5 Stochastic Petri Nets

In this lecture note we consider an important class of high level performance modelling paradigms—stochastic extensions of *Petri nets*. These are Petri net formalisms into which random variables have been added to represent the duration of activities, or the delay until events. The basic extension, *Stochastic Petri Nets* (SPN), present a very straightforward mapping between events in the SPN model and events in the underlying Markov process. As in the Markov process, a delay, represented by a random variable, is associated with every event in the model. This straightforward mapping has the advantage that generating the Markov process from any SPN model is simple and easy to implement. The disadvantage is that models developed in this way tend to result in Markov processes which have a large number of states.

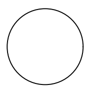
We will go on to consider *Generalised Stochastic Petri Nets* (GSPN). The generalisation involves adding some simple additional features to the set of modelling primitives provided for the modeller. As a result the mapping between events in the model and events in the underlying Markov process is more sophisticated. However the resulting state space is usually more compact, and often a closer representation of the behaviour of the system can be achieved.

Although Petri nets have been used for qualitative modelling of computer and communication systems since the 1960s, their use as a performance modelling paradigm started about twenty years ago. In particular they were found to be especially useful for modelling distributed and parallel systems. Such systems were difficult to model with queueing networks, which were the prevailing performance modelling paradigm at the time.

5.1 Petri Nets

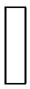
Petri nets provide a graphical notation for the formal description of the dynamic behaviour of systems. They are particularly well suited to systems which exhibit concurrency, synchronisation, mutual exclusion and conflict.

The primitives of the notation are the following:



PLACES

Places are used to represent conditions or local system states, e.g. a place may relate to one phase in the behaviour of a particular component.



TRANSITIONS

Transitions are used to describe events that occur in the system; these will usually result in a modification to the system state. The occurrence of the event in the system is represented by the *firing* of the corresponding transition in the Petri net.

TOKENS

Tokens are identity-less markers that reside in places. The presence of a token in a place indicates that the corresponding condition or local state holds.

ARCS

Arcs specify the relationships between local states or conditions (places) and events (transitions). An arc *from* a place *to* a transition is termed an *input arc*. This indicates the local state in which the event can occur. An arc *to* a place *from* a transition is termed an *output arc*. This indicates the local transformations which will be induced by the event.

Tokens move between places according to the firing rules imposed by the transitions. A transition can *fire* when each of the places connected to it has at least one token; when it fires, the transition removes a token from each of these places and deposits a token in each of the places it is connected to. This is called the *firing rule*.

Sometimes a transition will require an input place to contain two or more tokens before it can fire. In this case, rather than draw more than one arc between the place and the transition, we denote the *multiplicity* of the arc by a small number written next to the arc. Similarly for output arcs.

The state of the system combines information about all the local states. Since each local state is represented by the number of tokens present in a particular place, the state of the system is represented by a tuple, with one entry for each place, and the value of the entries denoting the number of tokens in that place. This is termed the *marking* of the net.

A Petri net consisting of places and transitions linked by arcs is incomplete if it does not also have tokens in some places. This initial placing of tokens is called the *initial marking*—this represents the starting state of the system.

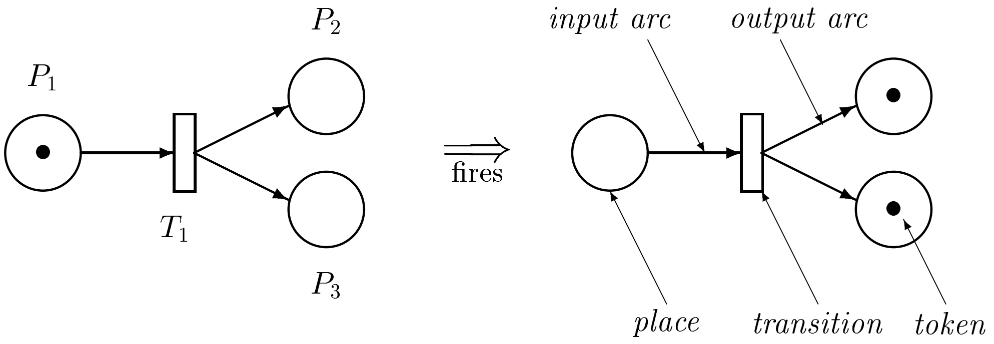


Figure 3: A Simple Petri Net Firing

Starting from an initial marking and following the firing rule we can progress through the states of the model. This is sometimes called *playing the token game*. Continuing in this way, recording all the states we see and stopping only when we can reach no states that we have not already seen, we obtain all the possible states of the model. This is called the *reachability set*; it is the set of all possible markings that a net may exhibit, starting from the initial marking and following the firing rules. Different initial markings might lead to different reachability sets as we will see in the example below. This is why the initial marking is an important part of the model definition.

If, when playing the token game, as well as all the states we come across, we record the transitions between those states, we obtain the *reachability graph*. This is a graph in which the nodes are the reachable markings and the arcs between nodes represent the possible transition firings which may move the model from one marking to the other.

Example: For the simple Petri net shown in Figure 3 the initial marking is $(1, 0, 0)$ and the final marking is $(0, 1, 1)$. These are the only possible markings. If the initial marking is changed to $(3, 2, 1)$ the set of reachable markings is: $\{(3, 2, 1),\ (2, 3, 2),\ (1, 4, 3),\ (0, 5, 4)\}$.
Exercise: Draw the reachability graph for this model with the second initial marking.

5.2 Stochastic Petri Nets

Recall that if we wish to extract timing information from a model we must represent timing information about the system in the model when it is constructed. In the case of Petri nets there has been a variety of suggestions of how to introduce timing information into Petri net notation.

If we consider the reachability graph of a Petri net model it resembles the state transition diagram of a Markov process. *Stochastic Petri Nets* (SPN) formalise this intuitive correspondence. Given a Petri net model (complete with initial marking):

- we associate a state in the Markov process with every marking in the reachability graph of the Petri net;
- we associate an event, or transition, in the Markov process with each firing of a transition in the Petri net which causes the corresponding change of marking.

Since an exponentially distributed delay is associated with the delay until each event in a Markov process, and transitions in the Petri net correspond to events, in an SPN model an exponentially distributed delay is associated with each transition in the net structure. Thus each transition in an SPN has a *firing rate* which is the parameter of the corresponding exponential distribution, and transitions are sometimes termed *timed transitions*.

How this works in practice is best illustrated by an example.

Example: Consider again the simple multi-processor system which was described in Section 3.3.1. We make a slight modification to the system: we now explicitly represent the processor requesting and gaining access to the common memory. In the previous model we used a higher level of abstraction in which this action was ignored. A processor executes locally for some time (mean duration $1/\lambda$), and then requests access to common memory (gaining access has mean duration $1/r$). Once it has gained access, the duration of common memory access is assumed to be $1/\mu$ on average. We can model the behaviour of a single processor interacting with the common memory using an SPN, as shown in Figure 4. In this SPN

- place P_1 represents the local state of the processor when it is executing privately;
- place P_2 represents the local state of the processor when it is ready to commence access to the common memory;
- place P_3 represents the state when the process is using the common memory;
- place P_4 represents the local state of the common memory when it is not in use;
- transition T_1 represents the *action* of the processor *executing privately*; the rate of this transition is λ;

- transition T_2 represents the processor gaining access to the common memory; the rate of this transition is r;
- transition T_3 represents the processor accessing the common memory; the rate of this transition is μ.

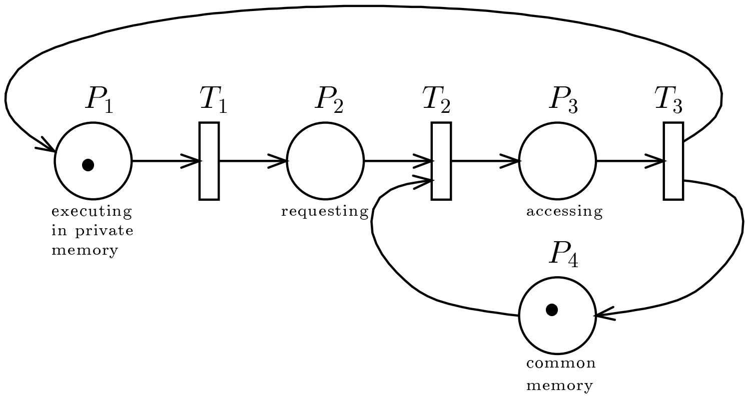
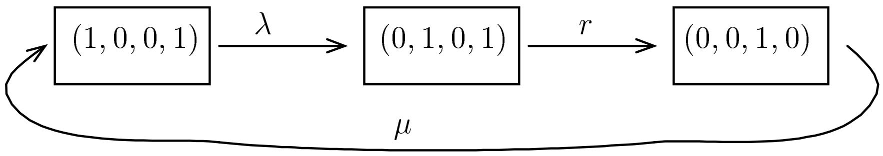


Figure 4: Stochastic Petri net representing a single processor in a shared memory system

The reachability set of this model is $\{(1,0,0,1),(0,1,0,1),(0,0,1,0)\}$, and the reachability graph is



This is also the state transition diagram of the underlying Markov process and thus we obtain the generator matrix:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 \\ 0 & -r & r \\ \mu & 0 & -\mu \end{pmatrix}$$

Solving this we obtain the steady state probability

$$\boldsymbol{\pi} = \left(\frac{\mu r}{\lambda r + \mu\lambda + \mu r}, \frac{\mu\lambda}{\lambda r + \mu\lambda + \mu r}, \frac{\lambda r}{\lambda r + \mu\lambda + \mu r} \right)$$

If we consider the two-processor system, as we did in Section 3.3.2, the SPN model is as shown in Figure 5. Now we see why the action of gaining access needs to be explicitly represented in the SPN model. Without it we could not enforce the necessary mutual exclusion between the processors. If we assume the upper subnet represents processor A and the lower subnet represents processor B, the interpretations for transitions should be clear: T_1 and T_4 correspond to T_1 in the single processor model, T_2 and T_5 correspond to T_2, and T_3 and T_6 correspond to T_3. Similarly for the places. The rates of the transitions

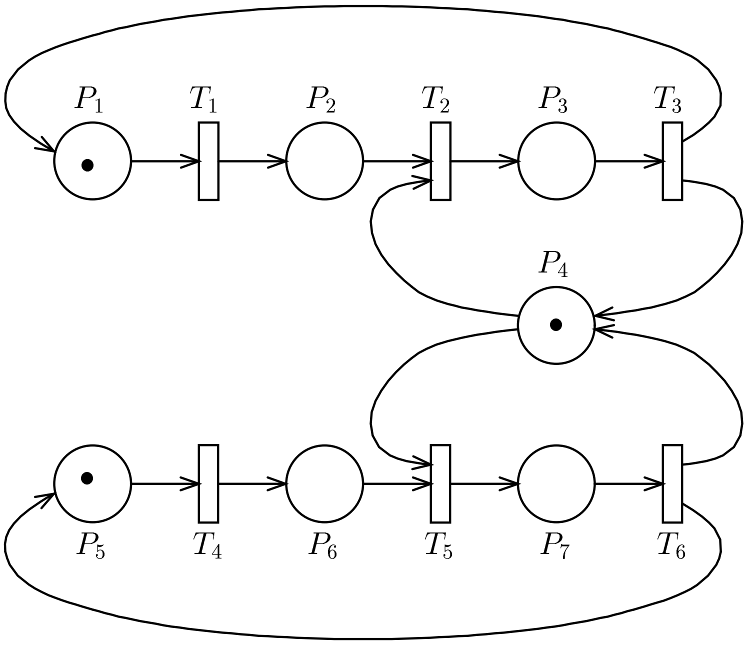


Figure 5: Stochastic Petri net representing a two-processor shared memory system

will be λ_A, r_A and μ_A for T_1, T_2 and T_3 respectively, and λ_B, r_B and μ_B for the transitions T_4, T_5 and T_6.

The reachability set is now[1]

$$\left\{ \begin{array}{llll} (1,0,0,1,1,0,0), & (1,0,0,1,0,1,0), & (1,0,0,0,0,0,1), & (0,1,0,1,1,0,0), \\ (0,1,0,1,0,1,0), & (0,1,0,0,0,0,1), & (0,0,1,0,1,0,0), & (0,0,1,0,0,1,0) \end{array} \right\}$$

Exercise: Draw the reachability graph (state transition diagram) for this model, and construct the generator matrix.

5.3 Generalised Stochastic Petri Nets

Although SPN provide a clear and intuitive formalism for generating Markov processes they do have the disadvantage that the models constructed in this way can soon become exceedingly large. One of the reasons for this is that actions which would not be explicitly represented if we were working directly at the Markov process level have to be represented by transitions. This has the effect of increasing the state space since we now have to consider the state of waiting for these actions to occur, whereas we would prefer to abstract away from these actions.

An example of this is the action of gaining access to the common memory in two-processor model of the previous section. When we modelled this system as a Markov process in Section 4.2.2 we did not represent this action because we constructed the state space directly and so could prohibit the state of both processors accessing the common

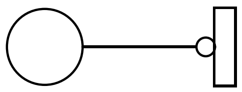
[1] Note that we now have eight distinct states as opposed to the five states we had in the Markov process when we modelled the system directly (Lecture Note 3).

memory at once. One of the disadvantages of using a high level modelling paradigm, such as SPN, to generate the state space automatically, is that we sometimes have to introduce mechanisms into the model to make sure that only "genuine" states are generated.

Generalised Stochastic Petri Nets (GSPN) represent an extension of the SPN formalism, which is designed to address this problem. Two new primitives are added to the notation, all others remaining the same and keeping the same interpretation. These new primitives are *immediate transitions* and *inhibitor arcs*.

IMMEDIATE TRANSITIONS

Immediate transitions are represented by a single bar, instead of the rectangle used to represent timed transitions. Immediate transitions are used to describe events which are assumed to take no time. When they are enabled in a model they fire immediately, taking precedence over any enabled timed transitions. In other respects the firing rule is exactly the same as for timed transitions. If two or more immediate transitions can be enabled at the same time, the probability that each of them is the one to fire must be declared in the model.



INHIBITOR ARCS

An inhibitor arc is used to indicate when a local state or condition *disables* a transition, rather than *enables* it. An inhibitor arc *from* a place *to* a transition, indicates that the transition cannot fire if there is a token in the place; it can fire when there is no token in the place if the places connected to its input arcs do contain tokens. In other words, the usual firing rule still applies; the inhibitor arcs impose an additional constraint to that rule. Inhibitor arcs may have multiplicities in the same way as ordinary arcs.

The types of events represented by immediate actions usually fall into two categories: *control actions* and *logical actions*. Control actions are ones which are necessary to ensure the correct behaviour of the model but which take negligible time to be executed, which means that they are unlikely to affect the performance of the system. For example, gaining access to the common memory in the two-processor model considered in the previous section can be regarded as a control action. The GSPN representing the system in this way is shown in Figure 6. In these cases immediate actions provide an additional tool for abstraction within the model.

Logical actions arise when the system makes a choice between two or more alternatives. In an SPN these actions must be represented as actions which take time. However it is more natural to think of them occurring instantaneously. In the GSPN a choice between two alternatives will be represented by two immediate transitions which have exactly the same input places (firing conditions), and the probabilities assigned to them will reflect the relative probabilities of the two choices.

5.3.1 Generating the Markov process from a GSPN model

One of the advantages of SPN models was the straightforward correspondence between the reachability graph of the SPN and the state transition diagram of the Markov process

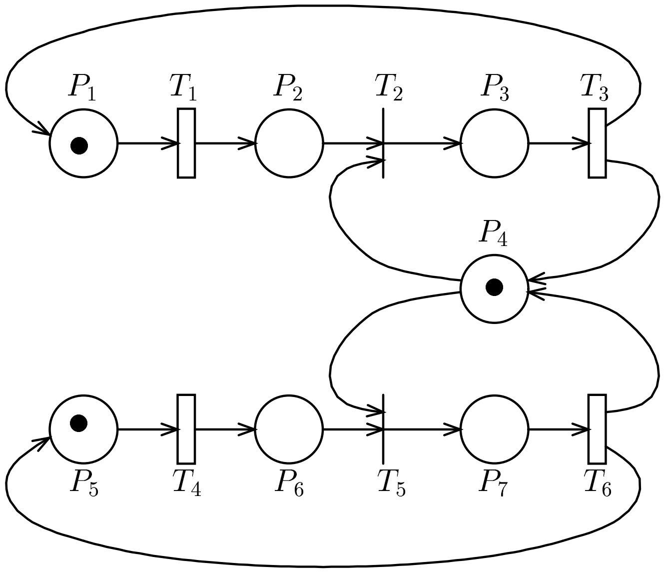


Figure 6: GSPN representing a two-processor shared memory system

it generated. The inclusion of inhibitor arcs within the GSPN notation does not affect this correspondence. The effect of the inhibitor arc will be to eliminate some arcs which might have been possible in the reachability graph. Thus the relationship between the Markov process and the reachability graph is unaffected. In contrast, the introduction of immediate actions has serious implications for the Markov process.

If there is an immediate transition in a GSPN model, in the reachability graph there will be arcs between nodes which represent immediate transitions between markings in the GSPN. Such arcs are labelled with rate ∞. In other words, in the reachability graph some transitions between states take place instantaneously, without an associated delay. Such instantaneous transitions are not possible in a Markov process. Therefore these transitions, and the states that give rise to them, must be eliminated from the reachability graph before the Markov process is generated.

Formally the transitions of a GSPN can be partitioned into two subsets—timed transitions and immediate transitions. Since immediate transitions always take precedence over timed transitions, we can also partition the marking of a GSPN into those which only enable timed transitions, and those which enable at least one immediate transition. The latter are called *vanishing markings*, since the model will always move on to another marking instantaneously. In contrast, the markings which enable only timed transitions are called *tangible markings*. It is the vanishing markings which must be eliminated from the reachability graph before the Markov process is generated. In the simple examples we will consider by hand it will be clear how this is done; for more complex GSPN models the elimination will be carried out automatically by software, as explained in Lecture Note 7.

Exercise: Draw the reachability graph of the GSPN model of the two-processor system and identify the vanishing markings and the tangible markings.

Jane Hillston ⟨jeh@inf.ed.ac.uk⟩. September 19, 2003.