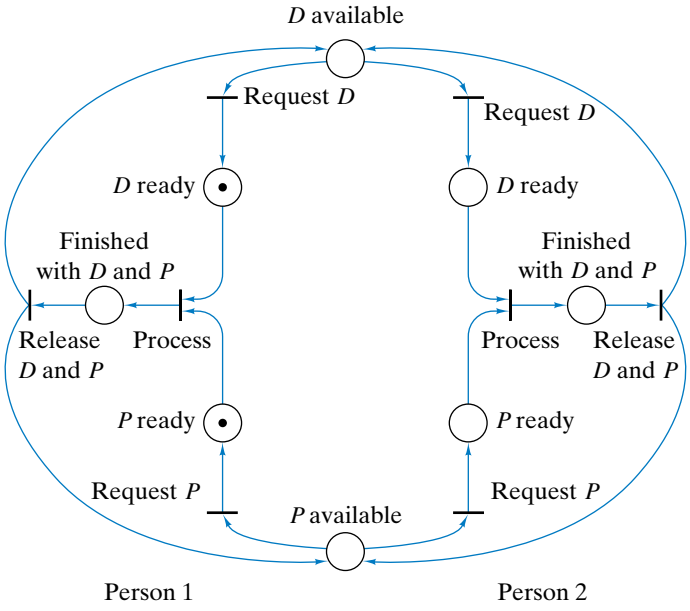


FIGURE 8.5.8 The Petri net of Figure 8.5.7 after firing "request D" and then "request P" for person 1. After person 1 finishes processing and releases D and P, simulated by firing "process" and then "release D and P," we obtain the Petri net of Figure 8.5.7 again.

If a marking M is live for a Petri net P, then no matter what sequence of

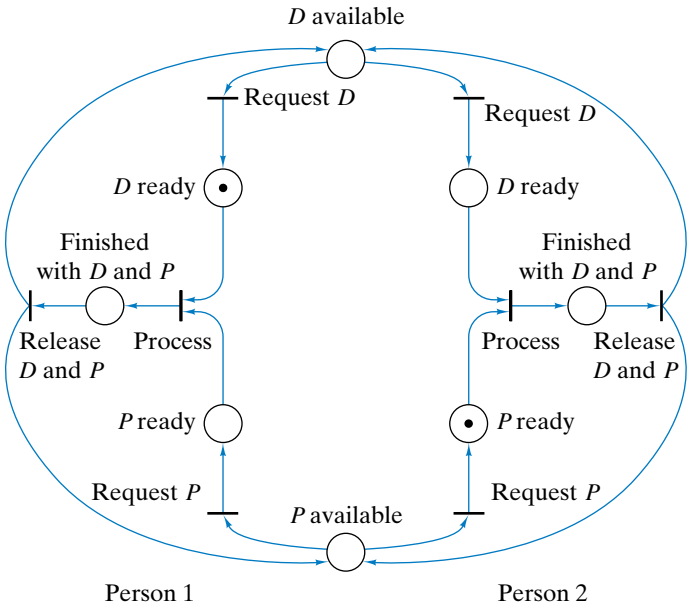


FIGURE 8.5.9 The Petri net of Figure 8.5.7 after firing "request D" for person 1 and "request P" for person 2. At this point the Petri net is deadlocked; that is, no transition can fire.

transitions is fired, P will never deadlock. Indeed, we can fire any transition by proceeding through some additional firing sequence.

EXAMPLE 8.5.12

The marking M of the net of Figure 8.5.6 is live. To see this, notice that the only transition for marking M that can be fired is t_1, which produces marking M'. The only transition for marking M' that can be fired is t_2, which produces marking M''. The only transition for marking M'' that can be fired is t_3, which returns us to marking M. Thus any firing sequence, starting with marking M, produces one of the markings M, M', or M'' and from there we can fire any transition t_1, t_2 or t_3 by proceeding as in Figure 8.5.6. Therefore, the marking M for the net of Figure 8.5.6 is live. □

EXAMPLE 8.5.13

The marking shown in Figure 8.5.4 is not live since after transition $A = 1$ is fired, it can never fire again. □

If a place is regarded as having limited capacity, **boundedness** assures us that no place will overflow.

DEFINITION 8.5.14

A marking M for a Petri net is *bounded* if there is some positive integer n having the property that in any firing sequence, no place ever receives more

than n tokens. If a marking M is bounded and in any firing sequence no place ever receives more than one token, we call M a *safe* marking.

If each place represents a register capable of holding one computer word and if an initial marking is safe, we are guaranteed that the memory capacity of the registers will not be exceeded.

EXAMPLE 8.5.15

The markings of Figure 8.5.6 are safe. The marking M of Figure 8.5.10 is not safe, since as shown, if transition t_1 is fired, place p_2 then has two tokens. By listing all the markings reachable from M, it can be verified that M is bounded and live (see Exercise 7). □

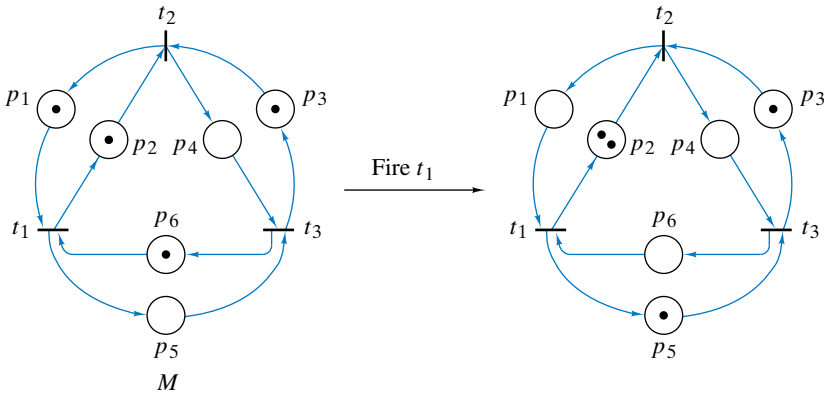


FIGURE 8.5.10 Marking M is not safe. After t_1 is fired, p_2 holds two tokens.

Exercises

In Exercises 1–3, model each program by a Petri net. Provide a marking that represents the situation prior to execution of the program.

1.
$A = 1$
$B = 2$
$C = A + B$
$C = C + 1$

2.
$A = 2$
$B = A + A$
$C = 3$
$D = A + A$
$C = A + B + C$

3.
$A = 1$
$S = 0$
10 $S = S + A$
$A = A + 1$
GOTO 10

4. Describe three situations involving concurrency that might be modeled as Petri nets.

5. Give an example of a marked Petri net in which two transitions are enabled, but firing either one disables the other.

6. Consider the following algorithm for washing a lion.

1. Get lion.
2. Get soap.
3. Get tub.
4. Put water in tub.
5. Put lion in tub.
6. Wash lion with soap.
7. Rinse lion.
8. Remove lion from tub.
9. Dry lion.

Model this algorithm as a Petri net. Provide a marking that represents the situation prior to execution.

7. Show that the marking M of Figure 8.5.10 is live and bounded.

Answer the following questions for each marked Petri net in Exercises 8–12.

(a) Which transitions are enabled?

(b) Show the marking that results from firing t_1.

(c) Is M live?

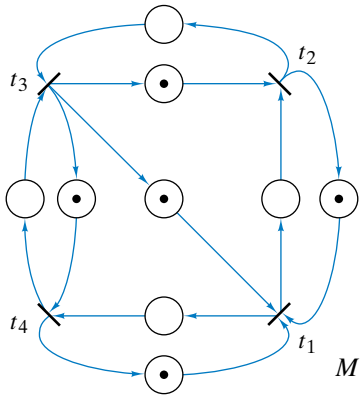
(d) Is M safe?

(e) Is M bounded?

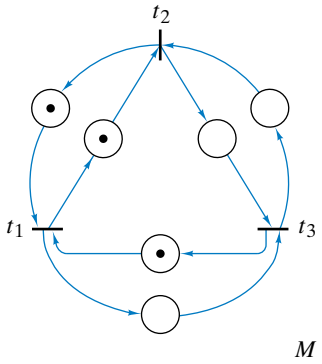
(f) Show or describe all markings reachable from M.

(g) Exhibit a marking (other than the marking that puts zero tokens in each place) not reachable from M.

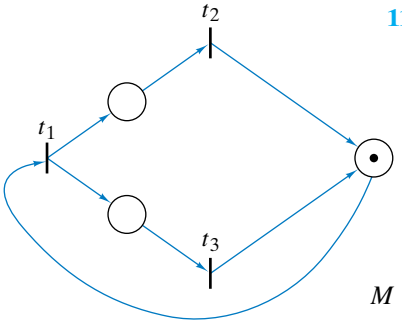
8.



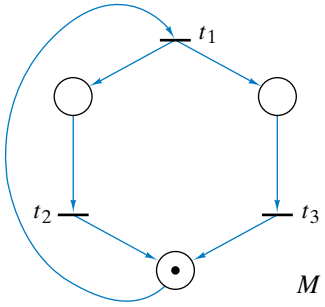
9.



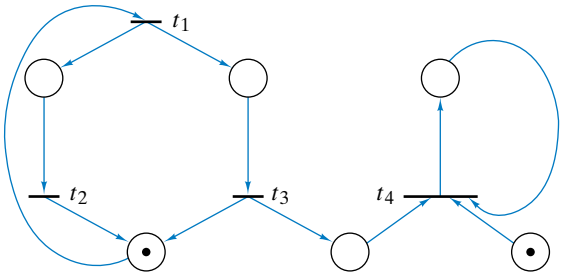
10.



11.



★ **12.**



M

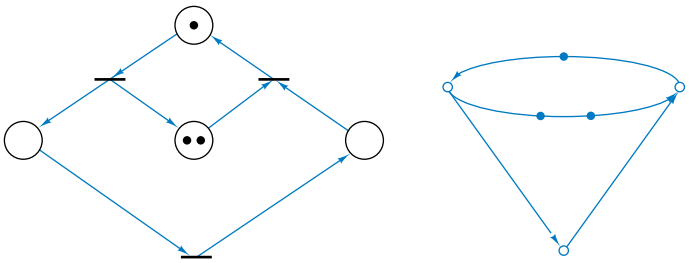
13. Give an example of a Petri net with a marking that is safe, but not live.

14. Give an example of a Petri net with a marking that is bounded, but not safe.

15. The **Dining Philosophers' Problem** (see [Dijkstra, 1968]) concerns five philosophers seated at a round table. Each philosopher either eats or meditates. The table is set alternately with one plate and one chopstick. Eating requires two chopsticks so that if each philosopher picks up the chopstick to the right of the plate, none can eat—the system will deadlock. Model this situation as a Petri net. Your model should be live so that the system will not deadlock and so that, at any point, any philosopher can potentially either eat or meditate.

16. Develop an alternative Petri net model for the situation of Example 8.5.10 that prevents deadlock.

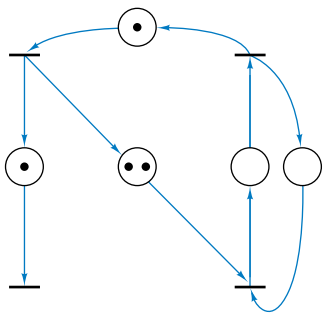
If each place in a marked Petri net P has one incoming and one outgoing edge, then P can be redrawn as a directed graph where vertices correspond to transitions and edges to places. The tokens are placed on the edges. Such a graph is called a *marked graph*. Here we show a marked Petri net and its representation as a marked graph.



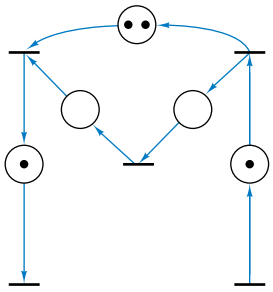
(a) (b)

17. Which Petri nets in Exercises 8–17 can be redrawn as marked graphs?

18. Redraw the marked Petri net as a marked graph.



19. Redraw the marked Petri net as a marked graph.



The *token count* of a simple directed cycle in a marked graph is the number of tokens on all the edges in the cycle.

20. Show that the token count of a simple directed cycle does not change during any firing sequence.

★ **21.** Show that a marking M for a marked graph G is live if and only if M places at least one token in each simple directed cycle in G.

★ **22.** Show that a live marking is safe for a marked graph G if and only if every edge in G belongs to a simple directed cycle with token count one.

23. Give an example of a marked graph with a nonlive marking in which every edge belongs to a simple directed cycle with token count one.

24. Let G be a marked graph. Show that each edge in G is contained in a simple directed cycle if and only if every marking for G is bounded.

★ **25.** Let G be a directed graph where, if we ignore the direction of the edges in G, G is connected as an undirected graph. Show that G has a live and safe marking if and only if given any two vertices v and w in G there is a directed path from v to w.