# The GTPN Analyzer: Numerical Methods and User Interface

by

Mark A. Holliday
Mary K. Vernon

Computer Sciences Technical Report #639

April 1986

# The GTPN Analyzer: Numerical Methods and User Interface

**Mark A. Holliday and Mary K. Vernon**

Computer Sciences Department

University of Wisconsin — Madison

Madison, WI 53706

*Abstract*

The GTPN (Generalized Timed Petri Net) is a performance model based on Petri Nets. The state space for a model of the system is automatically constructed and analyzed using results from Markov chain theory. We address some of the key computational issues involved in the Markov chain theory approach. In particular, we discuss two types of algorithms. The first type compute the *absorption probabilities* and *mean time to absorption.* The second type compute the steady state probability distribution for a single, possibly periodic, recurrent Markov chain class. We also describe the GTPN's user interface for input of the model description, execution of the tool, and the output of the performance results.

This research was partially supported by the National Science Foundation under grants DCR-8402680 and DCR-8451405, and by grants from IBM and Cray Research, Inc.

## Section 1. Introduction

Petri Nets are a formal graphical model of asynchronous parallel computation [PET62, PET81]. Modifying Petri Nets so that time is represented has recently been an active research area [MOL81, NAT80, ZUB80, AJM84, HOL85a]. The goal of these models is to analyze system performance and/or reliability as an extension of the reachability analysis. Our model, Generalized Timed Petri Nets (GTPN) [HOL85a], associates firing frequencies and deterministic firing times with each transition in the net. For the purposes of performance analysis, we view the GTPN as a stochastic process. The time-in-state is a deterministic function for each state of the net. However, a probability distribution is defined over the possible next states based on the firing frequencies. Analysis of the embedded discrete-parameter Markov Chain yields performance estimates. [HOL85b and VER86] contain results for multiprocessor performance issues obtained using the GTPN analyzer.

We have used our GTPN analyzer to solve models with up to 45,000 states. An important issue is the computational efficiency of the performance analysis. An earlier paper [HOL85a] describes efficient methods we developed for building the state space for GTPN models. States in the automatically generated Markov Chain are grouped into *transient* and *recurrent* classes. This paper focuses on numerical methods we use for solving the very large (typically $20,000 \times 20,000$), but very sparse (typically 2-3 non-zero entries per row) matrix equations to get the absorption probabilities and equilibrium state probabilities for the recurrent classes in the Markov Chain. For computing absorption probabilities, we use the key observation that the classes in the Markov Chain form a directed acyclic graph (DAG). For computing equilibrium state probabilities, we use the Power Method, with some shifting and scaling of the probability matrix needed for periodic Markov Chains. A second important issue we address is the user interface of the tool.

The organization of the remainder of this paper is as follows. Section 2 presents an overview of the GTPN and outlines the equations which must be solved. Section 3 considers efficient methods of computing the absorption probabilities and mean times to absorption for the recurrent classes. Section 4 considers methods for computing the steady state probability distribution for each recur-

rent class. Section 5 describes the current user interface of the GTPN. Section 6 summarizes our experiences.

## 2. The GTPN Model

### 2.1. The Net

A GTPN is a Petri net which includes: 1) a deterministic firing duration associated with each transition, 2) a mechanism for specifying next state probabilities for conflicting transitions, and 3) a set of named resources associated with transitions and/or places which are used to calculate performance estimates. We plan to add transition *enabling times* [RAZ83] to support modeling behavior such as timeouts in network protocols. This will require some minor changes in the model definition and how the reachability graph is built.

Letting S denote the set of reachable states, $\Re^+$ denote the positive reals, and $\mathcal{P}$ denote the power set, the current GTPN is formally defined by the following tuple:

$$GTPN = (P, T, A, M_0, D, F, C, R)$$

where $P = \{p_1, p_2, \ldots, p_n\}$ (places)

$T = \{t_1, t_2, \ldots, t_m\}$ (transitions)

$A \subseteq \{P \times T\} \cup \{T \times P\}$ (directed arcs)

$M_0 : P \rightarrow \{0, 1, \ldots\}$ (initial marking)

$D : T \times S \rightarrow \Re^+ \cup \{0\}$ (firing durations)

$F : T \times S \rightarrow \Re^+ \cup \{0\}$ (firing frequencies)

$C : T \rightarrow \{yes, no\}$ (CntComb boolean flags)

$R : T \cup P \rightarrow \mathcal{P}(\{r_1, r_2, \ldots, r_k\})$ (resources)

The first four components of the tuple are identical to the constructs in an untimed Petri Net (see [PET81]). The remaining four components are described below.

Figure 2.1 shows an example GTPN including the initial state distribution of tokens. Each transition is labeled with, from left to right, its firing duration expression, its frequency expression,

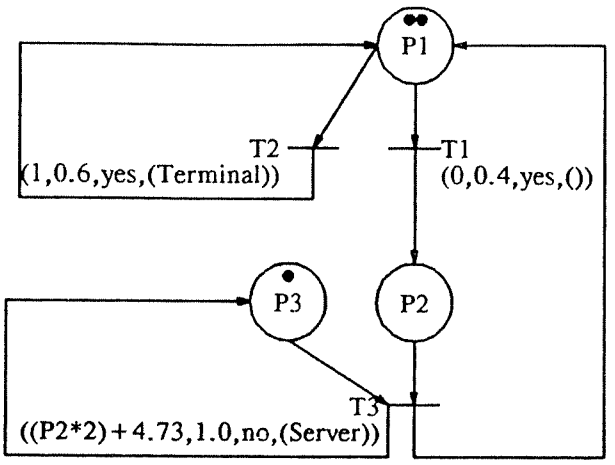


**Figure 2.1.Example of a GTPN net**

its CntComb boolean flag, and its list of resources. There are no resources associated with places in this model.

A transition's firing duration and frequency are expressions containing immediate values (real and integer), names of places (which represents the number of tokens in that place in the current state), names of transitions (which represents the value one if at least one firing of that transition is in progress in the current state and is otherwise zero), and arithmetic, relational, and logical operators. Thus, a transition's firing duration and frequency can be state-dependent, but for a given state they are deterministic.

The example in Figure 2.1 models users at terminals who, with a geometric think time, generate requests for a server. There is one token on place P1 for each user. Transitions T1 and T2 implement the think time. Note that the GTPN can represent geometric, as well as constant, holding times. Transition T3 implements a load-dependent server with a firing duration that depends on the number of tokens on P2.

## 2.2. The Reachability Graph

The *multiplicity* of an input place is the number of arcs from that place to that transition. An output place's multiplicity is defined analogously. An enabled transition *starts firing* by removing from each input place a number of tokens equal to its multiplicity. After start firing, the firing is *in*

*progress* until *end firing*. While the firing is in progress, the time to end firing, called the *remaining firing time* (RFT), decreases from the transition's firing duration to zero. At end firing a number of tokens equal to its multiplicity is put on each output place.

A marking together with the set of current RFT's defines a *state* of the net. State transitions in the GTPN are defined by maximal sets [1] of start firing or end firing events which occur at the same time. A state which has at least one transition enabled (e.g. the initial state in Figure 3.1), has zero duration, and leads to a set of next states defined by all possible combinations of start-firing events that can occur simultaneously. The new RFT's are set to their transitions' durations. If there are no enabled transitions, but there are some firings in progress, then there is one next state generated by the end firing of all firings in progress with the smallest RFT ( *Tmin*). The time-in-state value in this case is *Tmin*. If there are no enabled transitions and no firings in progress, then the net remains in the current state forever.

For each state, a probability distribution is defined over the set of next states. In the nontrivial case, we need to assign a probability to each maximal set of transitions that can start firing together. Calculation of the next state probabilities for start firing events is complex, due to the possibility of conflicting transitions. We only outline our approach and quote the relevant formulas in this paper. The reader is referred to [HOL85a] for a more complete discussion.

Two transitions whose sets of input places intersect are in the same *conflict set*. The transitive closure of the property of intersecting input places defines an equivalence relation on the set of transitions. This equivalence relation partitions the set of transitions into disjoint sets called *generalized conflict sets*. A maximal set of start firing events which comprise a state transition is the union of a set of independent *local maximals*, one from each generalized conflict set. The probability for the maximal is the product of the probabilities for the associated local maximals (since the local maximals are independent). Let *LocalMax[j,i]* denote the $j$th local maximal of the $i$th generalized conflict set. Let *NumComb* denote the *number of combinations*, or the number of

[1] a set with property $\alpha$ is a *maximal* set with property $\alpha$ if it is not a proper subset of any other set with property $\alpha$.

ways tokens can be removed from input places, in order to implement a local maximal. Then the state transition probabilities are given by the following:

$$Pr\{LocalMax[j,i]\} = \begin{cases} \dfrac{NumComb[LocalMax[j,i]] \times \prod_{\{k:k\in LocalMax[j,i]\}} f_k}{\sum_{\{m:m=1,\ldots,M\}} NumComb[LocalMax[m,i]] * \prod_{\{k:k\in LocalMax[m,i]\}} f_k}, & \text{if CntComb = yes;} \\ \dfrac{\prod_{\{k:k\in LocalMax[j,i]\}} f_k}{\sum_{\{m:m=1,\ldots,M\}} \prod_{\{k:k\in LocalMax[m,i]\}} f_k}, & \text{otherwise.} \end{cases}$$

In some cases, the value *NumComb* is needed in order to derive an intuitively reasonable probability. The boolean flag CntComb (Count Combinations) associated with each transition specifies whether this should be done. Only if the flag is *yes* for all transitions in the maximal, is *NumComb* used.

In Figure 2.2 and Table 2.1 we show the reachability graph for the net in Figure 2.1, assuming there is only one user (i.e. one initial token in P1). The labels on the edges of the graph are the next state probabilities. The labels on the vertices of the graph are the values for time-in-state. The marking vectors are shown in the table. The RFT sets are shown as a list of pairs with one pair per in progress firing of a transition. The first component of each pair is the name of the transition. The second component is the remaining firing time. The resources used and their number of uses are also shown in the table.

### Section 2.3. Analysis of the Embedded Markov Chain

Our analysis is based on the key observation that the times at which state changes occur form an embedded, discrete-time, finite state Markov Chain. Consequently, we need to sketch some of the relevant terminology and theorems of Markov Chain theory. We are only concerned currently with analyzing models with finite state spaces. The strongly connected components of the state space (when viewed as a directed graph) are the *classes* of the Markov Chain. The condensed graph (one vertex for each class) is a directed acyclic graph with one root. In the case of a finite state space, the interior vertices of the condensed graph are called *transient* classes. The leaf vertices of the condensed graph are called *recurrent* classes. In a typical evolution of the system being modeled, the system starts in a state in the root of this condensed graph. It then filters through the transient classes until it is absorbed by one of the recurrent classes. Once absorbed it stays in that recurrent class permanently.

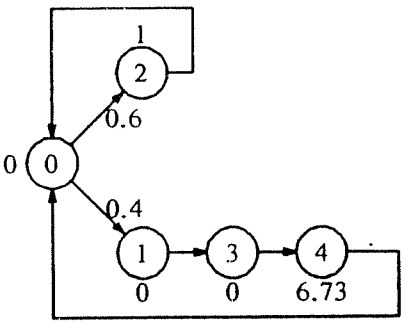


**Figure 2.2.**Reachability Graph for example

**Table 2.1.**Reachable States for example

| States | P1 | P2 | P3 | RFT Set | Resources |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | {} | {} |
| 1 | 0 | 0 | 1 | {(T1,0.0)} | {} |
| 2 | 0 | 0 | 1 | {(T2,1.0)} | {Terminal(1)} |
| 3 | 0 | 1 | 1 | {} | {} |
| 4 | 0 | 0 | 0 | {(T3,6.73)} | {Server(1)} |

Since, in the GTPN, we are interested in long run behavior, there are two characteristics of the movement through the transient classes that are of interest. First, is the *absorption probabilities*, the probabilities of being absorbed by each leaf. Second, is the *mean time to absorption.*

The primary approach to computing the absorption probabilities is based on First Step Analysis [TAY84]. Consider a particular initial state, $i$. On the first state change, the process will move from state $i$ to a state $j$ that is in a transient class or in a recurrent class. If $j$ is in class R, the future probability of being absorbed by class R is one. If $j$ is in another recurrent class, the future probability of being absorbed by class R is zero. If $j$ is in a transient class, then, by the memoryless property, the probability of being absorbed by class R is the same as if $j$ were the initial state.

More formally, suppose that the states in all the transient classes are numbered $0, ..., n-1$ and consider a fixed recurrent class $R$ and fixed initial state $i$. Let $U_{iR} = Pr\{\text{Absorption in class } R | X_0 = i\}$ for $0 \leq i \leq n-1$.

$$U_{iR} = \sum_{\{j \in R\}} P_{ij} + \sum_{j=0}^{n-1} P_{ij} U_{jR}, \qquad i = 0, 1, ..., n-1$$

This equation cannot be solved in isolation. However, if we consider all possible transient initial

states, then we have a system of linear equations that can be solved for the $U_{iR}$'s.

Once absorbed in a particular recurrent class, the probability distribution over the states in the long run needs to be computed. This probability distribution exists for any recurrent class R in a finite state space as long as it is interpreted as the long run expected fraction of visits in each state. This stationary probability distribution is easy to find since it is the unique solution to the set of equations

$$\pi_R = \pi_R P_R \qquad \text{and} \qquad \sum_{j \in R} \pi_j = 1$$

The matrix $P_R$ is $P_R = \{P_{ij} | i, j \in R\}$.

### Section 2.4. Computing Performance Estimates

The embedded Markov Chain has allowed us to obtain, for each recurrent class, the long run expected fraction of visits in each state. We still need to obtain, for each recurrent class, the performance estimate for each resource. This is done in two steps. First, we obtain, for each recurrent class, the long run expected fraction of time, $RelTime(S_1)$, spent in each state $S_1$, i.e. count time-in-state, instead of visits. Let $S$ be the set of states. As shown in [HOL85a], $RelTime(S_1)$ can be computed by:

$$RelTime(S_1) = \frac{TimeInState(S_1)\pi_{S_1}}{\sum_{k \in S} TimeInState(k)\pi_k}$$

Second, the performance estimate for a resource in a given recurrent class is the long run expected number of usages of that resource in that class. Consequently, we simply take the expectation of the random variable ResUsages.

$$E[ResUsages] = \sum_{k \in S} ResUsages(k) Pr\{statek\} = \sum_{k \in S} ResUsages(k) RelTime(k)$$

The probability distribution of the random variable ResUsages with respect to the number of usages of the resource is also computed. Computing this distribution is straightforward. We know the long run probability distribution of time over the states in the recurrent class. We also know the number of usages of the resource in question for each state. Simply summing the probabilities of the states that use the resource the same number of times generates the desired distribution. This

distribution is very useful for reliability prediction (e.g. 90% of the time will at least 3 processors be working?).

## Section 3. Absorption Probabilities and Mean Time to Absorption

As discussed in section 2.3, two of the characteristics of the system's performance that we need to determine are the absorption probabilities and mean time to absorption. First Step Analysis was proposed as the method for computing these characteristics. In this section we discuss the computational issues involved in determining those characteristics. Section 3.1 discusses some of the efficiency issues involved in implementing First Step Analysis for computing the absorption probabilities. Section 3.2 describes an optimization which can significantly accelerate solution of the First Step Analysis equations. Section 3.3 covers the analogous material for computing the mean time to absorption. In an important special case that arises frequently in GTPN models, an alternative to First Step Analysis can be used to compute absorption probabilities. Section 3.4 describes this still more efficient method.

### Section 3.1. First Step Analysis

Using First Step Analysis implies that $r$ systems of $n$ linear equations are solved where $r$ is the number of recurrent classes, and $n$ is the number of transient states in the Markov Chain. Solving one system of equations determines the absorption probability of interest for one recurrent class. Since the GTPN is intended to be a practical tool, an efficient solution method is important.

We write the systems of linear equations in matrix form as follows:

$$U_R = P_{TR} + P_T U_R, \tag{3.1}$$

where $U_R$ is the $n \times 1$ vector of absorption probabilities for recurrent class R from transient states $0, 1, ..., n-1$, $P_{TR}$ is the $n \times 1$ vector of one-step transition probabilities from transient state i to any state in R, and $P_T$ is the $n \times n$ one-step transition probability matrix for the transient states.

The standard form for a system of linear equations is $Ax = b$ where $A$ is $n \times n$, $x$ is $n \times 1$, and $b$ is $n \times 1$. Our form maps into the standard form by letting $A = (P - I)$, $x = U_R$, and $b = -P_{TR}$:

$$(P_T - I)U_R = -P_{TR}. \tag{3.2}$$

There are many methods of solving systems of linear equations. Gaussian elimination is the primary direct method. Iterative methods, such as the Gauss-Seidel method, are much more efficient for large matrices. Before selecting a solution method, however, we will discuss an important optimization that can be applied to First Step Analysis.

### Section 3.2 Optimization of First Step Analysis

An optimization exists that can significantly accelerate solution of the linear systems of equations for First Step Analysis. This optimization is based on a key observation: by permuting the rows and columns of the matrix $P_T$, $P_T$ can be put into block upper triangular form, where each diagonal block represents the transitions within one transient class of the Markov Chain. To see this, recall that the Markov Chain classes are the strongly connected components in the reachability graph. They thus form a directed acyclic graph (DAG), the condensed graph. As with any DAG, the vertices of the condensed graph can be numbered via a topological sort so that the number assigned to a vertex is always less than the numbers assigned to its children. This numbering defines the permutation that generates the block upper triangular form. We find the strongly connected components using Tarjan's $O(n)$ algorithm, and perform the topological sort on the DAG using another linear-time depth-first search [SED83].

After the permutation, equation 3.2 is of the following form:

$$\begin{pmatrix} A_{11} & A_{21} & \cdots & A_{1N} \\ & A_{22} & & A_{2N} \\ & & \ddots & \vdots \\ & & & A_{NN} \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} = \begin{pmatrix} -P_{1R} \\ -P_{2R} \\ \vdots \\ -P_{nR} \end{pmatrix} \tag{3.3}$$

where the elements are matrices, each diagonal block $A_{ii}$ is square of order $n_i$ and

$$\sum_{i=1}^{N} n_i = n.$$

The system (3.3) can be solved as a sequence of $N$ smaller problems. Problem $i$ is of order $n_i$ and the matrix of coefficients is $A_{ii}, i = 1, 2, ..., N$. The procedure is as follows [PIS84]:

(i) Solve the last subsystem, with $A_{NN}$ as the matrix of coefficients, for the last $n_N$ unknowns. Compute the vector $x_N$ of order $n_N$.

(ii) Subtract the products $A_{jN}x_N$ from the right-hand side for $j = 1, ..., N-1$. A block upper triangular matrix of order $N-1$ is obtained, and the procedure is repeated until the complete solution is obtained.

Note that the assumption must be made that the diagonal blocks in Equation (3.2) are non-singular. The solution of each subsystem can be done by any method for solving linear equations, such as Gaussian elimination or the Gauss-Seidel iteration.

### Section 3.3. Computing Mean Time to Absorption

The mean time to absorption can also be computed using a first-step analysis. One system of $n$ linear equations is solved where $n$ is the number of transient states. Each transient state $i$ has one equation. Starting in state $i$, the mean time to absorption is the time-in-state for state $i$ plus zero if the next state is in a recurrent class and plus $P_{ij}$ times the mean time to absorption for state $j$ if state $j$ is transient. Note that the Markov property is again being used. Formally, each equation is of the form:

$$U_i = TimeInState(i) + \sum_{j=0}^{n-1} P_{ij}U_j, \qquad i = 0, 1, ..., n-1$$

If the integer 1 replaces the time-in-state, then the mean number of visits to transient states before absorption can be computed. The mean number of visits to a given set of transient states before absorption can be computed if the time-in-state is replaced by the integer 1 when visiting a state in the given set and is replaced by the integer 0, otherwise.

The discussion in Section 3.2 applies here also. In particular, the system of equations can be changed into the form $Ax = b$, permuted into block lower triangular form, and then solved as outlined.

### Section 3.4. More Efficient Solution of an Important Special Case

In a special case that occurs frequently in GTPN models, an efficient alternative approach based on the condensed reachability graph (i.e. one vertex per strongly connected component) can be used to compute absorption probabilities. In this alternative, probabilities are assigned to the

edges leaving each vertex in the condensed graph A depth first search is then done. During the depth first search the probability along each path to each leaf is determined by taking the product of the edge probabilities along the path. The absorption probability for a given leaf is then simply the sum of the path probabilities terminating at that leaf.

The difficulty with this approach is with determining the edge probabilities leaving a vertex in the condensed graph, If all the exit edges originate at the same vertex $V_1$ within the strongly connected component, then there is not a problem. Find all the edges of which $V_1$ is the parent that are exit edges. Sum their probabilities as edges in the original graph. The probability of each edge in the condensed graph is its probability in the original graph normalized by this sum.

The problem is when more than one vertex, say $V_1$ and $V_2$, in the original graph are parents of exit edges. Determining the probability that each of these parent vertices is the vertex from which exiting occurs is dependent on the detailed structure of the strongly connected component. Our current implementation uses this condensed graph approach when the special case of one parent vertex in the strongly connected component is met. When the special case is not met, first-step analysis with Gauss-Seidel iteration is used. We plan to implement the optimization described in section 3.2 with subsystem solution by Gauss-Seidel iteration.

## Section 4. Stationary Probability Distributions

In this section we discuss the method we use to calculate the stationary probability distribution for each recurrent class in the Markov Chain. Let R denote a recurrent class with states $j = 0, 1, ..., n$, and let $\pi_j$ represent the long run expected fraction of visits to state j, given that the modeled system is absorbed in class R.

Recall that the vector $\pi_R = (\pi_0, \pi_1, ..., \pi_n)$ is uniquely determined by the following equations:

$$\pi_R = \pi_R P_R \qquad \text{and} \qquad \sum_{j=1}^{n} \pi_j = 1, \tag{4.1}$$

where $P_R$ is the $n \times n$ state transition probability matrix for R.

Because the matrix $P_R$ is sparse, but potentially very large, iterative methods are more practical than direct methods. One of the most widely used iterative methods, the Power Method [JOH82],

views equation 4.1 as an eigenvalue problem. In particular, $\pi_R$ is the eigenvector associated with the unit eigenvalue of $P_R$. Applying the Power Method to iteratively solve for $\pi_R$, is done as follows:

$$\pi_R^{k+1} = \pi_R^k P_R.$$

Since our current implementation uses the Power Method, the remainder of this section focuses on issues related to it and, in particular, with ensuring convergence.

### Section 4.1. Spectral Distribution

An eigenvalue, $\lambda$, of a real $n \times n$ matrix A, is ***strictly dominant***, if its modulus is strictly greater than those of all the other eigenvalues of A. Direct application of the Power Method converges to an eigenvector corresponding to the dominant eigenvalue, if and only if the matrix has a simple, strictly dominant eigenvalue.

This constraint on the direct application of the Power Method can be expressed in terms of the periodicity of the recurrent class in the Markov Chain. In order to make that connection between the spectral distribution and the periodicity of the Markov Chain class, we need the theorems of Perron and Frobenius [CIN75,SEN84]. These theorems state the following:

> An irreducible non-negative matrix, $A$, has a real, positive, and simple eigenvalue, $\alpha$, which is ***greater than or equal to*** all other eigenvalues of $A$ in modulus. The eigenvector corresponding to the eigenvalue $\alpha$ is strictly positive. If $A$ is ***aperiodic***, (i.e. $A^k > 0$ for some k), then $\alpha$ is ***strictly greater than*** all other eigenvalues of $A$. A periodic matrix with period $\delta$, has exactly $\delta$ eigenvalues with absolute values equal to $\alpha$. These eigenvalues are all distinct and are given by:
>
> $$\lambda_k = \alpha[e^{2\pi i/\delta}]^{k-1}, k = 1, 2, ..., \delta, i = \sqrt{-1}$$

If $A$ is a stochastic matrix (i.e. all rows sum to one), then $\alpha = 1$. An aperiodic stochastic matrix thus has a simple unit eigenvalue and all other eigenvalues are of strictly smaller modulus. A periodic stochastic matrix with period $\delta$ has exactly $\delta$ eigenvalues of unit modulus all of which

are simple. These eigenvalues can be regarded as a set of points around the unit circle in the complex plane, which goes over into itself under a rotation of the plane by the angle $2\pi/\delta$.

### Section 4.2. Ensuring Convergence

According to the above discussion, the Power Method can only be applied directly to find the stationary probability distribution when the recurrent class is aperiodic. To handle the case of a periodic recurrent class, the following theorem [STE74] concerning shifting and scaling the matrix becomes important:

*Theorem*: Let $A$ be a complex $n \times n$ matrix, and let $\lambda$ be an eigenvalue of $A$ with eigenvector x. Then:

1. $a\lambda$ is an eigenvalue of $aA$ with eigenvector x.
2. $\lambda - b$ is an eigenvalue of $A - bI$ with eigenvector x.

This theorem allows us to transform the matrix $P_R$ to ensure that the unit eigenvalue is strictly greater than all other eigenvalues in modulus. In particular, we make the following transformations on $P_R$:

$$P'_R = \epsilon(P_R - I) + I = \epsilon P_R + (1 - \epsilon)I \qquad 0 < \epsilon < 1.$$

The first transformation, subtracting I, shifts all eigenvalues of $P_R$ to the left by one in the complex plane. The eigenvector $\pi_R$ now corresponds to the eigenvalue zero. The second transformation, multiplication by $\epsilon$, shrinks all of the eigenvalues except the zero eigenvalue corresponding to $\pi_R$ (the circle is now centered at $-\epsilon$ and is of $\epsilon$ radius). The third transformation, adding I, shifts all eigenvalues to the right by one, creating a unit eigenvalue corresponding to $\pi_R$ whose modulus is strictly greater than that of all other eigenvalues. Thus, application of the Power Method using $P'_R$ always converges.

### Section 4.3. Convergence Rate

The convergence rate of this method is essentially the rate at which $\lambda_2'^k$ converges to zero, where $\lambda'_2$ is the second largest eigenvalue of the matrix $P'_R$ [JOH82]. The value of $\epsilon$ indirectly influences the rate of convergence by scaling all of the eigenvalues. Wallace and Rosenberg [WAL66] define a

suitable value for $\epsilon$ to be $0.99 \times [max(|P_{ii} - 1|)]^{-1}$. We have used this value, which equals 0.99 for most GTPN models, in our implementation.

In general,the value of $\lambda_2'$ depends on the size and structure of the particular GTPN model constructed, and convergence rates vary considerably. In particular, the convergence rate may be extremely slow. A recent paper by Stewart and Goyal [STE85], suggests that successive overrelaxation is a better method for solving steady-state equations for continuous time Markov chains. We plan to investigate whether this approach would also be more efficient for the GTPN.

## Section 5. The GTPN Analyzer User Interface

This section describes the current user interface to the GTPN tool. This interface has three parts: the format of the model description input to the tool, the process of running the tool, and the format of the output of the performance results. We should note that the nature of this interface is largely independent of the rest of the tool.

### Section 5.1. The Input Format

The model description is input in textual form. This text is processed by table-driven lexical and syntactic analyzers. In particular, we use two tools, Scangen and LLgen, that have been developed by others at the University of Wisconsin—Madison. Scangen accepts descriptions of tokens written as regular expressions and generates tables to drive a lexical analyzer. LLgen accepts a context-free grammar specification and generates tables for parsing sentences of the specified language. It generates tables for any LL(1) grammar. During execution of the GTPN tool, the tables generated by Scangen and LLgen are used in conjunction with the parser and semantic routines to construct the internal tables describing the net and its initial state.

The input format is quite simple. As an example, in Figure 5.1 is the actual input for the net which is shown graphically in Figure 2.1. There are four reserved words: NET, RESOURCES, INITSTATE, and END. These four words, in that order, divide the input into three sections.

The NET section is a sequence of entries, one for each transition. A transition's entry first lists the transition's input places, the token $--$ >, and the transition's output places. Each input place

```
NET
    P1(1) -- > P2(1): 0, 0.4, yes, ;
    P1(1) -- > P1(1): 1, 0.6, yes, Trans2;
    P2(1), P3(1) -- > P1(1), P3(1): (P2*2)+4.73, 1.0, no, Trans3;
RESOURCES
    Terminal: Trans2;
    Server: Trans3;
INITSTATE
    P1(2), P3(1)

END
```

**Figure 5.1.**Input Format for Example

has a number in parenthesis which is the number of tokens removed by a single firing. Each output place has a number in parenthesis which is the number of tokens added by a single firing. After the colon, the transition has four remaining fields: its duration expression, its frequency expression, the CntComb flag, and an optional name. A transition needs a name if it is to be referenced in a duration or frequency expression or in the RESOURCES section.

The RESOURCES section has one entry for each resource (performance measure). A resource's entry names the resource and then lists a sequence of names of places and transitions. In a given state the number of usages of a resource are determined by these place and transition names. A place has as many resource usages as there are tokens on it. A transition has as many resource usages as there are firings in progress. A more general specification of performance measures would allow expressions containing arithmetic, logical and relational operators as well as place and transition names. We have not yet implemented this capability, but it appears straightforward.

The INITSTATE section lists the places which contain at least one token in the initial state. Each such place has the number of its initial tokens in parenthesis.

There will be some minor changes to this format when we introduce enabling times.

## Section 5.2. Running the GTPN

The GTPN tool is a single executable file. It is invoked by the command line:

gtpna -[a-z] *file*

The net description comes from *file*. The results go to standard output. Various optional flags display aspects of the internal state of the tool and are useful for debugging.

### Section 5.3. Output Format

The output is also textual. First, general information is listed. This includes the number of places, transitions, resources, and states. Second, for each leaf, the performance results for each resource are given as well as the number of iterations needed for the Power Method to converge. Third, the absorption probability for each leaf is listed. Fourth, for each leaf and for each resource, the probability distribution over the number of resource usages is given.

## 6. Conclusions

A careful study of implementation issues is essential to the sucessful development of a modeling tool. We have discussed some of the most important implementation issues in the Generalized Timed Petri Net modeling approach. The first general issue concerns using results from numerical linear algebra to aid in efficient analysis of the state space.

The two important transient characteristics of the state space are the absorption probabilities and mean time to absorption. We proposed using First Step Analysis to compute these characteristics. In this method several systems of linear equations are solved. Each system could be solved immediately by Gauss-Seidel. Alternatively, we suggested a more efficient solution based on permuting the matrix into block upper triangular form and then solving a sequence of smaller problems. In an important special case a still more efficient method based on the graph of strongly connected components can be used.

The important steady state characteristic is the steady state probability distribution within each recurrent class. This distribution is determined by another system of linear equations. We currently solve this system of equations by treating it as an eigenvalue problem and using the Power Method. A direct application of the Power Method, unfortunately, does not converge if the recurrent class is periodic. We discuss how scaling and shifting of the transition probability matrix can ensure convergence without changing the desired eigenvector. We briefly discuss the

convergence rate. We plan to investigate whether other methods may be superior to the Power Method.

The second issue is the user interface to the GTPN tool. The most interesting aspect of this interface is the use of a table-driven scanner and a table-driven parser has greatly aided in the processing of the textual form of the input description.

**Acknowledgements**

Many people have made helpful comments and suggestions during our research. We would like to especially thank Klaus Hollig and Diane O'Leary.

**References**

[AJM84] M.A. Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets," *ACM Trans. on Computer Systems*, vol. 2, pp. 93-122, May 1984.

[CIN75] E. Cinlar, *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[HOL85a] M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis (extended version)," Tech. Rep. #593, Computer Sciences Dept., UW-Madison, May 1985, to appear in *IEEE Trans. on Software Engineering.*

[HOL85b] M.A. Holliday and M.K. Vernon, "Exact Performance Estimates for Multiprocessor Memory and Bus Interference," to appear in *IEEE Trans. on Computers*, also Tech. Rep. #594, Computer Sciences Dept., UW-Madison, May 1985

[JOH82] L.W. Johnson and R.D. Riess, *Numerical Analysis, Second Edition*, Addison-Wesley, Reading, MA, 1982.

[MOL81] M.K. Molloy, "On the integration of delay and throughput measures in distributed processing models," Ph.D dissertation, Univ. California, Los Angeles, 1981.

[NAT80] S. Natkin, "Reseaux de Petri stochastiques", these de Docteur-Ingenieur, CNAM-Paris, June 1980.

[PET81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems,* Englewodd Cliffs, NJ: Prentice-

Hall, 1981.

[PET62] C.A. Petri, "Kommunikation mit Automaten," *Schriften des Rheinisch– Westfalischen Institute fur Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany, 1962; translation: C.F. Greene, Supplement 1 to Tech. Report RADC-TR-65-337, Vol. 1, Rome Air Development Center, Grifiss Air Force Base, NY 1965.

[PIS84] S. Pissanetzky, *Sparse Matrix Technology.* Academic Press, Orlando, Florida, 1984.

[RAZ83] R.R. Razouk and C.V. Phelps, "Performance Analysis Using Timed Petri Nets," in *Proc. 1984 Int. Conf. on Parallel Processing*, pp. 126-129, August, 1984.

[SED83] R. Sedgewick, *Algorithms.* pp. 428-430, Reading, MA: Addison-Wesley, 1983.

[SEN81] E. Seneta,, *Non-negative Matrices and Markov Chains, Second Edition*, Springer-Verlag, New York, NY, 1981.

[STE73] G.W. Stewart, *Introduction to Matrix Computations,* Academic Press, New York, NY, 1973.

[STE78] W.J. Stewart, "A Comparison of Numerical Techniques in Markov Modeling," in *Communications of the ACM*, Vol.21, No.2, February 1978, pp.144-152.

[STE85] W.J. Stewart and A. Goyal, "Matrix Methods in Large Dependability Models," Tech. Report RC 11485, November 1985, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

[TAY84] H.M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, Academic Press, Orlando, Florida, 1984.

[WAL66] V.L. Wallace and R.S. Rosenberg, "The Recursive Queue Analyzer," Systems Engineering Dept., Tech. Report #2, Univ. of Michigan, Ann Arbor, MI, 1966.

[VER86] M.K. Vernon and M.A. Holliday, "Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets," to appear in *Performance '86 and ACM SIGMETRICS '86 Joint Conference on Computer Performance Modeling, Measurement and Evaluation*, also Technical Report #618, Computer Sciences Dept., UW-Madison, November 1985

[ZUB80] W.M. Zuberek, "Timed Petri nets and preliminary performance evaluation," in *Proc. 7th An-*

*nual Symp. on Computer Architecture*, pp. 88-96, 1980.