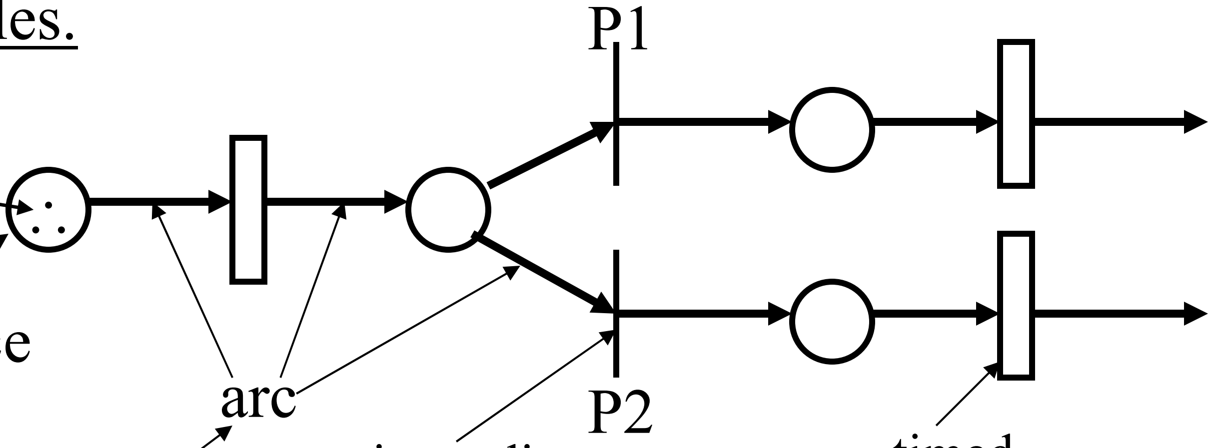
Chap 7: Stochastic Petri Net Models

* A stochastic Petri net (SPN) consists of places, transitions, arcs, tokens and a set of firing rules.



tokens representing jobs

place

arc

immediate transition

timed transition

the transition time is zero

the transition time is exponentially distributed

An arc's multiplicity can be 1 (default), >1, or a variable depending on the state of the system

Both are allowed in a generalized SPN model, or a GSPN model (SPNP is based on GSPN)

can be arbitrary distribution in an extended SPN (ESPN) model

* Firing rule: A transition is <u>enabled</u> if:

a) the # of tokens in each input place <u>without an inhibitor arc</u> is at least equal to the multiplicity of the input arc from that place.

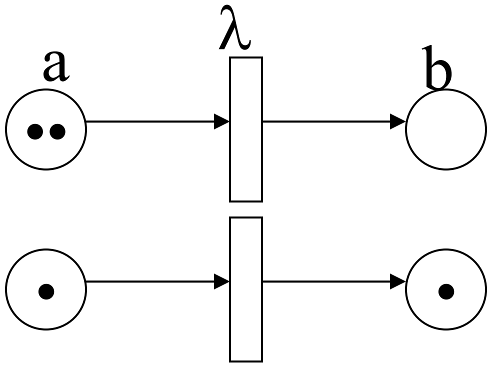
b) the # of tokens in each input place <u>with an inhibitor arc</u> is less than the multiplicity of the input inhibitor arc from that place.

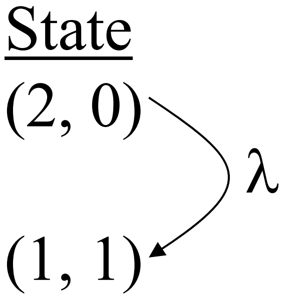
c) the enabling function of the transition (if any is assigned) returns TRUE — which is the default if not assigned.

<u>Concept of a state in SPN:</u>

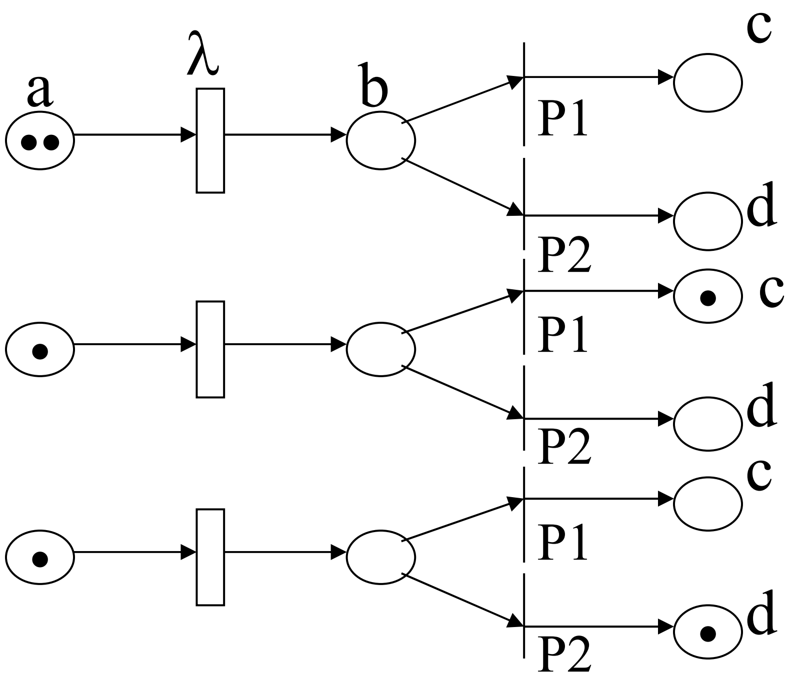
Each distinct Petri net marking (as a result of tokens being distributed to various places) constitutes a separate state in the underlying Markov model.

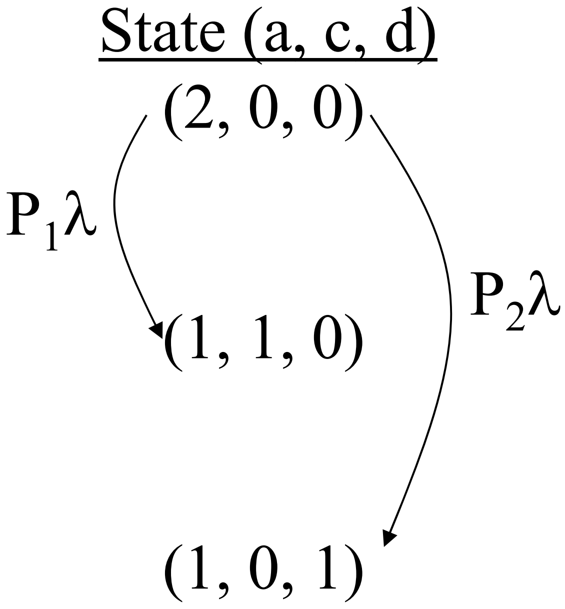
e.g.


λ
a
b

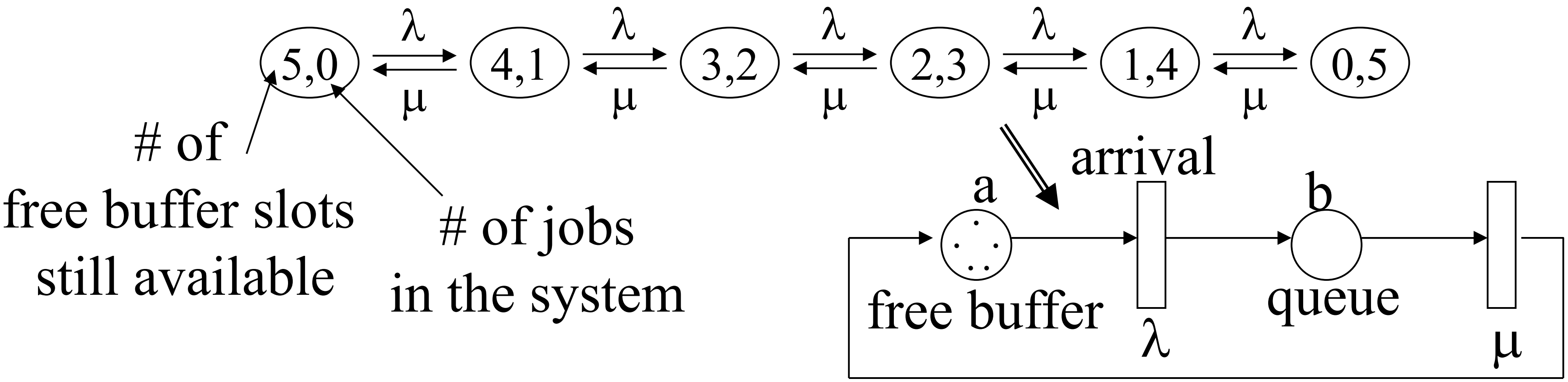

State
(2, 0)
λ
(1, 1)

e.g.


λ
a
b
c
P1
d
P2
c
P1
d
P2
c
P1
d
P2

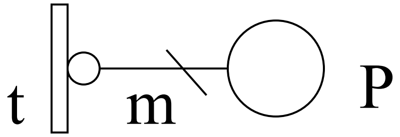

State (a, c, d)
(2, 0, 0)
P₁λ
(1, 1, 0)
P₂λ
(1, 0, 1)

Ex: draw an SPN corresponding to the following M/M/1/5 queue



---

* An inhibitor arc of multiplicity m from a place P to a transition t will disable t when P contains at least m tokens.



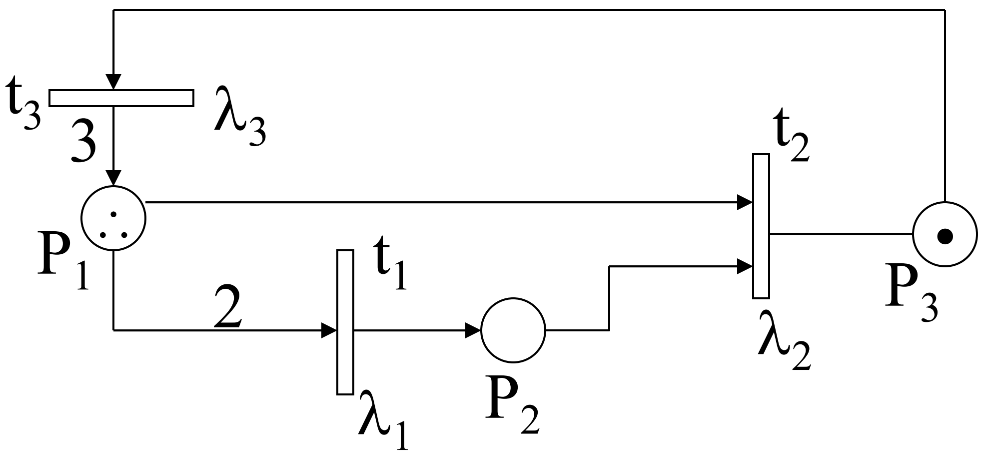
When there are many transitions enabled, the highest priority one will be fired first.

(also called a guard)

* A transition can be associated with a priority, an enabling function which can be state-dependent, and a rate function which can also be state-dependent.
* when both immediate and timed transitions are enabled in a marking, only immediate transitions will fire.

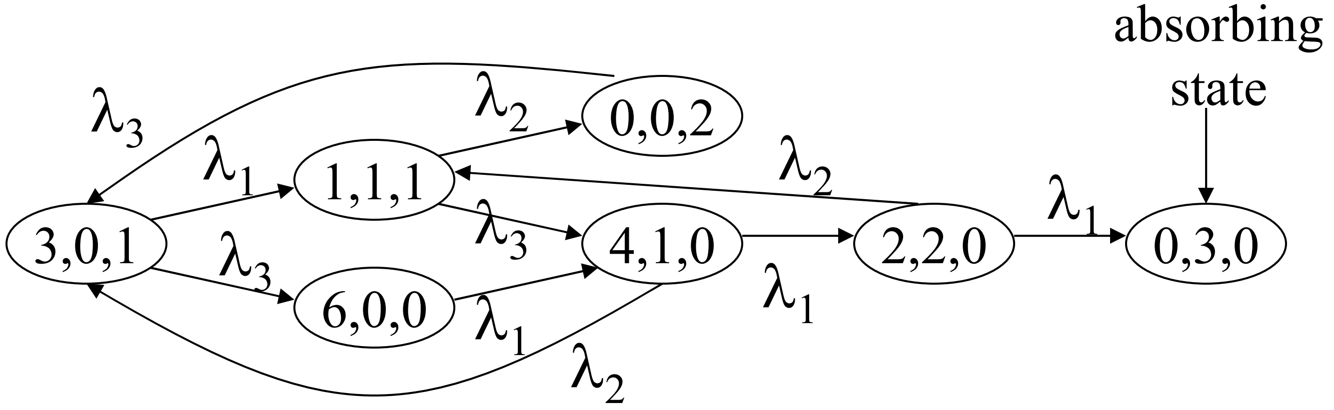
Q:

How many states will be generated based on this SPN?



* when a transition fires, the # of tokens removed in each of its input places is equal to the multiplicity of the input arc, and the # of tokens deposited in each of its output places is equal to the multiplicity of the output arc.

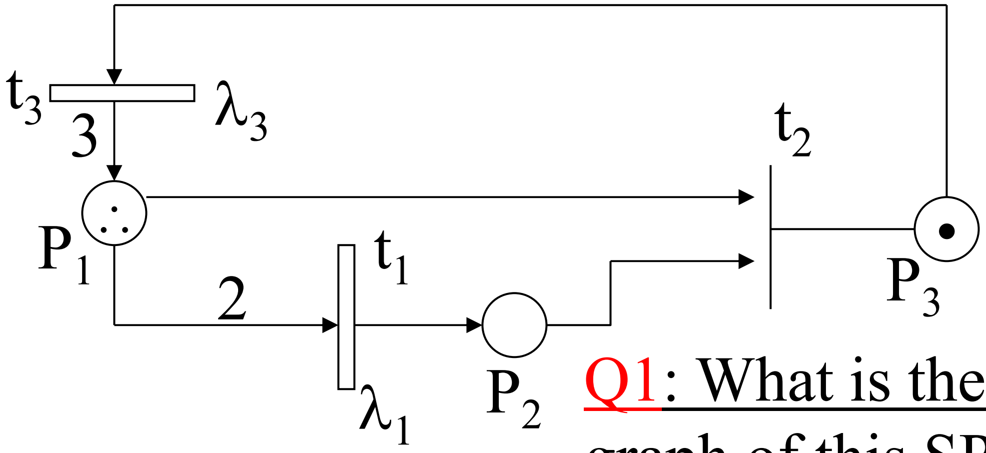
Ans:



Reachability Graph:

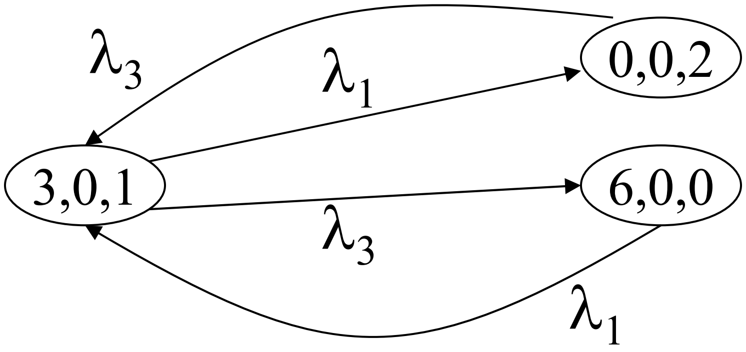
- * The above state diagram is called the reachability graph of the SPN model.
- * When the SPN model does not contain immediate transitions, the reachability graph is a Markov chain.

Q2: What is the underlying Markov model of this SPN?



Q1: What is the reachability graph of this SPN?

Ans:



Ans: same as the one on the previous page except λ_2 is replaced by infinity

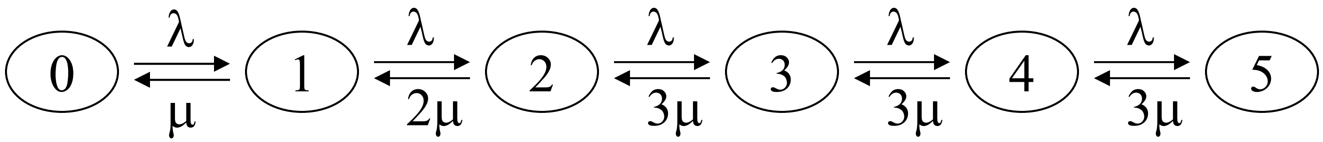
Ex: draw an SPN using inhibitor arcs for an M/M/m/b queue with m=3 and b=5

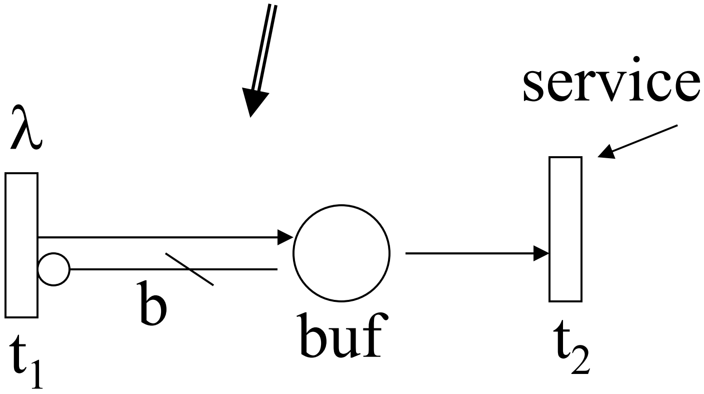
λ

μ

m servers

buffer size limitation



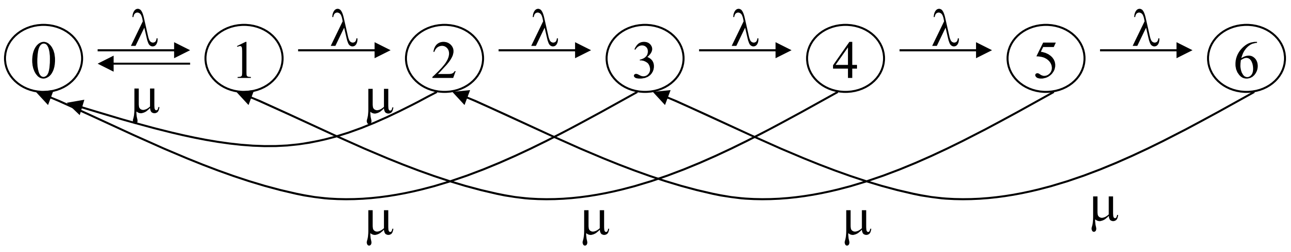


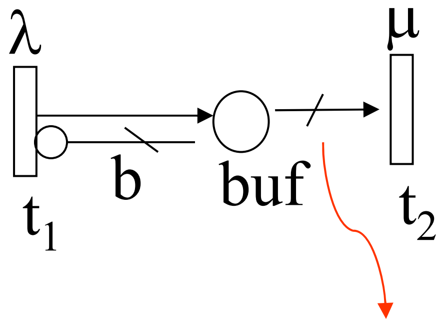
$$\text{service rate} = \begin{cases} (\text{\# of tokens in place “ buf ”}) \times \mu & \text{if \# of tokens} < m \\ m \times \mu & \text{if \# of tokens} \geq m \end{cases}$$

This example features an inhibitor arc and a transition rate function which depends on the marking (state) of the system.

Ex: a M/M/1/6 with a bulk service center (e.g., an elevator) capable of servicing 3 jobs per service whenever there are 3 jobs to be serviced

Markov model





the corresponding SPN model

Define an arc multiplicity function for the input arc from buf to t_2 to return a value of

$$\begin{cases} 3 & \text{if tokens(buf)} \geq 3 \\ \text{tokens(buf)} & \text{otherwise} \end{cases}$$

Stochastic “Reward” Petri Net

Assigning a reward to each “marking” of the system

Like a state in a Markov model

tangible:
a state that enables
no immediate transition

vanishing:
not tangible

absorbing:
a state that does not
enable any transition
(it is also tangible by definition)

vanishing markings are not shown as “states” in the corresponding Markov chain.
(they are shown in the reachability graph)

Structure of an SPNP program: (no main procedure)

The following procedures must be included in an SPNP program:

1. parameters(){}→ for reading input parameters:
   double input(msg) can be called within.

*2. net(){}→ for defining the stochastic Petri net.

3. assert(){}→ for checking illegal markings
   - return(RES_NOER)
   - return(RES_ERROR)

4. ac_init(){}→ called <u>before</u> starting the reachability graph
   construction: normally empty
   (pr_net_info() can be called within)

5. ac_reach(){}→ called <u>after</u> the reachability graph is
   constructed: normally empty
   (pr_rg_info() can be called within)

*6. ac_final(){}→ for calculating & reporting performance results.

*introduced later in detail

Output to .out file

Frequently-used SPNP built-in functions:

1. Within net() for defining a stochastic Petri net:

- place(char *name)
- trans(char *name)
- init(name, n)   the initial # of tokens in place “name” is n
- priority(char *name, int priority)   priority of transition “name”

associating an enabling function to transition “name”

- guard(char *name, func)

    enabling_type(*func)();

Default:
- priority=0 (lowest priority)
- no enabling function
(or a function always returning TRUE)

for a timed transition

- rateval(char *name, rate_type val)
- ratefun(char *name, func)

    rate_type(*func)();

for an immediate transition

- probval(char *name, probability_type val)
- probfun(char *name, func)

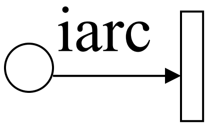
    probability_type(*func)();

“func” can be a marking-dependent function

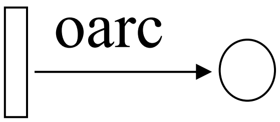
Use mark(p_name) to return # of tokens in place p_name,
& enabled(t_name) to see if transition t_name is enabled

with arc
multiplicity
=1

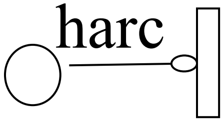
iarc(char *t_name, char *p_name);



oarc(char *t_name, char *p_name);



harc(char *t_name, char *p_name);



with arc
multiplicity = constant

miarc(t_name, p_name, $\overbrace{\text{mult}}^{\text{int}}$)
moarc(t_name, p_name, mult)
mharc(t_name, p_name, mult)

with arc
multiplicity defined
by a function

viarc(t_name, p_name, $\overbrace{\text{func}}^{\text{int (*func)()}}$)
voarc(t_name, p_name, func)
vharc(t_name, p_name, func)

2. within ac_final for reporting the final analysis results:

A. For calculating

E[Z(∞)]

char *msg;

pr_expected(msg, function)

expected(function)

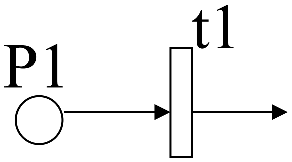
Printing/returning the expected reward defined by “function” which assigns rewards to states

$\underbrace{\text{reward\_type}}_{\text{double}}$ (*function)();

e.g.,

this will assign a reward to every state of the system

```
reward_type   util(){if mark (“P1”) return 1;
                         else return (0);}
reward_type    X(){if enabled (“t1”)
                              return rate (“t1”);
                         else return (0);}
```



Use mark(p_name) to return # of tokens in place p_name,
& rate(t_name) to return the transition rate of transition t_name

B. For calculating $E[Z(t)]$: expected reward at time t

reward_type(*function)()

expected(function)

must call time_value(double t) prior to calling expected(function) in ac_final()

must call para(IOP_METHOD, VAL_TSUNIF) in parameters() for transient analysis

C. For calculating $E[Y(\infty)]$: cumulative expected reward until absorption

cum_abs(function)

reward_type(*function)()

See p.13 SPNP reference guide v.3.1; TSUNIF stand for "Transient Solution using Uniformization";
if not set, the default is VAL-SSSOR (Steady State SOR)

D. For calculating $E(Y(t)]$: cumulative expected reward over [0, t]

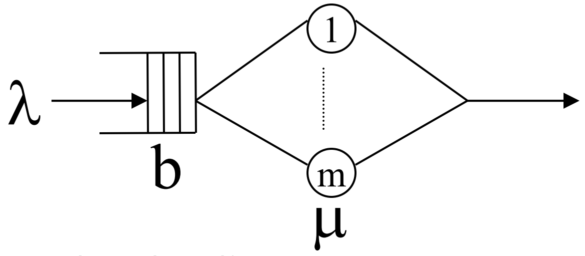
reward_type(*function)()

cum_expected(function)

must call time_value(double t) prior to calling cum_expected(function) in ac_final()

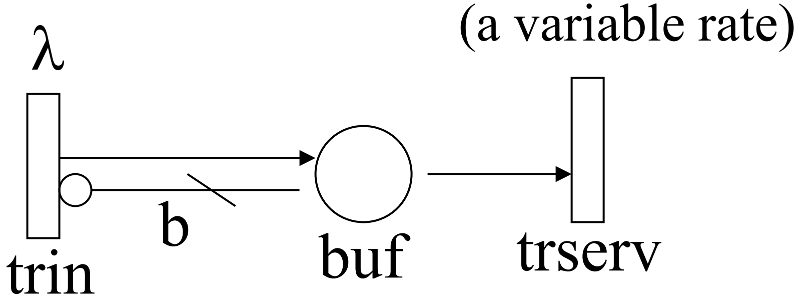
must call para(IOP_METHOD, VAL_TSUNIF) in parameters()

Example:



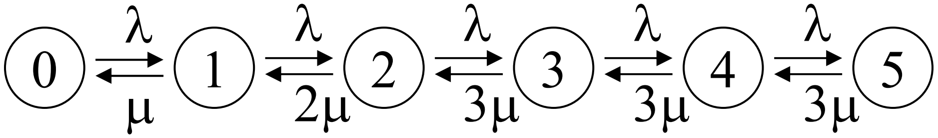
M/M/m/b

Stochastic Petri net model

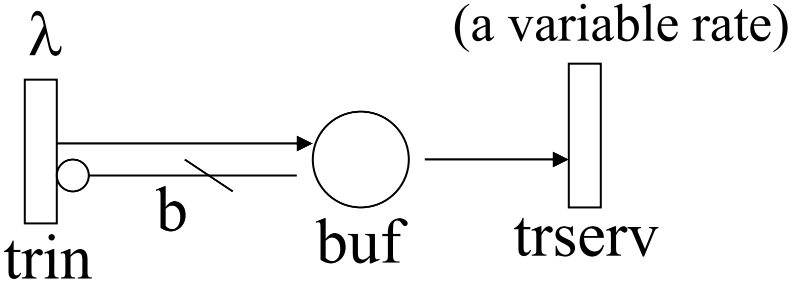


```
#include “user.h”
double lambda;
double mu;
int       b;
int       m;

parameters()
{
    lambda = input(“enter lambda”);
    mu = input(“enter mu”);
    b = input(“enter b”); /*b=5 in this example*/
    m = input(“enter m”); /*m=3 in this example*/
}
```

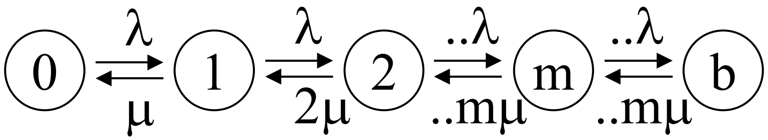
The underlying Markov model of M/M/3/5



```
rate_type   rate_serv()
{
        if ( mark("buf")< m) return ( mark("buf")*mu);
        else return (m*mu);
}
net()
{
        place ("buf");
        trans ("trin");
        trans ("trserv");
        rateval ("trin", lambda); ←——— fixed transition rate
        ratefun ("trserv", rate_serv) ←— variable transition rate
        oarc ("trin", "buf");
        iarc ("trserv", "buf");
        mharc ("trin", "buf", b)
}
```



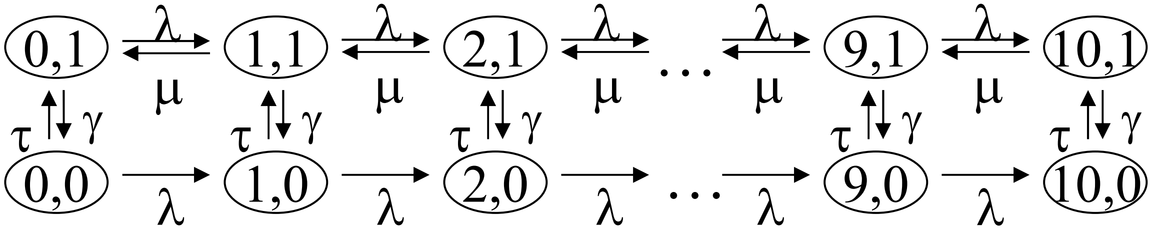
```
assert()
{
        if (mark(“buf”) > b)  return (RES_ERROR);
        else return (RES_NOERR);
}
ac_init() {pr_net_info();}
ac_reach() {pr_rg_info();}
/* reward assignment functions for calculating performance metrics */
reward_type  population() {return (mark(“buf”));}
reward_type  util() {return (enabled(“trserv”));} /* or return mark(“buf”) > 0 */
reward_type  tput() {return (rate(“trserv”));}
reward_type  probrej() {if (mark(“buf”)==b) return (1.0); else return (0.0);}
ac_final()
{
   printf(“average population = %f\n”, expected (population)); /*output to screen*/
   pr_expected (“average throughput”, tput); /* output to *.out */
   pr_expected (“average utilization”, util);
   pr_expected (“rejection probability”, probrej);
   pr_value (“response time”, expected (population)/expected (tput));
}
```



# of servers buffer space

An M/M/N/K queue with server failure & repair

For an M/M/1/10 queue with server failure/repair, we have:



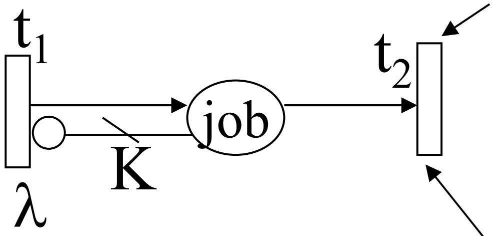
The state representation is (a, b)

# of jobs

# of alive servers

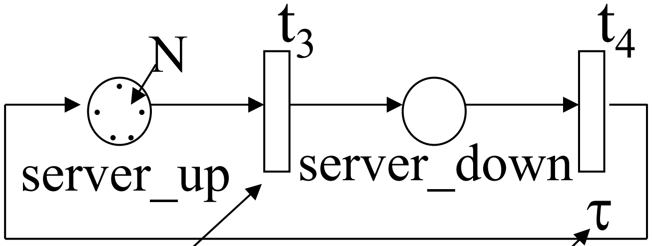
P.235, text

We can study any (N, K) easily with an SPN without having to recreate a Markov model for each (N, K) pair.



enabling function:
if mark(“server_up”)
return true;

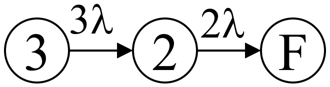
service rate: Depending on # of jobs, i.e., mark(“job”), and # of alive servers, i.e., mark(“server_up”)



variable rate:
mark(“server_up”)*γ
where γ is the per-server failure rate

τ is a fixed repair rate of one facility

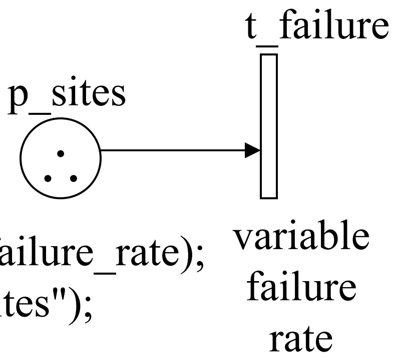
```
* An example to illustrate how to
* calculate reliability and
* mean time to failure of a TMR
* system using a Markov model
markov TMR
3 2   3 * lambda
2 F   2 * lambda
reward
3 1
2 1
end
* initial probability
3 1
end

bind
   lambda 0.001
end
* value(t; system, state) or
* tvalue(t; system, state) returns the
* probability of the system in “state” at time t
func R(t)   1 - value(t; TMR, F)

echo The following loop prints the
echo reliability as a function of t
```



```
 * number of digits after decimal point set to 8
format 8
loop t, 0, 100, 50
   expr R(t)
   expr exrt(t,TMR)
end
expr mean(TMR,F)
expr mean(TMR)
end
=========== output =============
* The following loop prints the
* reliability as a function of t

t=0.000000
    R(t):   1.00000000e+00
    exrt(t,TMR):   1.00000000e+00
t=50.000000
    R(t):   9.93096301e-01
    exrt(t,TMR):   9.93096301e-01
t=100.000000
    R(t):   9.74555818e-01
    exrt(t,TMR):   9.74555818e-01
-----------------------------------------
 mean(TMR,F):   8.33333333e+02
-----------------------------------------
 mean(TMR):   8.33333333e+02
```

```
/* An example to illustrate how to calculate reliability
    and mean time to absorption of a TMR system using SPNP */
#include <stdio.h>
#include "user.h"
#define LAMBDA 0.001

parameters(){
iopt(IOP_METHOD,VAL_TSUNIF); /* for transient analysis */ }
assert() {}
ac_init() { pr_net_info();}

ac_reach(){
  fprintf(stderr,"\nThe reachabiliity graph has been generated \n");
  pr_rg_info();
}
rate_type failure_rate(){
  return(LAMBDA * mark("p_sites"));
}
net(){
  place("p_sites");
  init("p_sites", 3);
  trans("t_failure");
  ratefun("t_failure", failure_rate);
  iarc("t_failure","p_sites");
}
```

See p.13 SPNP reference guide v.3.1; TSUNIF stand for "Transient Solution using Uniformization"; if not set, the default is VAL-SSSOR (Steady State SOR)

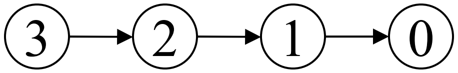


```
reward_type reliability(){
  if (mark("p_sites") >= 2) return(1.0);
  else return(0.0);
}
ac_final(){
  double t;
  for (t=0; t<=100; t += 50){
   time_value(t);
   pr_expected("Reliability at this time = ", reliability);
  }
  pr_mtta("mean time to absorption = ");
}
```

Must call time_value() before calling expected() for transient analysis

```
============= output ================
NET:
=====================================
      places: 1
      immediate transitions: 0
      timed transitions: 1
      constant input arcs: 1
      constant output arcs: 0
      constant inhibitor arcs: 0
      variable input arcs: 0
      variable output arcs: 0
      variable inhibitor arcs: 0
=====================================
RG:
=====================================
      tangible markings: 4 (1 absorbing)
      vanishing markings: 0
      marking-to-marking transitions: 3
=====================================
=====================================
                 TIME :   0.000000000000
=====================================
EXPECTED: Reliability at this time  = 1
```

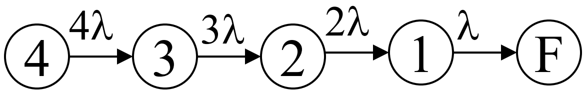
```
=====================================
                 TIME :   50.000000000000
=====================================
EXPECTED:
  Reliability at this time  = 0.993096301257
=====================================
                 TIME :   100.000000000000
=====================================
EXPECTED:
  Reliability at this time  = 0.97455581787
MTTA:
  mean time to absorption  = 1833.33333333
```



Because the absorbing state is 0, not 1.
To model a true TMR system,
add guard("t_failure", t_efunc) in net(){}
where t_efunc is defined as:

```
  enabling_type  t_efunc()
     {
          if  (mark("P_sites") >= 2)
             return 1;
          else return 0;
     }
```

```
* An example to illustrate how to
* calculate the performability of a 1
* out of 4 processor system using sharpe
bind
lambda 0.001
* assume that one processor is able to process
* one job per time unit
mu 1
end
```



```
markov mp readprobs
4 3   4 * lambda
3 2   3 * lambda
2 1   2 * lambda
1 F   lambda
* reward is throughput
reward
4 4*mu
3 3*mu
2 2*mu
1 mu
end
* initial probability
4 1
end
```

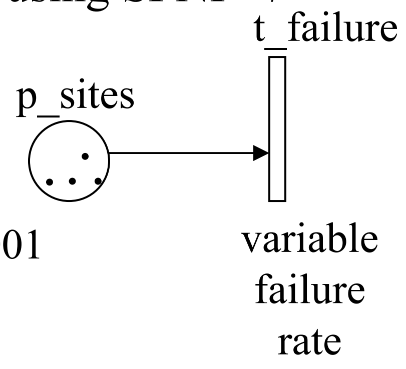
```
* number of digits after decimal point is set to 4
format 4

echo =================================
echo The following loop prints the cumulative
echo expected reward, i.e., the total number of jobs
echo having been serviced, over (0,t)
loop t, 0, 100, 50
* print cumulative expected reward over (0,t)
   expr cexrt(t; mp)
end

echo
echo
echo =================================
echo The following loop prints the probability
echo that cumulative reward is less than a specified
echo value r (i.e., the probability that less than r jobs
echo have been serviced) when the system fails
loop r, 2000, 0, -1000
* print probability that the cumulative reward is less
* than r when the system fails
   expr rvalue(r; mp)
end
```

```
echo
echo
echo ================================
echo how to compute the cumulative expected
echo reward (number of jobs serviced) until
echo absorption ????
end
```

```
================ output ==============
=================================
The following loop prints the cumulative
expected reward, i.e., the total number of jobs
having been serviced, over (0,t)

t=0.000000
    cexrt(t;mp):   0.0000e+00

t=50.000000
    cexrt(t;mp):   1.9508e+02

t=100.000000
    cexrt(t;mp):   3.8065e+02
```

```
=================================
The following loop prints the probability that
the cumulative reward is less than a specified value
r (i.e., the probability that less than r jobs have
been serviced) when the system fails
------------------------------------------


r=2000.000000
    rvalue(r; mp):   1.4288e-01

r=1000.000000
    rvalue(r; mp):   1.8988e-02

r=0.000000
    rvalue(r; mp):   0.0000e+00


=================================
how to compute the cumulative expected
reward (number of jobs serviced) until
absorption ????
```

```
/* An example to illustrate how to calculate
   the performability of a 1 out of 4
   processor system using SPNP */
```



```
#include <stdio.h>
#include "user.h"

#define LAMBDA 0.001
#define MU 1

parameters(){
iopt(IOP_METHOD,VAL_TSUNIF);
                /* Transient analysis */
}
assert() {}
ac_init() { pr_net_info();}
ac_reach(){
  fprintf(stderr,”\nThe reachabiliity graph
                          has been generated \n");
  pr_rg_info();
}
rate_type failure_rate(){
  return(LAMBDA * mark("p_sites"));
}
```

```
net(){
  place("p_sites");
  init("p_sites",4);
  trans("t_failure");
  ratefun("t_failure", failure_rate);
  iarc("t_failure","p_sites");
}
reward_type job_service_rate(){
  if (mark("p_sites"))
     return(mark("p_sites")*MU);
  else return(0.0); /* reward is throughput */
}
ac_final(){
double t;
for (t=0; t<=100; t += 50){
  time_value(t);
  pr_cum_expected ("Expected cumulative
                     number of jobs serviced from (0,t) ",
                job_service_rate);
}
pr_cum_abs("Expected cumulative number
                        of jobs serviced until absorption",
           job_service_rate);
}
```

```
============== output ==============
NET:
====================================
      places: 1
      immediate transitions: 0
      timed transitions: 1
      constant input arcs: 1
      constant output arcs: 0
      constant inhibitor arcs: 0
      variable input arcs: 0
      variable output arcs: 0
      variable inhibitor arcs: 0
====================================
RG:
====================================
      tangible markings: 5 (1 absorbing)
      vanishing markings: 0
      marking-to-marking transitions: 4
====================================
```

```
====================================
          TIME :   0.000000000000
====================================
Expected cumulative number of jobs serviced
from (0,t)  = 0
====================================
          TIME :   50.000000000000
====================================
Expected cumulative number of jobs serviced
from (0,t)  = 195.082301997
====================================
          TIME :   100.000000000000
====================================
Expected cumulative number of jobs serviced
from (0,t)  = 380.650327856
Expected cumulative number of jobs serviced
 until absorption = 4000
```