Notes on Petri Nets

Introduction

Petri nets were invented by Carl Adam Petri in 1939 at the age of 13. This work was the foundation for his 1962 doctoral dissertation entitled *Kommunikation mit Automaten*. Petri nets have been used in a variety of fields including computer science, chemistry, and biology. He retired from the Theoretical Foundation of Computer Science group at the University of Hamburg in 1991.

Figure 1: Carl Adam Petri

Petri nets are a graphical for representing a system in which there are multiple independent activities in progress at the same time. The ability to model multiple activities differentiates Petri nets from finite state machines. In a finite state machine there is always a single "current" state that determines which action can next occur. In Petri nets there may be several states any one of which may evolve by changing the state of the Petri net. Alternatively, some, of even all, of these states may evolve in parallel causing several independent changes to the Petri net to occur at once.

Basic Structure

A Petri net consists of four elements: places, transitions, edges, and tokens. Graphically, places are represented by circles, transitions by rectangles, edges by directed arrows, and tokens by small solid (filled) circles. There are a wide variety of extensions to Petri nets. These extensions add features to model probabilistic behavior, allow weighted edges, or have tokens of various colors among others. Only the most basic Petri net concepts will be covered here.

A basic Petri net is shown in Figure 2. This Petri net has four places, labeled P0 through P4, and three transitions, labeled T0 through T2. Notice that places P0 and P2 each have a single token represented by the black dot inside each place. Edges, represented as directed arcs, connect places to transitions and transitions to places. In a properly formed Petri net, places cannot be directly connected to other places and transitions cannot be directly connected to other transitions. Also notice that the Petri net may contain cycles. The Petri net in Figure 2 contains two cycles. One cycle contains P0, T0, P1, T1, P3, and

T3. The other cycle contains T1, P4, T2, and P2. Cycles are common in Petri nets which represent activities that happen repeatedly. For example, a web server repeated services incoming requests to deliver web page content to different clients.

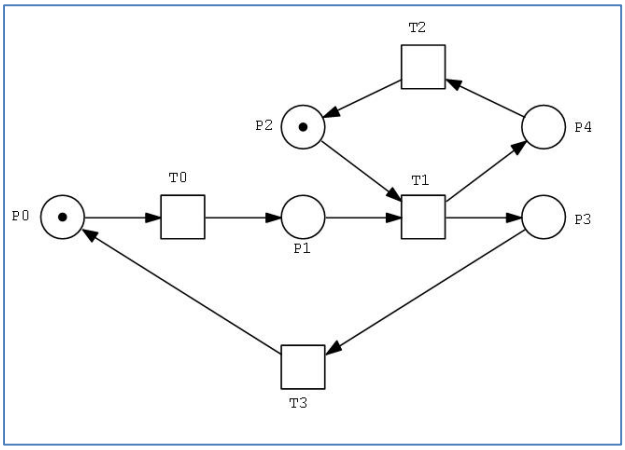

Figure 2: A Basic Petri Net

The state of a Petri net is represented by the occurrence of the tokens at various places. The state of the Petri net in Figure 2 has tokens at places P0 and P2. It will be shown that in another state of this Petri net there are tokens at states P1 and P2. Yet another state has tokens at states P3 and P4. Not all placements of tokens at places represent a possible state of the system. For example, the Petri net in Figure 2 will never have as a possible state one in which the only tokens are at places P1 and P4. Which states are possible and which are not are determined by the structure of the Petri net and the rules that define how a Petri net changes its state.

A Petri net changes from one state to the next state when a transition "fires". The firing of a transition involves the transition's input places and output places. The input places for a transition are all those places that have an edge directed from the place to the transition. The output places of a transition are all those places that have an edge directed from the transition to the place. For example, in Figure 2 the input places for transition T1 are places P1 and P2. The output place for transition T0 is place P1 while the output places for transition T1 are places P3 and P4.

The firing rules for a transition are:

- a transition is able to fire when there is at least one token on each of the transition's input places, and
- when a transition fires it removes one token from each of its input places and produces a single token on each of its output places.

A transition that is able to fire is said to be enabled and otherwise disabled. If there is more than one enabled transition any one of enabled transitions may be the next one to fire. That is, Petri nets are able to model systems with *non-deterministic behavior*. An example of this will be shown later.

The Petri net in Figure 2 will be used to explain how a Petri net changes from one state to the next. In Figure 2 the only transition that is able to fire is transition T0 because it has a single input place, P0, and that input place has at least one token. Notice that transition T1 is not able to fire because it has two input places, P1 and P2, and P1 does not have at least one input token. As a result of the firing on transition T0, the token in place P0 is removed and a single token is created in place P1. This state is shown in Figure 3.

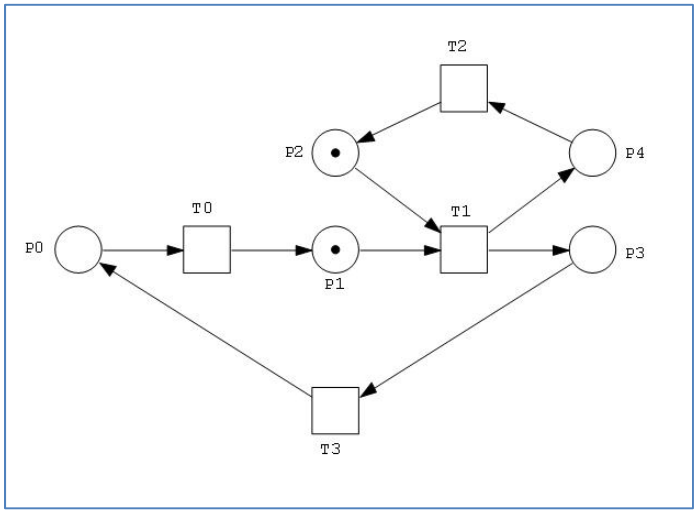

Figure 3: State after the first transition

In the state of the Petri net shown in Figure 3 transition T1 is able to fire because there are input tokens on each of its two input places, P1 and P2. Notice that transition T1 was not able to fire in the previous state of the Petri net (as shown in Figure 2). The firing of transition T0 in the earlier state create a new state (the one shown in Figure 3) in which transition T1 is now able to fire. It is common to find that the firing of a transition creates a new state in which previously disabled (i.e., unable to fire) transitions now become enabled (i.e., able to fire). Notice that in Figure 3, transition T1 is the only transition that is enabled. The firing of transition T1 in the state shown in Figure 3 produces the new state shown in Figure 4.

In the state of the Petri net shown in Figure 4 both transitions T2 and T3 are enabled (i.e., able to fire). As noted about, these transitions may fire in either order because the Petri net does not determine which one of the two transitions is the next one to fire. The next state of the Petri net is, thus, not uniquely determined. The next state can be the one following the firing of transition T3 or the one following the firing of transition T2. The reader should draw both of these states and see that they are different.

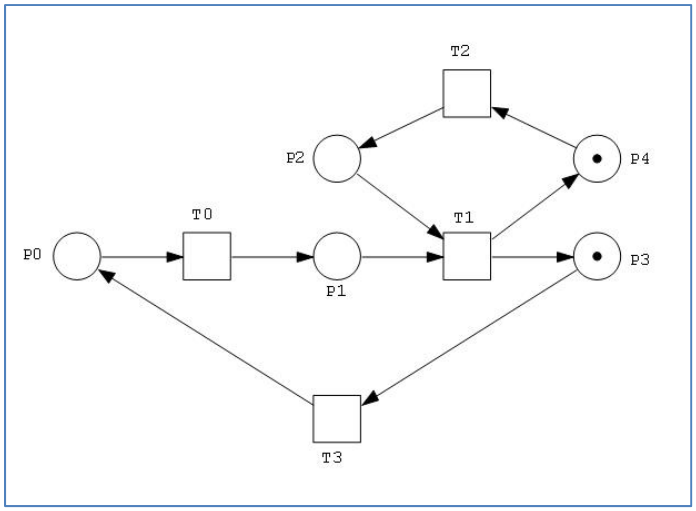

Figure 4: State after Transition T1 fires.

The state shown in Figure 4 will eventually lead back to one of the previous states. The possibilities are:

- transition T2 fires and then transition T3 fires, leading to the state shown in Figure 2,
- transition T3 fires and then transition T2 fires, leading to the state shown in Figure 2, or
- transition T3 fires, transition T0 fires, and then transition T2 fires, leading to the state shown in Figure 3.

This Petri net will continue to transition among these states repeatedly.

Extended Edge Types

There are various extensions to the basic Petri net structure, two of which are read edges and inhibitor edges. These new types of edges have different meanings from the normal edges and also have different graphical representations. Both read edges and inhibitor edges are also restricted in that they can only be drawn *from* places *to* transitions. Recall that regular edges can also be drawn from transitions to places.

Figure 5 shows the use of read edges in three cases. The read edge is depicted graphically as an arc (a solid line) from a place to a transition where a closed circle (a black dot) is drawn at the point where the arc meets the transition. In Figure 5(a) the read edge is drawn from place A to the single transition; similarly for the read edge shown in Figure 5(b) and 5(c).

Like a normal edge, a read edge influences whether the transition to which it is connected is able to fire. In Figure 5(a) the transition is not able to fire because there is no token in place B even though there is a token in place A. In Figure 5(b) the transition is able to fire because there are tokens at both places A and B. What distinguishes a read edge from a normal edge is what happens when the transition fires. A read edge is so named because it only “reads” if there is a token present on the place to which it is

connected and does not cause the token at this place to be removed if the transition to which the read edge is connected happens to fire. For example, if the transition in Figure 5(b) fires it results in the state shown in Figure 5(c). As expected, a new token has been produced in place C. Also as expected, the token has been removed from place A (because place A is connected to the transition by a normal edge). Notice, however, that the token from place B has *not* been removed (because place B is connected to the transition by a read edge).

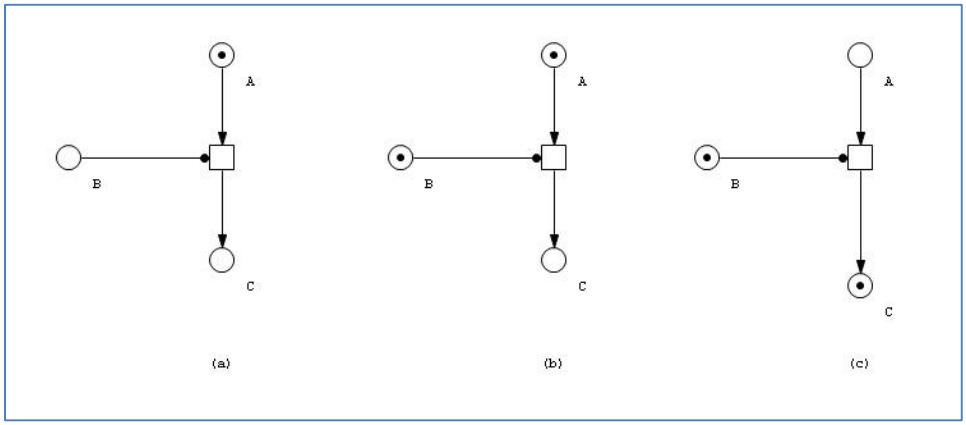

Figure 5: Read Edges

Intuitively, inhibitor edges are the opposite of normal and read edges. Normal edges and read edges represent when a given condition holds. The presence of a token at a place signifies that the condition associated with the place holds as long as a token is present. A transition having several normal or read edges as its inputs will fire when all of its input places have tokens present. Intuitively, this is a form of "and" logic in the sense that the transition fires when all of its input conditions hold. Inhibitor edges are a form of "not" logic in the sense that the presence of a token on the place to which it is connected inhibits or prevents the firing of the transition to which it is connected.

Figure 6 shows the use of an inhibitor edge. Notice that the inhibitor edge is drawn as an arc (a solid line) connecting a place to a transition where an open circle (a white dot) is drawn at the point where the arc meets the transition. There are two inhibitor edges in Figure 6 both drawn from place B to the single transition in each of Figures 6(a) and 6(b).

The transition in Figure 6(a) is able to fire because there is a token at place A (place A is connected to the transition by a normal edge) and there is no token at place B (place B is connected to the transition by an inhibitor edge). The transition in Figure 6(b) is not able to fire because there is a token at place A (place A is connected to the transition by a normal edge) and there is a token at place B (place B is connected to the transition by an inhibitor edge).

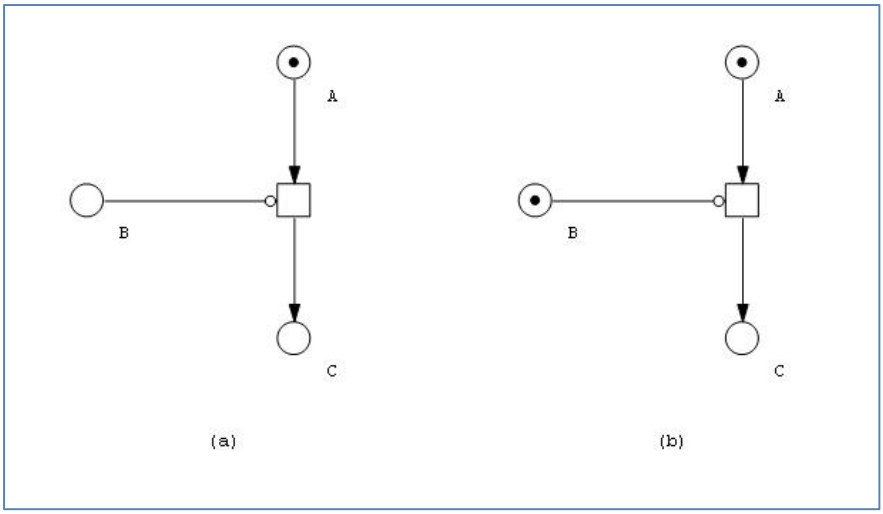

Figure 6: Inhibitor Edges

Example: Mutual Exclusion

It is often necessary for concurrent activities to ensure that they are both not performing some action on a shared resource at the same time. That is, the concurrent activities must mutually exclude each other from performing the action. There are many real world examples of mutual exclusion. Cars at a four way stop intersection must mutually exclude each other from the use of the intersection in order to prevent accidents that could occur by having multiple cars trying to pass through the intersection simultaneously. Users of a vending machine must mutually exclude each other. Interleaving the actions of two or more customers (depositing coins, making selections, taking the dispensed goods, taking any change) is unlikely to result in a state where the customers are satisfied.

The Petri net in Figure 7 models the mutual exclusion among two cars using an intersection. In this figure, each car modeled as being in one of four states: waiting at the intersection, using the intersecting, leaving the intersection, and driving. In the figure, these states are abbreviated as At, Use, Done, and Drive. The two cars continually drive and then return to the intersection and attempt to pass through. The tokens in Figure 7 represent a state of the system when both cars are driving.

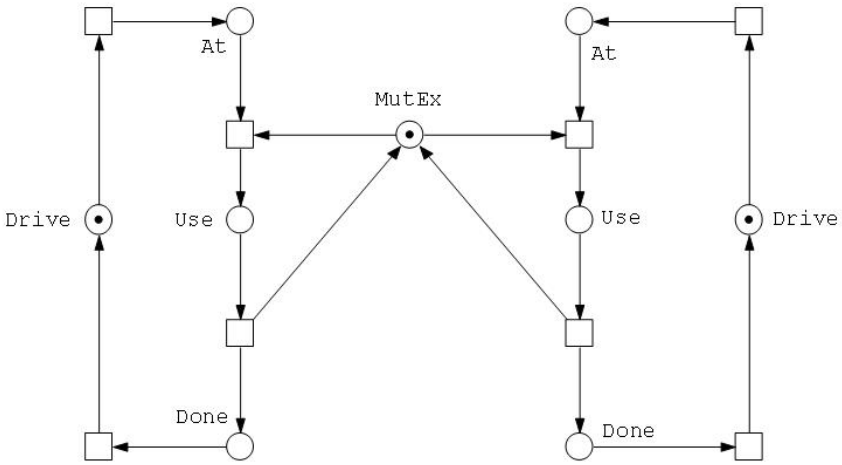

Figure 7: Mutual Exclusion

To provide the guarantee of mutual exclusion an additional place along with other edges are also part of the system shown in Figure 7. This additional place, named MutEx, has a single token. Consider the case when both cars have arrived at the intersection. That is the transition between the Drive and At places in each car fires with the result that each car has a single token in its At place. Notice that the transition between the At and Use states can only fire when there is a toke in both the At and MutEx places. Whichever one of these transitions fires causes the token in the MutEx state to be removed. The removal of the token from the MutEx place means that the transition for the other car cannot fire now. In this way, mutual exclusion is brought about. When the car using the intersection is done it must be sure that the other car (if waiting) is allowed to proceed through the intersection. This is accomplished in the Petri net in Figure 7 by the edges that lead into the MutEx place. When the transition between the Use and Done places fires a token is produced in both the Done place and the MutEx place. By recreating a token in the MutEx place, the intersection is again available to the other car.

In addition to mutual exclusion the Petri net model in Figure 7 has two other desirable properties. The first desirable property is *fairness*. This means that neither car is disadvantaged in attempting to use the intersection. When both car are at the intersection (i.e., there is a token in each of the two At places), either one might be the one that is allowed to use the intersection. Fairness would not be achieved if the car on the left was always allowed to proceed first. The second desirable property is that the solution is *non-blocking*. This means that when the intersection is not being used by one of the cars nothing blocks the other car from using the intersection.

Example: Resource Allocation

It is possible that a single place may contain multiple tokens at one time. This situation occurs frequently in dealing with resource allocation problems where there are multiple units of a given resource to allocate. In these problems, a place typically represents the number of available units of the resource. The specific number of units available at a given time is denoted by the number of tokens contained in the place. When there are no token in the place, meaning that there are no available units, activities

which need a unit of the resource to execute must wait until a unit is returned or produced. In some systems there are a fixed number of units which are acquired and returned by the activities. For example, several processes may acquire and release a printer during their execution. In other cases, the number of units is variable. For example, in distributed system a sender may generate many data packets that are waiting to be read by the receiver. Finally, the number of available units, though variable, may be limited to a maximum amount. For example, in a distributed system the number of unread data packets may be limited to some number so that the amount of buffer space at the receiver is limited.

The classical producer-consumer problem is a resource allocation problem with a variable number of resources that are limited to a maximum number. There is a producer that generates new units and makes them available to a consumer. The consumer takes one unit of the resource at a time. The primary synchronization constraints are:

- *overflow*: the producer cannot produce a new unit unless the number of units is below the maximum number allowed, and
- *underflow*: the consumer cannot take a unit unless there is at least one available.

The Petri net shown in Figure 8 is a model of a producer-consumer system where the maximum number of units is limited to 3. In this model, there are two places that are used to represent the number of units produced but not yet consumed and the number of additional units that can be produced. These places are named Full and Empty, respectively. The names full and empty reflect that the units are often contained in a fixed sized buffer of size N, where N is the maximum number of units allowed. The number of buffer entries that are full contain produced units that are available to the consumer and the number of buffer entries that are empty contain spaces available to the producer to store new units. An invariant in this model is that number(full) + number(empty) = N. The buffer is modeled in Figure 8 by the two places in the middle named Empty and Full. The three tokens in the Empty place represent the initial state of the system with three empty buffer elements. The absence of tokens in the Full place represents the initial state of the system where there are no buffer elements with information.

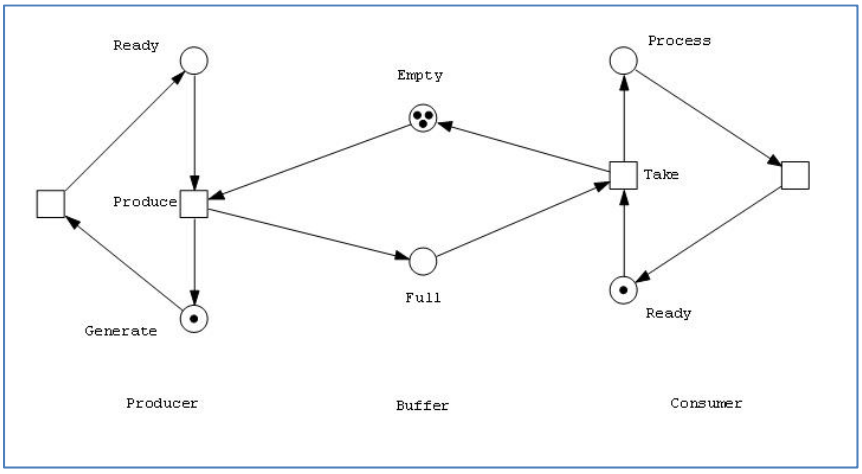

Figure 8: Producer-Consumer System

The producer in Figure 8 is modeled as a subsystem with two places. The Generate place represents the condition of the producer when it is generating the next unit of information to transmit to the consumer. The Ready state represents the condition where the producer is ready to insert the newly generated information into the buffer where it is will be available to the consumer. Notice that the transition Produce for the producer can only fire when the producer is Ready and there is at least one token in the Empty place (denoting a currently empty buffer element into which the new information can be placed).

The consumer in Figure 8 is modeled as a subsystem with two places. The Ready place represents the condition where the consumer is ready to receive the next unit of new information that was generated by the producer. Notice that the transition Take for the producer can only fire when the producer is Ready and there is at least one token in the Full place (denoting a buffer element containing new information which can be retrieved). The Process place in the producer represents the condition of the consumer when it is processing the new information most recently retrieved from the buffer.

It can be observed that the producer-consumer system in Figure 8 satisfies the two primary synchronization constraints noted above. The overflow constraint is satisfied because the producer cannot fire its Produce transition unless there is at least one token in the Empty place. Thus, it is not possible for the Produce transition to fire four (or more) times in a row without the Take transition firing one or more times. The underflow constraint is satisfied because the consumer cannot fire its Take transition unless there is at least one token in the Full place. Thus, it is not possible for the Take transition to fire four (or more) times in a row without the Produce transition firing one or more times.