6 More about GSPN Models

In this note we will consider two simple systems modelled by GSPN and in the course of doing so examine more closely the dynamics of these models with respect to timed and immediate transitions. At the end of the note we will summarise some of the key features of the `SPNP` modelling tool and describe how it may be used to generate and solve GSPN models.

6.1 System Dynamics

Recall that in an untimed Petri net the firing of a transition corresponded to an event in the system. In GSPN we recognised that these events might arise in two different ways:

- it may be induced by the completion of some activity; these activities can be further divided into two categories:
  - those which take significant time to complete.
  - those whose duration is negligible compared with other actions in the system, often called *control actions*.
- it may result from the verification of some logical condition of the system.

Timed transitions represent actions which take some time to be completed. Immediate transitions represent logical events and control actions.

For timed transitions, when they become enabled, i.e. all the input places of the transition become marked with the necessary number of tokens, we imagine the corresponding activity starting in the system. This activity will continue until it is completed or interrupted. We assume that each transition has a timer or local clock associated with it: when the transition is enabled the timer is set to a value. Since we are using a random variable to represent activity durations, this value is *sampled*[1] from the exponential distribution with the appropriate rate parameter. While the transition is enabled we assume that the timer decrements at a constant speed. When the timer reaches zero the transition fires.

For immediate transitions, the firing of the transition takes no time and is assumed to occur instantaneously. We also assume that enabled immediate transitions always have priority over any enabled timed transitions, so we fire all possible immediate transitions first. If only one immediate transition is enabled, it fires, and the new marking is produced. If several immediate transitions are enabled, some mechanism is necessary to identify which transition will fire. In fact the choice of transition to be fired is only important in cases where there is conflict between the enabled transitions, i.e. the firing of one will disable another. If the enabled immediate transitions are concurrent, they can be fired in any order. When enabled immediate transitions are in conflict GSPNs associate *weights*, or relative probabilities, with each transition. We can think of a distribution being sampled to determine which transition actually fires in each case.

[1] *Sampling* means randomly choosing a value, in such a way that if you kept on doing this over time and kept track of the values they would satisfy the probability distribution function. Sampling will be treated in more detail when we study simulation. Note that for GSPN models solved as a Markov process this is just a way of thinking about the dynamics of the system, we do not actually sample the distributions.

Bearing this view of the system dynamics in mind we will consider again the two-processor shared memory system from lecture note 5. For convenience it is shown again in Figure 7.

6.1.1 Example: The multiprocessor system revisited

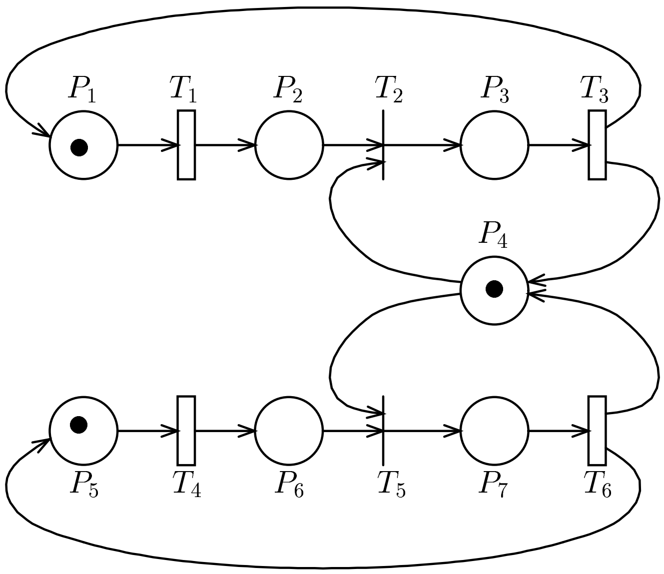

Figure 7: GSPN representing the two-processor shared memory system

In the initial marking of the system we see that both transitions T_1 and T_4 are enabled. Thus we think of them sampling their respective exponential distributions, with parameters λ_A and λ_B, and obtaining values V_A and V_B say. Which transition fires first will be determined by whose associated timer reaches the value zero first—this is called the *race policy*. If $V_A < V_B$, transition T_1 will be the one to fire and the marking changes as $(1, 0, 0, 1, 1, 0, 0) \rightarrow (0, 1, 0, 1, 1, 0, 0)$. If $V_B < V_A$, transition T_4 fires and the new marking is $(1, 0, 0, 1, 0, 1, 0)$. The case of $V_A = V_B$ can only occur with probability zero[2] so we do not need to worry about it. Of course each time the distributions are sampled different values may be obtained, and so which transition fires first will vary.

In either case we need to consider what happens to the remaining time on the timer associated with the transition which did not fire. There seem to be two possibilities: either it is forgotten or it is remembered. For example, in the two-processor system if $V_A < V_B$ and transition T_1 fires, the time remaining on the timer associated with T_4 will be $V_B - V_A$. Since T_4 remains enabled after T_1 has fired it would seem unreasonable for time the system has already spent on the corresponding activity to be forgotten. In contrast, if T_1 and T_4 had been in conflict, so that the firing of T_1 removed the enabling tokens for T_4 it might seem reasonable to forget the time that has already been spent on the corresponding activity, and imagine the system starting that activity again from the

[2]The probability that a sample extracted from an exponential distribution takes any specific value v is always zero, so if we know that the value sampled by the first timer is V_A the probability that the second timer will sample the same value is zero.

beginning whenever T_4 next becomes enabled. Note that in the case of conflict between timed transitions which transition will fire is always resolved by the race policy.

Fortunately, the memoryless characteristics of the exponential distribution mean that we do not need to differentiate between the situations when a timed transition should forget its previous work and those when it should remember. Indeed, whether the timer is reset or not, the probability distribution of the time remaining until the firing of a transition will always be the same, and distributed according to the exponential distribution when the transition was first enabled. Thus we can model either of these situations in the same way in a GSPN.

Despite initial impressions from inspecting the net, the two immediate transitions in the GSPN of the two-processor system can never both be enabled at the same time. Thus at most one immediate transition is enabled in any marking and this transition will fire instantaneously leading to the following marking. For example, if T_1 fires as described above, leading to the marking $(0, 1, 0, 1, 1, 0, 0)$, transition T_2 is now enabled and fires at once, leading to the marking $(0, 0, 1, 0, 1, 0, 0)$.

6.1.2 Example: The reader-writer system

The GSPN in Figure 8 is a representation of what is sometimes referred to as the *reader-writer system*. The model represents a system in which there is a set of processes who share access to a common database. On any particular access a process may wish to perform a read or a write. Any number of readers may access the database concurrently; in contrast, a writer requires exclusive access to the database. In between accesses to the database each process will undertake some processing independently.

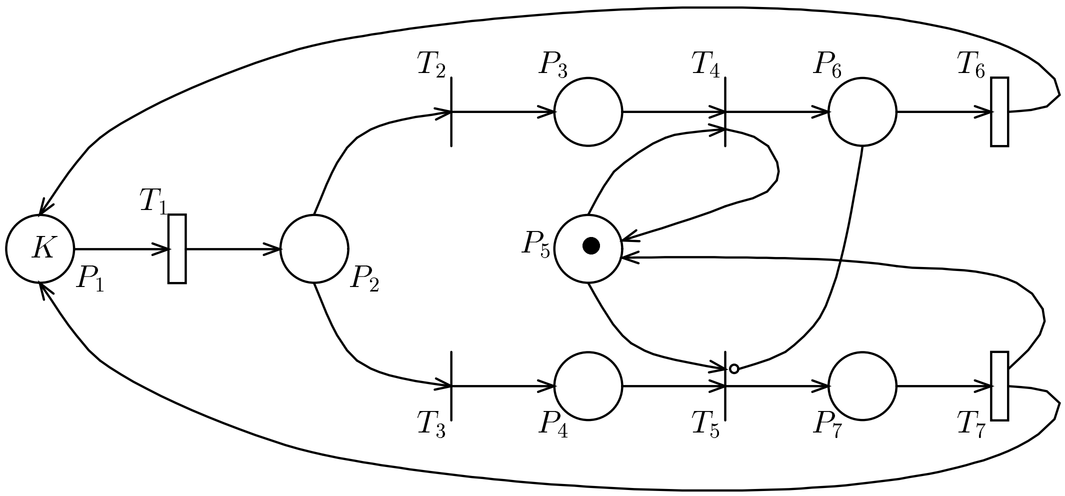

Figure 8: GSPN representing the simple reader-writer system

Timed transitions are used to represent the reading and writing actions as well as the local activities performed by the processes between two subsequent accesses to the database. Immediate transitions are used to implement the decision of which action a process wants to perform on the database (T_2 and T_3) and to implement the scheduling between reading and writing activities (T_4 and T_5). The interpretations of the places and transitions in the model shown in Figure 8 are given below.

- P_1 represents a process undertaking local processing.
- P_2 represents a process ready to commence access to the database.
- P_3 and P_4 model processes ready to read and ready to write respectively.
- P_5 enforces unique access to the database for writing; it can be thought of as representing the condition *no write access is in progress*.
- P_6 and P_7 represent processes currently reading or currently writing respectively.
- T_1 models the independent processing, not related to the database access; each process undertakes local processing for a mean duration of $1/\lambda$ milliseconds.
- T_2 and T_3 model the "decision" of the process to make a read access or a write access; the switching probabilities are chosen to reflect the relative frequencies of these types of access.
- T_4 represents a process gaining access to the database to start a read access.
- T_5 represents a process gaining access to the database to start a write access; note that the transition is inhibited whenever a read is already in progress (P_6).
- T_6 models the database read activity; the average read access lasts for $1/r$ milliseconds.
- T_7 models the database write activity; the average write access lasts for $1/w$ milliseconds.

This system is commonly used in texts about modelling because it is small and easily understood but yet it exemplifies three interesting features which are often important in models: *concurrency of events* (two or more processes may be concurrently accessing the database for reading), *mutual exclusion* (only one process at a time may access the database for writing) and *choice* (an access can either be a read or a write). This version of the model is based on the one presented in the book by Ajmone Marsan *et al.*[3]; in other books variations on this model appear.

Note that the initial marking of the GSPN has a single token in place P_5, representing that the condition is currently satisfied, and a parameter K in place P_1. The *parametric marking* represents the situation that there is some number of processes, K, within the system, but the value of K has not yet been decided. A value must be assigned to K before the model can be solved because, as we have already seen, the number of tokens in the initial marking can drastically change the reachability set of a model.

Let us first consider the immediate transitions of the model. Whenever there is a token in place P_2 the two immediate transitions T_2 and T_3 are both enabled. These transitions are in conflict because the firing of one will disable the other. Thus weights, or relative probabilities, must be assigned to the transitions to resolve the conflict. If we know that 60% of database accesses are read accesses and 40% are write accesses, we associate weights 0.6 and 0.4 with T_2 and T_4 respectively.

[3] *Modelling with Generalized Stochastic Petri Nets*, M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis, John Wiley, Series in Parallel Computing, 1995

If transition T_2 fires, and P_5 contains a token, transition T_4 is now enabled and fires immediately—no other immediate transitions are enabled and the immediate transition takes priority over any timed transitions which might be enabled, such as T_1. Thus two immediate transitions may fire one after another without any time progressing in our model. If T_3 fires a token appears in place P_4, but transition T_5 will only be enabled if there is no token in place P_6—this is the effect of the inhibitor arc—and there is a token in place P_5.

Let us now consider the timed transitions of the model. If there are K tokens in place P_1 it means that there are K processes currently involved in local processing, each progressing at a rate λ. Thus the total rate at which processes complete local processing is $K \times \lambda$. In the model we represent this by saying that the rate of the transition T_1 is dependent on the marking of transition P_1: the rate of the transition is λ multiplied by the number of tokens in P_1 (by the superposition principle). Such transitions are said to have *marking dependent firing rate.*

Similarly when more than one process is engaged in a read access, the rate at which a single read access is completed will depend on the number of processes currently reading—the marking of place P_6—and the rate at which reading occurs—the transition rate of transitions T_6 (r).

In general, whenever a transition's enabling condition is satisfied more than once, e.g. K tokens in place P_1 when only one is required to enable transition T_1, there are two possibilities with respect to the firing rate of the transition. We can imagine the transition progressing each set of enabling input tokens *serially.* In this case we only consider the transition to be enabled or not, and when it is enabled it always works at the same rate. This is called *single server semantics* and it is the default case for GSPN models. Alternatively we can think of the transition progressing each complete set of enabling input tokens *concurrently.* In this case the rate at which the transition works will depend upon its *enabling degree.* This is called *infinite server semantics* and in GSPN models it is achieved by setting a marking dependent firing rate.

In the reader-writer model the enabling degree of the transition T_1 in the initial marking is K. Similarly the enabling degree of transitions T_6 can have any value between 1 and K. In contrast, enabling degree of T_5 is at most 1: even though the place P_4 may have up to K tokens, the enabling degree will not be greater than 1 because there is only one token in place P_5.

6.2 The SPNP package

The Division (of Informatics) has several Petri net solution packages but the one which we use on this course is `SPNP` (Stochastic Petri Net Package). The package is composed of several C files. The GSPN to be studied must be described in a language called CSPL (C-based Stochastic Petri Net Language). This language consists of various predefined functions which can be used to specify the structure of the GSPN and the desired outputs of the model. The following steps can be identified in the performance analysis of a system using a GSPN model:

1. the model must be constructed to reflect the behaviour of the system;

2. the performance measures of interest must be defined in terms of GSPN markings and of transition firings;
3. the reachability set and the reachability graph must then be generated, to obtain the Markov process associated with the GSPN;
4. the Markov process must be solved;
5. the performance measures must be computed from the Markov process solution.

Using SPNP the modeller is responsible for the first two steps; the model and the performance measures are specified in the CSPL file. Steps 3-5 are carried out automatically by the SPNP package. The CSPL file (*filename.c*) is compiled, linked to the other files of the package, and run by typing the command

`make -f /home/jeh/SPNP/obj/Makerun SPN=`*filename*
or `make -f /home/jeh/SPNP/obj/Makerun SPN=`*pathname/filename*

The intermediate files generated by the package and the final results will be placed in the same directory as the CSPL file. Files have different extensions, according to the kind of information that they contain. If your CSPL file is named `model.c` then the following files may be generated. Some output files are optional and will only be produced if a flag in the source file is set to the appropriate value.

`model.o` obtained when compiling `model.c`

`model.out` contains some default information and the requested output (according to what is specified in `model.c` using the provided functions)

`model.spn` executable file obtained by linking the package object files together with `model.o`

`model.rg` contains the reachability graph information: composition of each marking and the transition firings between them.
(`IOP_PR_RGRAPH = VAL_YES`)

`model.mc` contains the (numerical) Markov process corresponding to the GSPN.
(`IOP_PR_MC = VAL_YES`)

`model.prb` contains the (numerical) results of the analysis of the underlying Markov process: the steady state probability for each tangible marking.
(`IOP_PR_PROB = VAL_YES`)

`model.log` contains the output messages produced by the package during model solution.

The syntax and semantics of CSPL are based on the C language: a correct CSPL file will be a correct C file too. What distinguishes CSPL from C is the set of predefined functions available for the definition of GSPN entities. Every CSPL file must specify the following functions: `parameters, net, assert, ac_init, ac_reach` and `ac_final`. We briefly describe the significance of each function below. You should look at the provided example files for more details (`/home/cs4/www/modsim/spnp-examples`).

Function `void parameters(void)` This function is used to set the various flags which control options in the package. There are two option-setting functions `iopt` or `fopt` according to whether the option being set has integer or floating point value.[4] In addition there are two input functions `input()` and `finput()` which accept integer and floating point values respectively.

Function `void net(void)` This defines the structure and initial marking of the GSPN, together with firing rates (for timed transitions) and firing probabilities (for immediate transitions). The most important predefined functions are:

`place` **and** `trans` are used to declare the names of the places and transitions used in the GSPN respectively.

`init` gives the initial number of tokens in the named place. By default, places are assumed to be empty.

`rateval` **and** `ratedep` define the transition rate for timed transitions with single-server and infinite-server semantics respectively. When a dependent rate is assumed the place on which the rate depends must be given.

`probval` **and** `priority` define the probability and priority associated with an immediate transition. SPNP assumes all timed transitions have priority level 0 and will not allow immediate transitions to be given priority levels less than 1; it will however allow higher levels as a way of enforcing an ordering between conflicting immediate transitions.

`iarc, oarc, harc` these specify input, output and inhibitor arcs respectively. In each case the transition and the place involved are named, in that order. All these arcs are assumed to have multiplicity one.

`miarc, moarc, mharc` as above, but for arcs with multiplicity greater than one. In each case the transition, place and multiplicity must be specified in that order.

Function `int assert(void)` It is often desirable to carry out some form of model validation to ensure that the model behaves correctly with respect to the system.[5] SPNP supports this in the CSPL file by allowing the modeller to make assertions about the model which are checked when the state space is generated. For example, in the reader-writer system we know that at most one process should have write access to the database at any time, i.e. the marking of place P_7 is less than or equal to 1. Similarly, we know that it should not be possible for transitions T_6 and T_7 to both be enabled.

```
assert(){
   if (mark("p7") > 1 || enabled("t6") && enabled("t7"))
     return(RES_ERROR);
   return(RES_NOERR);
}
```

[4] A full list of options and values is in the SPNP manual but you do not need to know these and can probably safely copy the parameter section of one of the example files into any CSPL program you write.

[5] We will learn more about model validation techniques later in the course.

If we make these assertions in the CSPL file (as shown above), and they are violated when the reachability set is generated, SPNP will stop the execution and report the error.

Functions `void ac_init(void)`**,** `void ac_reach(void)` **and** `void ac_final(void)`
These are called before reachability graph construction, after reachability graph construction and for solution of the Markov process, respectively. The function `ac_final` is designed to give the user flexibility with respect to the output produced by the model solution. Much of this flexibility is beyond the scope of this course. In addition this function is used to control the performance measures calculated in the model (see lecture note 7).

6.3 A GSPN model of the PC LAN with 4 nodes (8 nodes)

The model below is a GSPN representation of the PC LAN with four nodes considered in lecture note 4. Note that the inhibitor arc from place P_{i2} to transition T_{bi} (for $i = 1, \ldots, 4$) ensures that a token does not bypass a PC in which a data packet is waiting for transmission. In the CSPL file `/home/cs4/www/modsim/spnp-examples/PC-LAN.c` you will find the SPNP model of the LAN with eight PC nodes.

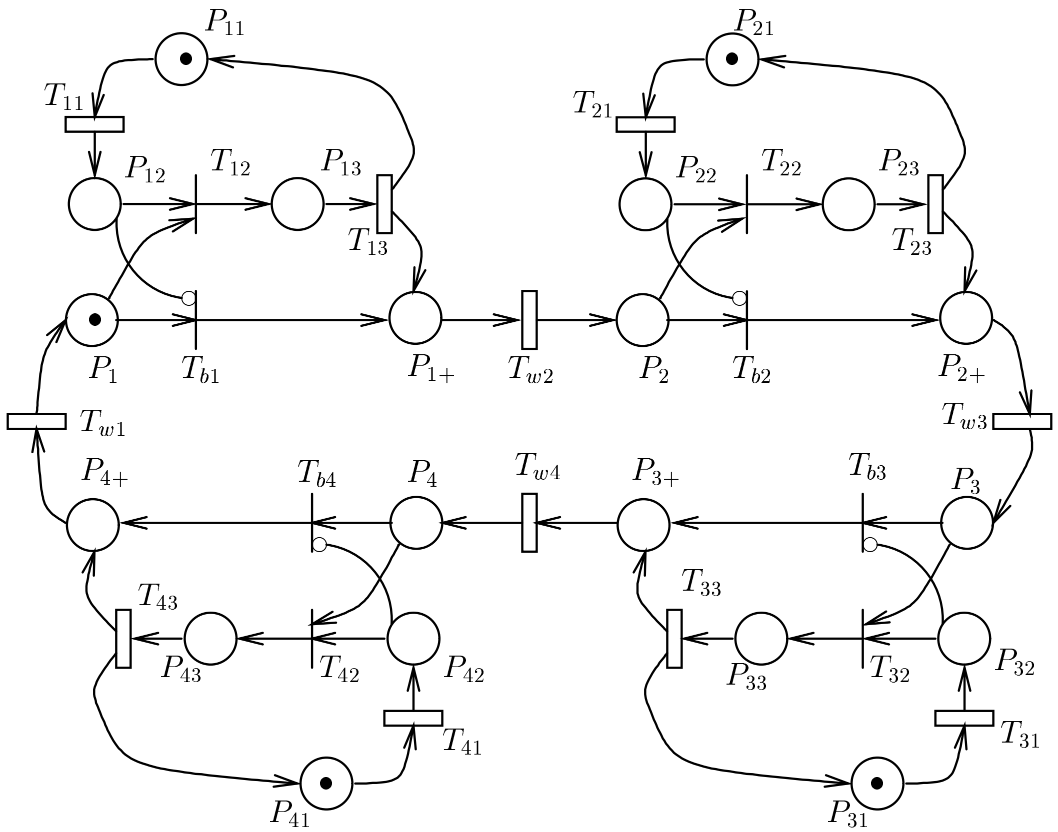

Figure 9: GSPN representing the PC LAN with four nodes