Tools for Efficient Analysis of Concurrent Software Systems

Rami R. Razouk
Daniel S. Hirschberg

ABSTRACT 85-15

The ever increasing use of distributed computing as a method of providing added computing power and reliability has sparked interest in methods to model and analyze concurrent hardware/ software systems. Efficient automated analysis tools are needed to aid designers of such systems. The Distributed Systems Project at UCI has been developing a suite of tools (dubbed the P-NUT system) which supports efficient analysis of models of concurrent software. This paper presents the principles which guide the development of P-NUT tools and discusses the development of one of the tools: the Reachability Graph Builder (RGB). The P-NUT approach to tool development has resulted in the production of a highly efficient tool for constructing reachability graphs. The careful design of data structures and associated algorithms has significantly enlarged the class of models which can be analyzed.

Technical Report #85-15

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

June 1985

Tools for Efficient Analysis of Concurrent Software Systems

Rami R. Razouk†
Daniel S. Hirschberg
Information and Computer Science Department
University of California, Irvine
Arpanet addresses: razouk@uci, dan@uci

Abstract

The ever increasing use of distributed computing as a method of providing added computing power and reliability has sparked interest in methods to model and analyze concurrent hardware/ software systems. Efficient automated analysis tools are needed to aid designers of such systems. The Distributed Systems Project at UCI has been developing a suite of tools (dubbed the P-NUT system) which supports efficient analysis of models of concurrent software. This paper presents the principles which guide the development of P-NUT tools and discusses the development of one of the tools: the Reachability Graph Builder (RGB). The P-NUT approach to tool development has resulted in the production of a highly efficient tool for constructing reachability graphs. The careful design of data structures and associated algorithms has significantly enlarged the class of models which can be analyzed.

1. Introduction

The increasing availability of low-cost processors, coupled with ever-increasing demands for processing power and reliability, have sparked interest in the design of distributed systems. The design of software for such systems requires great care. In addition to the well-understood problems inherent in the design of sequential software, distributed software is subject to timing, synchronization and resource contention errors. These errors are difficult to isolate and are often uncovered at late stages of the development process, resulting in added development costs.

† This research has been supported in part by a MICRO grant co-sponsored by Hughes Aircraft Co. and the University of California, and by a grant from the National Science Foundation (grant # DCR 84-06756).

One approach to solving these problems is to construct models of the software at various stages of the development process, and to analyze the models in an effort to gain a better understanding of system behavior. Models are usually abstract representations of the actual system, and generally omit details which are either irrelevant to the questions being investigated, or are not known. Models derived from early drafts of requirement specifications can be considered early prototypes which can be used to clarify customer needs. Models constructed during the design process can be used to investigate performance and correctness issues long before actual hardware or software is constructed. In general, models are intended to provide designers with early feedback on the impact of certain design decisions.

The Distributed Systems Project at UCI has been investigating new techniques, based on Petri Net models, for analyzing distributed systems with an eye on both correctness and performance issues. One result of the research has been the development of a suite of highly efficient tools which can aid a designer in verifying that some key system requirements are met. This paper outlines the suite of tools which have been developed (the P-NUT system) and then focuses attention on one of the tools: the Reachability Graph Builder. Section 2 of the paper briefly reviews Petri Nets and discusses existing analysis methods which have been shown useful. Section 3 describes the overall design philosophy of the P-NUT system and briefly describes existing tools. Section 4 demonstrates the impact of the overall design philosophy on the design of a reachability graph builder. Section 5 presents some performance data on the reachability graph builder which shows that relatively large and complex models can be effectively analyzed.

2. Petri Net Models

Petri Net models have long been proposed as useful tools for investigating issues of timing, synchronization and resource contention. Since a detailed introduction to Petri Nets is beyond the scope of this paper, the reader is referred to [Peterson J. 81] for an excellent introduction to the subject. In the discussion below, we assume that the reader has some familiarity with Petri Nets.

Petri Net models require the designer to describe each component of a system in terms of conditions (places) and events (transitions). Each possible event in a system component is given a set of pre-conditions which must hold before the event can occur (a transition can fire). Any time the pre-conditions of an event hold, the event can occur. Each possible event is also given a set of post-conditions: conditions which must hold immediately after the occurrence of the event. Petri Net models can be represented graphically with places drawn as circles and transitions drawn as bars (or rectangles). Pre- and post-conditions are drawn as arcs between places and transitions.

Petri Net models are particularly well suited to modelling concurrency and resource contention in both hardware and software systems. All transitions whose pre-conditions are not mutually exclusive, can potentially fire at the same instant (although each firing is an atomic action). Resource contention can easily be modeled by constructing transitions whose pre-conditions require the availability of the same resource and whose post-conditions show the resource as being unavailable. In such cases only one of the contending transitions can fire. The simplicity of Petri Net models makes them analyzable and understandable.

One field where Petri Net models have been used extensively is that of communications protocols. In that field, Petri Nets have been shown to be useful in proofs of correctness [Symons F. 80, Berthelot G. 82, Berthomieu B. 83] and performance analysis [Molloy M. 82, Ramchandani C. 74, Ramamoorthy C. 80, Holliday and Vernon 85, Razouk and Phelps 84].

As an example of the use of Petri Nets to describe communication protocols, Figure 1 shows the textual description of a Petri Net model of the alternating bit protocol [Bartlett et.al. 69]. This model assumes a communication medium of capacity one. Line 3 of the description shows that if the sender is ready, and the sender flag has the value i (zero or one), then the sender can enter the sending state, retain a flag i, and enqueue a message with sequence number i. Line 25 shows that if the receiver is waiting, and it receives a message with a number which matches its flag, it changes its flag (increments it by one, modulo 2), enters the sending_positive_ack state, and enqueues an acknowledgment packet with the same number as the message received. Line 26 shows the receiver action when packets are received with numbers which do not match the expected number (duplicate packets).

Petri Net models have been the subject of much theoretical work which has resulted in the development of well understood analysis methods. Among these methods are:

Invariant analysis. Techniques have been developed to automatically (and efficiently [Martinez and Silva 81]) derive a set of equations describing invariant properties of Petri Net markings (token distributions). This type of analysis cannot be used in isolation to prove properties related to state transitions and sequences of state transitions.

Exhaustive state exploration (reachability graphs). This common analysis method is based on constructing all possible successor states of the initial state. Although the analysis can be completely automated, the complexity of the analysis limits the class of nets which can be effectively analyzed. A reachability graph, since it contains all state transitions, can be used to prove properties related to sequences of state transitions.

Formal verification by induction. Keller [Keller R. 76] showed how induction can be used to verify properties of concurrent programs. This early work has had a strong influence on subsequent work on Petri Nets. This type of verification requires a good deal of creativity on the part of the individual constructing the model. Inductive proofs are more powerful than the Invariant Analysis discussed above since they can be applied to Petri Nets which have been

```
/* Sender model */
1. for i = 0 to 1
2. {
3. S_ready,S_flag[i]                           → S_sending,S_flag[i],Q_m[i]                      /* Queue message */
4. S_wait_ack,S_flag[i],S_ack[i]               → S_ready,S_flag[i+1 % 2]                         /* Good ack */
5. S_wait_ack,S_flag[i],S_ack[i+1 % 2]         → S_sending,S_flag[i],Q_m[i]                      /* Bad ack */
6. S_wait_ack,S_flag[i],read_RS_Q              → S_sending,S_flag[i],Q_m[i]                      /* Timeout */
7. }
8. S_sending,SR_Q_sent                         → S_wait_ack, read_RS_Q                           /* Wait for ack */

/* Sender-to-Receiver Queue model */
9. for i = 0 to 1
10. {
11. Q_m[i]                                     → SR_Q_sent                                       /* lose packet */
12. Q_m[i],filled_SR_slot                      → SR_Q_sent,filled_SR_slot                        /* Queue OVFL */
13. Q_m[i],free_SR_slot                        → slot_m[i],filled_SR_slot,SR_Q_sent              /* Transmit*/
14. filled_SR_slot,read_SR_Q,slot_m[i]         → free_SR_slot,R_m[i]                             /* Give to Receiver */
15. }

/* Receiver-to-Sender Queue model */
16. for i = 0 to 1
17. {
18. Q_ack[i]                                   → RS_Q_sent                                       /* lose packet */
19. Q_ack[i],filled_RS_slot                    → RS_Q_sent,filled_RS_slot                        /* Queue OVFL */
20. Q_ack[i],free_RS_slot                      → slot_ack[i],filled_RS_slot,RS_Q_sent            /* Transmit */
21. filled_RS_slot,read_RS_Q,slot_ack[i]       → free_RS_slot,S_ack[i]                           /* Give to Sender */
22. }

/* Receiver model */
23. for i = 0 to 1
24. {
25. R_waiting,R_flag[i],R_m[i]                 → R_flag[i+1 % 2],R_sending_pack,Q_ack[i]         /* Positive Ack */
26. R_waiting,R_flag[i],R_m[i+1 % 2]           → R_flag[i],R_sending_nack,Q_ack[i+1 % 2]         /* Negative Ack */
27. }
28. R_sending_nack,RS_Q_sent                   → R_waiting, read_SR_Q                            /* Wait for message */
29. R_sending_pack,RS_Q_sent                   → R_ready                                         /* Ready to receive */
30. R_ready                                    → read_SR_Q,R_waiting                             /* Receive next message */

/* Initial State */
31. <S_ready, R_ready, S_flag0, R_flag0, free_SR_slot, free_RS_slot>
```

Figure 1. Petri net model of alternating-bit protocol

extended with predicates and actions (Predicate/Action Nets).

Simulation. Since it is sometimes impractical to *prove* a Petri Net model to be totally correct with respect to some specification, it is desirable to exercise the model under some controlled test conditions. Simulation environments based on Petri Net related models [Vernon M. 82] have been constructed.

Performance analysis. Petri Net models which have been extended to include timing information (e.g. processing delays, timeouts) have been used to derive performance measures[Ramchandani C. 74, Ramamoorthy C. 80, Molloy M. 81, Razouk and Phelps 84, Holliday and Vernon 85].

Each analysis method attempts to answer important questions about the correctness of the model or about expected performance. Each technique has strengths and weaknesses. No single model and analysis method is ideal for use under all circumstances. Therefore, the use of Petri nets in the design of complex distributed systems requires a variety of tools which can be mixed and matched to achieve the overall goal of ensuring that the designs being modeled meet the overall system requirements.

3. Philosophy

The suite of tools under development at UCI has been dubbed P-NUT (for Petri-Net UTilities). A few simple guiding principles have been used during the development of the P-NUT system. None of these principles are particularly new or innovative. The discussion below is simply a clarification of some of the overall concepts which resulted in certain specific design decisions.

1. *Tools should be built in small pieces which can be mixed and matched.* Each tool has a small, well-defined, function. Algorithms and Data Structures can be tuned to minimize space and time requirements (worst-case or average-case). The tuning of data structures and algorithms is particularly important in this type of software since many of the analysis methods involve solving problems that are known to be NP-complete, and hence require exponential time. Careful implementation will have a significant impact on the size of the problems which can be effectively analyzed.

2. *Tools should take advantage of designer's understanding of the design.* Building the most general tool will cost in terms of execution efficiency.

3. *Tools should have interfaces which are simple (for debugging) and flexible (enhancements to one tool should not cause others to change).* The interfaces between the tools should rely on a few standard forms which are consumed and produced by every tool. The adoption of standard forms leads to the ability to interface tools which may not have been intended (during their design) to be interfaced.

4. *Tools are to be developed in an environment which will enhance portability.* Portability issues should be addressed by selecting appropriate implementation languages and operating systems which are known to be widely used by researchers in the field. All software should be in the public domain.

The current tools available in P-NUT are:

Translator (timed or untimed). This tool processes textual descriptions such as that shown in Figure 1. The output of the tool is a standard form of Petri Net which carries sufficient information to support efficient processing by other tools (such as the number of places and transitions). The tool alleviates the need for other tools to interface with the user in a user-friendly manner. The translator accepts Petri Nets which have been extended through the addition of time delays, firing probabilities (to model probabilistic events), enabling predicates and firing actions. Any and all of these extensions can be omitted from any particular model.

Untimed Reachability Graph Builder (RGB). This tool constructs reachability graphs, ignoring user-specified timing information, predicates and actions. The result is a time-independent graph which can be used to verify properties which hold independent of specific timing delays.

Timed RGB. This tool implements the performance analysis described in [Razouk and Phelps 84]. Unlike the untimed RGB, it takes into account timing constraints and firing probabilities. This tool ignores predicates and actions.

Reachability Graph Analyzer. This is one of the most innovative tools in P-NUT. It allows designers to ask questions about reachable states and possible sequences of events. By automatically searching the reachability graph, the tool can answer the user's questions.

Reachability Graph Pretty-Printer. This tool displays reachability graphs on a simple computer terminal.

Currently under development are a set of tools which focus on performance and which provide more flexible user interfaces. Among these tools are:

Simulator (Animator). This tool is intended to support *fully interpreted* Petri Nets: Petri Nets which may have predicates, actions and timing delays. The simulator is in its early development and is being developed on a SUN-2 workstation using a high-resolution bit-mapped graphics input/output device.

Interactive Graphics Petri Net Editor. Each net that is intended to be used in the Animator will have a graphical representation. The Editor will enable easy and rapid modification of that representation. This tool is also in its early stages of development.

For the remainder of this paper the focus of attention will be on one particular tool, the RGB.

4. The Reachability Graph Builder

Focus

Consistent with the philosophy outlined above, the Reachability Graph Builder (RGB) focuses on only one task: constructing the graph containing all states reachable from the initial state of the model. It has not been burdened with the additional tasks of displaying, perusing or analyzing the graph. It also does not allow for editing of the net. As a result, some a-priori knowledge of the net is assumed, most important of which are the numbers of places and transitions, which are assumed to be fixed. Fixed-sized vectors can then be used to represent transitions in place of linked lists. This leads to a speed-up in processing and a reduction in storage space. The alternative is the use of linked lists which, while providing flexibility, are slower to process and waste storage for links. This storage is minimized in the case of simple (no weighted arcs) Petri Nets by storing bit-maps of the input and output sets of each transition. The major variable in this tool is the size of the reachability graph.

The RGB has several key tasks which are time consuming: 1) identifying firable transitions in each state, 2) calculating successors and detecting duplicates, and 3) detecting unbounded graphs. These tasks have been optimized as follows.

1. Identifying firable transitions in each state. The token requirements for firing, and the current token population, are stored as vectors of integers. A transition is firable if the result of subtracting the token requirement from the current token population is non-negative. The detection is accomplished by comparing the current token population with the input vector. In the worst case, when the transition is firable, this involves n comparisons (where n is the number of places). In the case of safe nets, where each place never holds more than one token, the vector of integers can be compressed to become a vector of bits. Detecting firability is implemented by Boolean manipulation of the bit vectors (requiring 2 Boolean operations). In the worst case, only $2n$/WORDSIZE Boolean operations are needed. (In the slightly more general case, where each place is known not to hold more than 2**k tokens, only k bits are needed to store the related integer.)

2. Calculating successors and detecting duplicates. For the integer case, successors are generated by subtracting the input tokens and adding the output tokens of the transition to the current token population. This can be done using only n arithmetic operations, since the input and output token vectors can be precombined to a "net difference" vector. In the bit-vector case, the successor vector can be obtained by exclusive-or (modulo 2 addition and subtraction) of the current token population vector with the bit vectors representing the input and output tokens of the transition. This can be accomplished using only

n/WORDSIZE Boolean operations, since the exclusive-or of the input/output transition token vectors can be precomputed.

Once successors have been detected, a major difficulty is determination of whether the newly-generated state is new or was previously encountered. This is accomplished using a straightforward hashing scheme, in which states are hashed into a large table with linear probe used for collision resolution. As a result, duplicate detection is accomplished using (on the average) less than two vector comparisons (involving $2n$ arithmetic comparisons). For the bit-vector case, less than $2n$/WORDSIZE compares are needed on the average.

3. Detecting unbounded graphs. This is the most time-consuming portion of the program. For safe nets and nets which are known to be bounded, this phase is unnecessary and is omitted. Otherwise, we must ensure that any new state (multiset of positioned tokens) is not a superset of one of its ancestor states. If a new state should be a superset of an ancestor then that state is precluded from generating any successors and is denoted as an "infinite" state.

   The graph is expanded in a breadth-first manner. That is, there is a set of states (called the frontier) for which successors have not yet been determined. The successors of all of the members of the frontier are determined and, only then, the successors are checked to determine if they can potentially lead to an infinite number of other states. This avoids the problem of repeatedly checking ancestors of states that are successors of more than one state in the frontier.

Taking advantage of designer's knowledge

The RGB has been built to take advantage of the designer's knowledge of the model. For example, if the model is known to be bounded, then checks for infinite graphs are not necessary. This leads to immense savings in processing time. If the model is known to be bounded by some small number (such as 127), then vectors of integers can be packed into vectors of bytes which consume less storage. If the model is known to be safe, then vectors of integers are compacted into vectors of bits. The calculation of successor states can be accomplished using Boolean operations on bit-vectors. Savings in time and space have been achieved in this area. The above savings were further enhanced by having different tools for each of these cases. These tools are generated from a single version of the software by conditionally compiling different code sub-segments that interact with the different data structures. As a result, each tool can run at full speed without need of checking user-defined information.

Simple and flexible interface

The RGB expects as input a canonical representation of the Petri Net. This representation has been designed to accommodate various extensions to normal Petri Nets. Among these extension are: 1) enabling times and firing times which model timing delays, 2) firing probabilities to model probabilistic events, 3) enabling predicates, and 4) firing actions. The interface between the Petri Net translator which generates the canonical form and the RGB has been built in a way which allows each tool to pick and choose the parts of the Petri Net which are relevant to that tool. For example, the RGB ignores all timing information. It is expected that this approach will ease incorporation of new Petri Net extensions without necessitating any updating of existing tools. The price being paid is a loss of efficiency both in time and space during the storage and retrieval of the Petri Net. Since the tools being built are computation-intensive (time growing exponentially with the size of the net), it is expected that the processing delay for loading the net will be insignificant even for large nets (time growing linearly with the size of the net).

The output of the RGB is also simple ASCII text which can be stored in a file, passed on to the reachability graph analyzer or to the pretty-printer. UNIX's‡ capabilities for piping are particularly useful in routing the output of the RGB to the desired post-processing tool. The selection of a standard form for reachability graphs has simplified the task of incorporating new tools into the P-NUT system. For example, the pretty printer and reachability graph analyzer which were originally intended to process only un-timed reachability graphs have now been extended to process the output of the Timed Reachability Graph Builder.

Portability

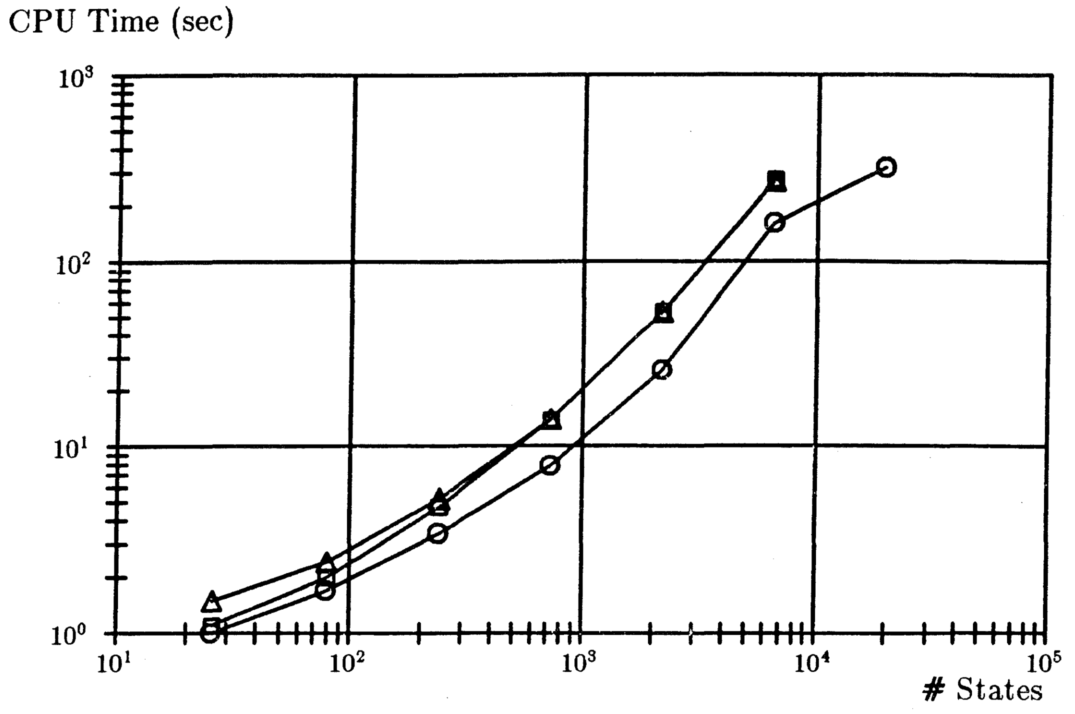
The tools have been developed on a VAX† 11/750 running 4.2bsd UNIX. The tools are highly portable since they use no special features of 4.2bsd UNIX other than the memory allocation routines. Any version of UNIX with a standard C compiler and with memory allocation routines can execute these tools after recompilation. The tools have also been ported to a VAX 11/780 (University of Wisconsin, Madison) running 4.2 UNIX and a VAX 11/750 running LOCUS (UCLA). In both cases they executed without recompilation.

5. Performance

Figures 2 and 3 show some results obtained by analyzing a set of problems. All performance measurements were completed on a lightly loaded VAX 11/750 with 4MBytes of main memory. The same example was analyzed under different assumptions about the model to highlight the time and space savings offered by each version of the RGB. The example used was the classical dining philosophers problem with varying number of philosophers. Table I shows the user time/ system time/ Main Memory requirement for each run. The time measurements increase only slightly when the system is loaded, although the elapsed time increases dramatically under such conditions. The example protocol shown in Figure 1 (594 states) could be analyzed using RGB in 6.2 CPU seconds.

‡ UNIX is a Trademark of the Bell System

† VAX is a registered trademark of Digital Equipment Corporation.



○ Known to be safe

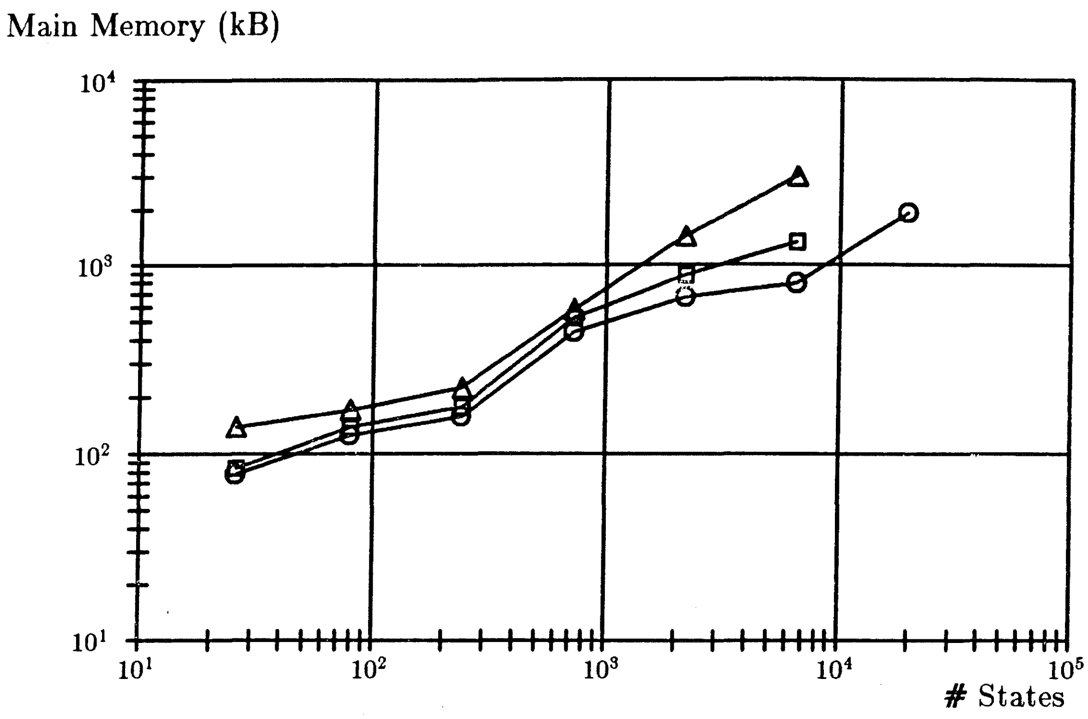
□ Known to be bounded at 127

△ Known to be bounded

Figure 2. CPU Time Consumption as a Function of the Number of States

Conclusions

The most significant benefit gained from the development of P-NUT in general, and the RGB in particular, is that the class of models generally considered analyzable has been expanded through the careful design of simple tools. The RGB has been used to build graphs as large as 20,000 states in less than 7 minutes of CPU time. The efficiency of the RGB, in addition to new tools which present analysis results in understandable terms, have allowed us to analyze significant problems. To date, the tools have been used to analyze some classical problems (e.g., the dining philosophers) and some simple communications protocols (e.g., the alternating-bit protocol), and to analyze CCITT standard protocols (X.21). Current work is exploring the use of the tools in verifying even more complex communications protocols such as TCP/IP and X.25, and in analyzing the performance of some pipelined processors such as Intel's iAPX286.



○ Known to be safe

□ Known to be bounded at 127

△ Known to be bounded

Figure 3. Main Memory Consumption as a Function of Number of States

		Number of dining philosophers						
RGB Version	Measure	3	4	5	6	7	8	9
All	Reachable States	26	80	242	728	2186	6560	19682
Safe	User time	1.0	1.7	3.4	7.8	25.5	160.6	318.1
	System time	0.5	0.8	1.3	2.6	6.4	20.0	67.4
	Main Mem. (kB)	78	126	157	442	674	794	1898
Bound = 127	User time	1.1	2.0	4.7	13.8	53.1	271.6	-
	System time	0.7	0.7	1.2	3.3	7.8	28.2	-
	Main Mem. (kB)	84	138	177	524	877	1316	-
Bounded	User time	1.5	2.4	5.2	13.9	53.1	267.7	-
	System time	1.0	1.2	1.7	3.1	8.3	24.8	-
	Main Mem. (kB)	139	171	224	580	1423	2978	-

Table I. Performance Data for the RGB

References

[Bartlett et.al 69] Bartlett, K.A., R.A. Scantlebury, and P.T.A. Wilkinson, "Note on Reliable Full-Duplex Transmission Over Half-Duplex Links," *CACM*, vol. 12, no. 5, pp. 260-261, May 1969.

[Berthelot G. 82] Berthelot, G. and Richard Terrat, "Petri Net Theory for the Correctness of Protocols," *Protocol Specification, Testing and Verification*, North Holland Pub. Co., (1982).

[Berthomieu B. 83] Berthomieu, B. and Menasche, M. "An Enumerative Approach for Analyzing Time Petri Nets," *Proceedings of the* 1983 *IFIP Congress*, Paris (Sept. 1983).

[Keller R. 76] Keller,R. "Formal Verification of Parallel Programs," *CACM*, Vol. 19, No. 7, July 1976.

[Holliday and Vernon 85] Holliday, M., and M. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," Technical Report #593, Computer Sciences Dept., University of Wisconsin-Madison, May 1985.

[Martinez and Silva 81] Martinez, J. and M. Silva, "A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net", *Proceedings of the Second European Workshop on Application and Theory of Petri Nets*, September 1981.

[Molloy, M. 82] Molloy, M., "Performance Modeling Using Stochastic Petri Nets," *IEEE Trans. on Computers*, vol. C-31, pp. 913-917, Sept. 1982.

[Morgan and Razouk 85] Morgan, E.T, and R. R. Razouk, "Computer-Aided Analysis of Concurrent Systems," To Appear in the *Proceedings of the 5th International Workshop on Protocol Specification Verification and Testing*, Toulouse, FRANCE, June 1985.

[Peterson J. 81] Peterson, J., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1981).

[Ramchandani C. 74] Ramchandani, C. "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Ph.D. Thesis, Project MAC Report No. MAC-TR-120, MIT (1974).

[Ramamoorthy C. 80] Ramamoorthy C.V. and G.S. Ho, "Performance Evaluation of Asynchronous Concurrency Systems using Petri Nets," *IEEE Transaction on Software Engineering*, SE-6, 5 (September 1980), 440-449.

[Razouk R. 84] Razouk, R.R. "The Derivation of Performance Expressions for Communication Protocols from Timed Petri Net Models," *Proceedings of the ACM SIGCOMM'*84 *Symposium on Communications Architectures and Protocols*, June 1984.

[Razouk and Phelps 84] Razouk, R.R. and C. Phelps "Performance Analysis Using Timed Petri Nets," *Proceedings of the 4th International Workshop on Protocol Specification, Testing, and Verification*, June 1984.

[Sifakis J. 77] Sifakis, J. "Petri Nets for Performance Evaluation," *Measuring, Modeling and Evaluating Computer Systems*, Proceedings of the 3rd Symposium, IFIP Working Group 7.3, H. Beilner and E. Gelenbe (eds.), North Holland, 1977, pp. 75-93.

[Symons F. 80] Symons, F.J.W., "Verification of Communication Protocols using Numerical Petri Nets," *Australian Telecommunication Research*, 14,1 (1980) 34-38.

[Vernon, M 82] "The UCLA Graph Model of Behavior: Support For Performance-Oriented Design," Methodologies for Computer System Design, W.K. Giloi and B.D. Shriver (Editors), Elsvier Science Publishers B.V. (North Hollad), pp 47-65.