An Exploratory Study of Program Metrics as Predictors of Reachability Analysis Performance

Albert Timothy Chamillard

CMPSCI Technical Report 95-16
February 24, 1995

email: **chamilla@cs.umass.edu**
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

Abstract

This paper presents the results of an exploratory experiment investigating the use of program metrics to predict reachability graph size, reachability graph generation time, and deadlock check times for concurrent Ada programs. In general, reachability analysis is intractable, so it is important to be able to predict whether or not this analysis technique is feasible given a specific program to be analyzed. We briefly introduce the program representations used to perform the deadlock reachability analysis and describe a set of metrics on those representations. We present empirical results that quantify the value of each metric as a predictor of the reachability graph size and the practicality of deadlock reachability analysis.

1 Introduction

Because concurrent programs are often used for safety-critical applications, developers of such applications need cost-effective techniques they can use to acquire confidence in the reliability of that software. Analysis of concurrent programs is difficult because in many cases the patterns of communication among the various parts of the program are nondeterministic and the number of possible communications is large. One class of techniques that can be used for analysis of concurrent programs is static analysis, which uses information available at compile-time to answer questions about properties of the program being analyzed. For example, static analysis can help determine if it is ever possible for a given program to enter a deadlocked state.

The Laboratory for Advanced Software Engineering Research has constructed a set of tools that support analysis of concurrent Ada programs. One such tool generates the reachability graph for a program, and another tool checks for deadlock in the reachability graph. Reachability analysis is known to be intractable in general [Tay83], so it is important to predict when performing deadlock reachability analysis will be feasible and when alternate static analysis techniques should be used instead. For instance, it would be helpful to have the ability to inspect the program to be analyzed and predict how long the reachability graph generation and deadlock checking will take. Toward this end, we have experimented with a variety of metrics to quantify how well each metric predicts the feasibility of using the reachability graph generation and deadlock checking tools for analysis of a given program.

The next section introduces the program representations we use for our analysis. Section 3 describes a set of metrics on those representations and the measurements we collect in the experiment. Section 4 describes the programs analyzed for the experiment, and Section 5 discusses several constraints imposed on the experiment. Section 6 describes our approach and presents our empirical results. Section 7 closes with some final remarks.

2 Program Representations

Because Ada is one of the few commonly used languages supporting concurrency, and because our current tools are written for analysis of Ada programs, we analyze Ada programs in our experiment. We briefly describe here the principal concurrency constructs in Ada and several sources of nondeterminism in concurrent Ada programs. In Ada programs, potentially concurrent activities occur in *tasks* or procedures; for simplicity, we call them tasks in this paper. Ada tasks typically communicate with each other using a *rendezvous*. In a rendezvous, the calling task makes an *entry call* on a specific *entry* in the called task; the calling task then suspends execution until the called task terminates the rendezvous. The called task executes any statements contained in the *accept body* for the entry (the entry in conjunction with the accept body is called an *accept*), then terminates the rendezvous and continues execution.

Nondeterminism is introduced into an Ada program's execution in several ways. If several tasks make an entry call on the same entry, the called task could rendezvous with any of the calling tasks; we call this *entry nondeterminism*. Another source of nondeterminism in an Ada program's execution is the *select* statement. Select statements have one or more alternatives; when program execution reaches a select statement, the run-time system nondeterministically selects a task for the rendezvous from the alternatives available in a task queue. We call this *select nondeterminism*.

In our analysis approach, we use a variety of representations of a concurrent program to capture information about the program. We use a Task Interaction Graph (TIG) for each task to abstract sequential regions of control flow into single nodes. The nodes in the TIG for a task are connected by edges representing possible interactions (entry calls/accepts) between that task and other tasks in the program. We combine the set of TIGs for all the tasks in a program into a Petri net to model the system as a whole. Finally, we use the Petri net to generate a reachability graph to represent an estimate of all states the program can enter when started in the initial program state. Each of these representations is described briefly below.

Task Interaction Graphs

Long and Clarke [LC89] suggest using Task Interaction Graphs (TIGs) as a concise program representation that retains interaction information. The TIG consists of a finite set of nodes, $N = \{n_i\}$, and a finite set of directed edges, $E = \{e_i\}$. Each node n_i represents a region of sequential code, and each directed edge represents a task interaction (either the start or end of an entry call or an accept). The set of nodes includes a single start node and a set of terminal nodes for the TIG. There is an edge from n_i to n_j if and only if the task can potentially participate in the task interaction represented by the edge, causing the task to exit the sequential region represented by n_i and enter the sequential region represented by n_j. An example program is shown in Figure 1; the TIG for the caller1 task is shown in Figure 2.

```
procedure data is

   task accepter is
      entry entry2;
      entry entry1;
   end accepter;
   task caller1;
   task caller2;
```

```
task body accepter is
begin
   accept entry2;
   accept entry1;
end accepter;
```

```
task body caller1 is
   BranchCond : boolean
begin
   if BranchCond then
      accepter.entry1;
   else
      accepter.entry2;
   end if;
end caller1;
```

```
task body caller2 is
begin
   accepter.entry2;
end caller2;
```

```
begin   -- data
   null;
end data;
```

Figure 1. Example Program

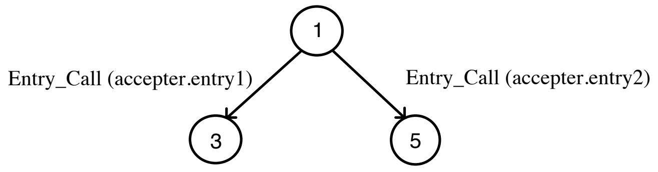


Figure 2. Example TIG

Petri Nets

Petri nets have been proposed as a natural and powerful model of information flow in a system [Pet77]. A Petri net can be represented as a 5-tuple (P, T, I, O, M0). P is the set of places in the Petri net; a place can hold zero or more tokens. If a place holds one or more tokens, the place is said to be *marked*. T is the set of transitions in the Petri net; tokens are moved between places in the net by the *firing* of transitions. A transition can only be fired if it is *enabled*; for a transition to be enabled, each of the input places for the transition must contain at least one token. I is a function mapping places in P to inputs of transitions in T. When a transition fires, a token is removed from each of the places that are inputs to the transition, and a token is deposited in each of the output places of the transition; O is a function mapping places in P to outputs of transitions in T. M0 is a list of all the places in the net that are initially marked.

Petri net modeling appears to be a valuable tool for modeling concurrent software [SC88]; a compact Petri net representation can be generated from the TIGs of a program [DCN94]. To generate the Petri net, we build a place for each node in the set of TIGs of a program. We build a transition for each possible interaction between tasks in the program, where the input places correspond to the sources of the matching entry call and accept TIG edges, and the output places correspond to the targets of the matching entry call and accept TIG edges. To form the initial marking M0 of the Petri net, for each task in the program we put one token in the place that corresponds to the start node of the TIG for that task. For an example Petri net, based on program in Figure 1, see Figure 3. The places in the figure are labeled with the task name and TIG node number from which they were created.

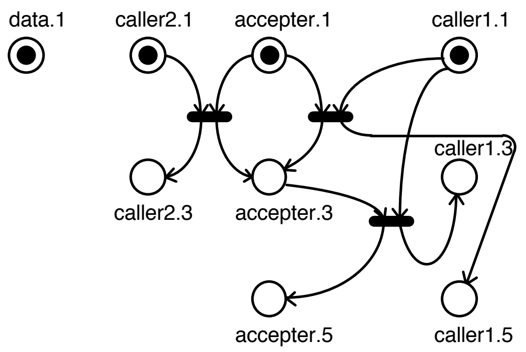


Figure 3. Petri Net

Reachability Graphs

In many cases, we would like our analysis to determine whether or not the concurrent program being analyzed could potentially enter a state in which a specified property holds (deadlock, for instance). A reachability graph enumerates all possible program states and thus can be used to check the property at each state.

A reachability graph for a Petri net consists of a set of nodes, $N = \{n_i\}$, and a set of arcs, $A = \{a_i\}$. Nodes in the reachability graph correspond to markings of the Petri net; the root node of the reachability graph corresponds to the initial marking (M0) of the Petri net. An arc goes from n_i to n_j if and only if the marking of the Petri net can change from n_i to n_j with the firing of a single transition. Although in actuality several interactions, represented by fired transitions, can take place concurrently, we can capture all possible execution sequences by firing a single transition at a time; we use this approach, because the resulting graph is greatly simplified. We note that only markings reachable from the initial marking by some sequential combination of transition firings are included in the reachability graph. It is helpful to observe that a marking of a Petri net simply represents the states of all the tasks being modeled by the Petri net; we therefore consider nodes in the reachability graph as states the program can reach when started from the initial program state. Figure 4 provides the reachability graph for the Petri net in Figure 3. The nodes in the figure are labeled with the Petri net places that are marked in the corresponding program state.

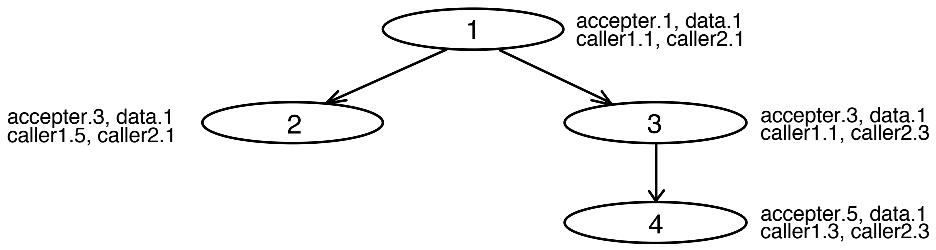


Figure 4. Reachability Graph

3 Metrics and Measurements

This section describes the metrics used as predictor variables and the measurements used as the response variables in the experiment. Our goal is to determine how well each of the predictor variables predicts the values of the response variables.

Metrics

We hypothesize that there are certain characteristics of programs that affect the analyzability of those programs. A program metric is, literally, a measurement of some characteristic of the program. The metrics described below are used to measure characteristics of the TIGs and the Petri net used to represent the program being analyzed; we treat these metrics as predictor variables in the experiment.

We include the number of tasks in the program (T) and the average number of nodes per TIG (N) as predictor variables because the theoretical upper bound on the size of the reachability graph is given by N^T. We also consider N^T as a predictor variable explicitly, and we include the maximum number of nodes in a TIG as a predictor variable, since an outlier could significantly affect the average number of nodes per TIG. Wampler has proposed the metric $N^{T/2}$ as a good predictor of reachability graph size, at least for some programs [Wam85]; we include this metric in our experiment as well.

We suspect that the number of possible communications in a program affects the number of reachable states for that program. We therefore also consider the average number of communications per task and the maximum number of communications in a task as predictor variables. To calculate the number of communications in a task, we add the number of accepts in the task to the number of entry calls in the task. We refer to all the metrics above, with the exception of the Wampler metric, as *TIG metrics* because they can be calculated from the TIGs of a program.

In addition to the metrics above, we also include various characteristics of the Petri net as predictor variables. We include the number of Petri net places because this corresponds to the total number of sequential regions in the program (as opposed to the average or maximum number of sequential regions in single tasks). We also include the number of Petri net transitions as a predictor variable, because each transition represents a possible communication in the program. In addition, we consider the average number of outputs per place in the Petri net and the maximum number of outputs from a place in the Petri net, since these metrics may affect the branching factor in the reachability graph, which could in turn affect reachability graph generation time. We call the measurements of the Petri net characteristics described above *Petri net metrics*. We refer to all the metrics discussed thus far, with the exception of the Wampler metric, as the *syntactic metrics*.

One of the characteristics of concurrent Ada programs that makes them particularly difficult to analyze is nondeterminism. None of the metrics above try to account for nondeterminism in the program being analyzed. Damerla and Shatz [DS92] propose several metrics that we include in our experiment; the metrics are intended to quantify the nondeterminism in Ada programs. α is used to account for the nondeterminism in entries when several tasks can make entry calls on those entries (entry nondeterminism). α is given by $\sum_{i=1}^{e}(Calls_i - 1)$, where e is the number of entries not contained in selects and Calls_i is the number of calls on entry i. The one is subtracted because an entry with only one caller is deterministic. β is used to account for the nondeterministic selection of rendezvous within select statements (select nondeterminism). β is given by $\sum_{i=1}^{s}(Calls_i - 1)$, where s is the number of selects and Calls_i is the number of calls on entries within select i. The one is subtracted because a select with only one call on an entry within the select is deterministic. γ is used to account for total nondeterminism ($\gamma = \alpha + \beta$).

Levine and Taylor [LT93] propose a metric similar to γ called C_{nd} to account for nondeterminism; the difference is that C_{nd} includes entry nondeterminism for all entries (as opposed to only those not in selects) and counts the number of select alternatives with one or more callers when calculating select nondeterminism. Levine and Taylor also propose a metric intended to capture the graph theoretic complexity of the program (C_{gt}); C_{gt} is given by E - N + T + 1, where E is the number of entry calls and accepts in the program, N is the number of TIG nodes in the program, and T is the number of tasks in the program. Another metric, C_{if}, is proposed for capturing the communication structure or information flow for the tasks comprising the program. C_{if} is given by $\left(\sum_{i=1}^{T}(in - edges_i)(out - edges_i)\right)^{\ln T}$, where T is the number of tasks in the program, in-edges$_i$ is the sum of task entries and shared variables read in task i, and out-edges$_i$ is the sum of entry calls and shared variables written by task i. C_{nd}, C_{if}, and C_{gt} are also included in our experiment. We refer to the metrics proposed by Wampler, Damerla and Shatz, and Levine and Taylor as the *literature metrics*.

Measurements

We consider a variety of measurements as response variables in the experiment; these measurements have been chosen as indicators of the feasibility of reachability analysis on the program to be analyzed. The measurements can be broken into two categories: size measurements and time measurements.

The size measurements include the number of nodes in the reachability graph and the number of arcs in the reachability graph. Because we check for deadlock on each node in the reachability graph and each possible program state is represented by a reachability graph node, we include the number of reachability graph nodes as a response variable. We include the number of reachability graph arcs as another basic measure of reachability graph size.

Certain time measurements may be better estimators of the feasibility of reachability analysis than reachability graph size. Because the reachability graph must be generated before we perform our deadlock analysis, we include the amount of time it takes to create the reachability graph as a response variable. To determine if deadlock is possible in the program, we perform deadlock analysis on the reachability graph; we therefore also include the time to check for deadlock in the reachability graph as a response variable.

4 Programs Analyzed

We conduct our experiment on various sizes of three problems commonly used in the literature for concurrency analysis: dining philosophers, the gas station problem, and readers/writers.

The dining philosophers problem consists of a certain number of philosophers sitting around a table, with a single fork between a philosopher and the neighbor to their left and a single fork between that philosopher and the neighbor to their right. Each philosopher thinks for a while, then picks up both forks to eat, then puts the forks back down and thinks some more. Because the forks between the philosophers are shared, it is not possible for all the philosophers to eat at the same time. Our solution for the dining philosophers problem uses a task for each fork and a task for each philosopher.

The gas station problem models a gas station. The problem includes a pump, an operator, and a number of customers using the gas station. A customer who wishes to pump gas must first prepay money to the operator, who then activates the pump. The customer then pumps the gas and returns to the operator, who gives the customer change based on how much gas was pumped. Our solution for the gas station problem uses a task for the pump, a task for the operator, and a task for each customer.

The readers/writers problem includes a set of readers and a set of writers; conceptually, the readers and writers are accessing the same document. If there are no readers or writers accessing the document, a writer can access the document to write; only a single writer (and no readers) can be accessing the document at any given time. If there are no writers accessing the document, a reader can access the document to read it; multiple readers (and no writers) can be accessing the document at any given time. Our solution for the readers/writers problem uses a task for each reader, a task for each writer, and a single task to enforce the constraints on access to the document.

5 Experimental Constraints

Before discussing the results of our experiment, we should note that our study imposes various limitations on our experimental design and the generalizability of our results.

This experiment is clearly observational, since we do not have the capability to manipulate the metrics used as predictor variables. Given the potentially subtle dependency relationships between the metrics, it is not clear how we could hold certain metrics constant while varying others. We therefore conduct an observational experiment rather than trying to manipulate the metrics. The drawback, of course, is that there may be latent factors that affect both metrics and measurements, which could lead us to infer predictive relationships that do not actually exist. If we were performing a manipulation experiment, we could exert experimental control to solve the latent factor problem; in an observation experiment, we can not solve this problem.

We also do not know if our empirical results are generalizable. The population of concurrent Ada programs to which we have access is fairly limited in size, which is why we perform the experiment on various instances of the academic problems described above. Unfortunately, there is no evidence that these programs are representative of the population of "real" Ada programs. This means that we are not able to make general inferences about the metrics as predictors of feasibility, because our sampling from the population of programs is not random. On the other hand, we can use the predictor/response variable relationships discovered here as a point of comparison when we do gain access to other, more realistic, concurrent programs.

6 Empirical Results

Recall that our goal is to determine how well (or poorly) each of our metrics predicts the performance of deadlock reachability analysis, both in terms of reachability graph size and the times to generate the reachability graph and check for deadlock. We first generate the metric and measurement data, using parameter estimation techniques to quantify our confidence in certain measurements. We then use Pearson's Correlation Coefficient to examine relationships within metric/measurement pairs. For selected metric/measurement pairs, we use randomization tests to determine the probability of the given correlation if the metric and measurement were actually independent. We also utilize multiple regression techniques to generate standardized regression coefficients, which we then use to estimate the relative strengths of each metric as a predictor of the measurement values. Finally, we develop several causal models using the ω heuristic described in [Coh94].

Parameter Estimation

Most of the metric and measurement values for the experiment are constant for a given program; in other words, the values will be exactly the same every time they are calculated for that program. For example, the number of tasks for our solution to the dining philosophers problem with two philosophers is always four. The two exceptions to this constancy are the reachability graph generation time and the deadlock check time. Each of these values can be affected by a variety of factors, such as system load during the execution of the trials, so we must examine these values closely to ensure we are using a reasonably accurate estimate of the times in our data analysis.

The method for estimating our confidence in the reachability graph generation and deadlock checking times is identical, so we explain in the context of reachability graph generation time. To calculate reachability graph generation time, we generate the reachability graph 30 times and take the mean, $\bar{x}$, of the 30 execution times as the estimate of graph generation time. We use the standard deviation, s, of the set of 30 execution times to calculate the estimated standard error of the mean, which is given by $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{N}}$. We then calculate the confidence interval as $\bar{x} \pm 1.96 * \hat{\sigma}_{\bar{x}}$. The 1.96 value is from a Z-distribution for confidence at the 0.05 level for a two-tailed test. The sample is large enough to justify using the Z-distribution value for the multiplier, and since we don't know if the sample mean is higher or lower than the population mean of graph generation times, the two-tailed multiplier is required. Results of these computations for reachability graph generation time can be found in Figure 5; using the same method for deadlock check time yields the results in Figure 6. In these figures, gasW indicates an instance of the gas station problem with W customers, philX indicates an instance of the dining philosophers problem with X philosophers, and rwYZ indicates an instance of the readers/writers problem with Y readers and Z writers.

	mean	standard dev	standard error	confidence interval
gas1	0.184	0.010	0.002	(0.182, 0.186)
gas2	2.016	0.015	0.003	(2.010, 2.022)
gas3	17.357	0.051	0.009	(17.339, 17.375)
gas4	119.643	0.177	0.032	(119.580, 119.706)
gas5	738.470	1.787	0.326	(737.831, 739.109)
phil2	0.245	0.014	0.003	(0.239, 0.251)
phil3	1.409	0.015	0.003	(1.403, 1.415)
phil4	8.368	0.033	0.006	(8.356, 8.380)
phil5	47.134	0.099	0.018	(47.099, 47.169)
phil6	256.410	1.987	0.363	(255.699, 257.122)
rw11	0.259	0.001	0.000	(0.259, 0.259)
rw22	6.248	0.034	0.006	(6.236, 6.260)
rw23	27.777	0.027	0.005	(27.767, 27.787)
rw32	25.740	0.025	0.005	(25.730, 25.750)
rw33	111.656	0.181	0.033	(111.591, 111.721)

Figure 5. Reachability Graph Generation Time Confidence Intervals

	mean	standard dev	standard error	confidence interval
gas1	0.338	0.001	0.000	(0.338, 0.338)
gas2	0.650	0.017	0.003	(0.644, 0.656)
gas3	1.447	0.026	0.005	(1.437, 1.457)
gas4	4.619	0.020	0.004	(4.611, 4.627)
gas5	108.397	1.413	0.258	(107.891, 108.903)
phil2	0.265	0.001	0.000	(0.264, 0.266)
phil3	0.461	0.011	0.002	(0.458, 0.464)
phil4	1.007	0.010	0.002	(1.004, 1.010)
phil5	3.233	0.016	0.003	(3.227, 3.239)
phil6	85.829	1.099	0.201	(85.436, 86.222)
rw11	0.327	0.002	0.000	(0.326, 0.328)
rw22	0.578	0.008	0.001	(0.575, 0.581)
rw23	1.112	0.019	0.004	(1.105, 1.119)
rw32	1.086	0.021	0.004	(1.079, 1.093)
rw33	2.992	0.029	0.005	(2.981, 3.003)

Figure 6. Deadlock Check Time Confidence Intervals

We see that the largest confidence interval, which represents the most uncertainty in our estimate of execution time, for graph generation time is only approximately 4% of the mean. For deadlock check time, the largest confidence interval is only approximately 2% of the mean. We can therefore assume, for both of these measurements, that our estimates of the execution times are very close to the actual mean execution time.

Pearson's Correlation Coefficient

Pearson's Correlation Coefficient provides an estimate of the linear relationship between two variables x and y. The coefficient ranges from -1.0 to 1.0, with a coefficient magnitude close to 1.0 indicating a strong relationship and a magnitude close to 0.0 indicating no linear relationship. We note that a low correlation only indicates that the variables are not linearly associated; they could still be related in some non-linear way.

We calculate the correlation between each of our metrics and each measurement. If we calculate a high correlation, we have at least some indication that the metric is a good predictor of the measurement. These correlations are provided in Figure 7 below.

	number of reachability graph nodes	number of reachability graph arcs	reachability graph generation time	deadlock check time
number of tasks (T)	0.5795	0.6869	0.3575	0.5158
average TIG nodes per task (N)	0.4301	0.2499	0.5326	0.3834
N^T	0.7340	0.8349	0.4913	0.7820
maximum TIG nodes in a task	0.5212	0.3328	0.6653	0.4492
average edges per task	0.4527	0.2731	0.5458	0.4064
maximum edges in a task	0.5241	0.3392	0.6726	0.4474
# of Petri net places	0.8076	0.7324	0.7412	0.7236
# of Petri net transitions	0.6276	0.5329	0.7297	0.5064
average Petri net place outputs	0.0588	0.0715	0.1622	-0.0257
maximum Petri net place outputs	0.1615	0.2189	0.2150	0.0596
Wampler	0.8600	0.9088	0.6568	0.8728
α	0.6886	0.6072	0.6008	0.6512
β	0.0153	0.0585	0.1052	-0.0777
γ	0.7903	0.7371	0.7718	0.6649
C_{nd}	0.7903	0.7371	0.7718	0.6649
$C_{\sigma t}$	-0.4997	-0.3082	-0.6292	-0.4405
C_{if}	0.7648	0.5750	0.9192	0.7245

Figure 7. Pearson's Correlation Coefficients

We infer from the table above that, for the syntactic metrics, the number of tasks (T), N^T, and the number of Petri net places and transitions may be the best predictors of the number of nodes in the reachability graphs, because they have the strongest linear relationship with this measurement. These metrics also have the highest correlations with the number of reachability graph arcs. For the literature metrics, the Wampler metric, γ (which is identical to C_{nd} for the programs in this experiment), and C_{if} have the highest correlations with the size measurements. The number of Petri net places and transitions have the strongest correlation with reachability graph generation time, although the maximum number of edges in a task and the maximum number of TIG nodes in a task seem to be linearly related to this time as well. For deadlock check time, N^T and the number of Petri net places appear to be the best predictors; T and the number of Petri net transitions have a weaker linear relationship with this time. The Wampler metric, C_{if}, and γ are most highly correlated with the time measurements for the literature metrics. In a later subsection, we will see how standardized regression coefficients can help us estimate the relative strengths of these metrics as measurement predictors; first, however, we examine the probability of calculating certain correlations under the null hypothesis that the metric and measurement are independent.

Randomization Tests

We know that we can use randomization tests, in conjunction with correlation, to test the hypothesis that two samples are dependent [Coh94]. We can therefore select a metric and a measurement, perform a randomization test, and determine the probability of calculating the given correlation under the null hypothesis that the metric and the measurement are independent. If the probability is low ($p < 0.05$ is typically an acceptable level), we can reject the null hypothesis with high confidence, i.e., we can state that there is a dependence between the metric and the measurement with only a small probability that we are wrong. We note that randomization tests do not provide results that are generalizable to populations, but we observed in Section 5 that our results are probably not generalizable for other reasons, so limiting our attention to the samples does not represent a severe limitation to us.

We have performed randomization tests on a variety of metric/measurement pairs, where the pairs have been selected to give a range from a very low correlation to a very high correlation between the metric and the measurement in the pair. The results of these tests are provided in Figure 8, where the p values in the table represent the probability that we reject the null hypothesis incorrectly.

Most of the results in the table below are as we expected. The p values for the metric average Petri net place outputs are high, indicating a low dependency between this metric and the measurements; this can also be seen in the extremely low correlations for these metric/measurement pairs. The results for the number of Petri net places metric are also unsurprising; the p values indicate a strong dependency in the metric/measurement pairs, as do the high correlation values. Similarly, the result for the number of tasks metric is as expected. The correlations for the Wampler metrics are very unlikely to have occurred by chance (particularly notable is the p value for # of reachability graph nodes), as are the correlations for γ and C_{if}.

metric	measurement	p value
T	deadlock check time	0.051
N^T	# of reachability graph nodes	0.059
N^T	# of reachability graph arcs	0.071
N^T	reachability graph generation time	0.052
N^T	deadlock check time	0.084
# of Petri net places	# of reachability graph nodes	0.034
# of Petri net places	deadlock check time	0.001
# of Petri net transitions	deadlock check time	0.034
average Petri net place outputs	# of reachability graph nodes	0.271
average Petri net place outputs	# of reachability graph arcs	0.232
average Petri net place outputs	reachability graph generation time	0.151
average Petri net place outputs	deadlock check time	0.362
Wampler	# of reachability graph nodes	< 0.001
Wampler	# of reachability graph arcs	0.003
Wampler	deadlock check time	0.038
γ	# of reachability graph nodes	0.035
γ	deadlock check time	0.048
C_{if}	reachability graph generation time	0.037

Figure 8. Randomization Test Results

There are certainly some surprising results in the table, however. The number of Petri net transitions has only a 0.5064 correlation with deadlock check time, but we can reject the null hypothesis of independence at better than the 0.05 level ($p = 0.034$). The results for the N^T metric are even more striking. The correlation between N^T and the number of reachability graph arcs and the deadlock check time are higher than for any other metric (except Wampler), yet we cannot reject the null hypothesis of independence at the 0.05 level for these pairs. The correlation between N^T and the number of reachability graph nodes is also high, but the p value does not let us reject the null hypothesis at the 0.05 level for this pair either. We are closest to rejecting the null hypothesis for the N^T- reachability graph generation time pair, but this pair has the fourth lowest correlation for pairs considering the generation time.

Multiple Regression

Thus far, we have tried to consider correlations within metric/measurement pairs. While this approach is statistically sound, it does not account for dependencies between the metrics in a robust manner. Since many of the dependency relationships between our metrics are unidentified or subtle, we would like to account for these dependencies in our analysis of predictive power. Multiple regression techniques let us implicitly account for these dependencies.

In multiple regression, we determine the relative effect of each predictor (metric) on the response variable (measurement); the effect is reflected in the regression coefficient for each metric. To make the regression coefficients comparable among the metrics, we first standardize each metric variable, which in essence puts each variable on the same scale. We then perform the multiple regression, which yields a set of standardized regression coefficients. The standardized regression coefficients for our syntactic metrics and measurements are provided in Figure 9; note that larger coefficients (positive or negative) indicate stronger predictive power. Because of dependencies between our syntactic metrics and the literature metrics, we were unable to include the latter in these multiple regressions.

	number of reachability graph nodes	number of reachability graph arcs	reachability graph generation time	deadlock check time
T	29.7821	15.6022	54.6693	60.9877
N	23.8965	29.3481	23.0003	12.1439
N^T	1.2576	1.0427	1.6608	2.1662
maximum TIG nodes in a task	38.1620	-12.5045	101.4456	136.2092
average edges per task	-17.8429	-21.9048	-15.2004	-6.1416
maximum edges in a task	-14.8690	19.3744	-54.1592	-79.8170
# of Petri net places	-41.9813	-22.6861	-77.0609	-85.4558
# of Petri net transitions	5.8388	6.3527	7.6067	5.4942
average Petri net place outputs	-2.2517	-3.8478	-1.8145	-0.0834
maximum Petri net place outputs	-6.8775	-3.8852	-11.7384	-12.3938
R^2	0.9977	0.9947	0.9930	0.9899

Figure 9. Standardized Regression Coefficients (Syntactic Metrics)

The results above are somewhat counter-intuitive. Some of the metrics (such as number of Petri net places) have high correlations and high regression coefficients, which is as we expect; on the other hand, the N^T metric has extremely high correlations with measurements but very low regression coefficients, which is exactly the opposite of what we expect. This phenomenon can be explained through further consideration of the correlation and the regression coefficients. Correlation simply measures the linear relationship between two variates; it does not determine how much of the effect is direct and how much is indirect. The regression coefficient, however, reflects the amount of direct effect of one variate on the other. Therefore, a variate can have a high correlation with another variate and a low regression coefficient if most of the correlation is indirect through other variates.

We should also note that the R^2 values below are extremely high. For each measurement, we can use a set of simple syntactic metrics to form a predictive equation that accounts for 99% of the variance in the measurement.

Because metric inter-dependencies would not let us include the metrics from the literature in this multiple regression, we also performed multiple regressions of the four measurements on the Wampler metric, γ, C_{gt}, and C_{if}; the results can be found in Figure 10. We note that the R^2 values are also very high for these regressions, though slightly lower than for the syntactic metrics. This is interesting, because in general these metrics are intended to capture more subtle program characteristics than the syntactic metrics, yet they provide slightly weaker predictive power.

	number of reachability graph nodes	number of reachability graph arcs	reachability graph generation time	deadlock check time
Wampler	0.5917	0.6798	0.3040	0.7253
γ	0.1399	0.2532	0.0705	-0.1370
C_{gt}	0.1897	0.2848	0.2555	0.1992
C_{if}	0.5956	0.3693	0.9613	0.7080
R^2	0.9801	0.9290	0.9770	0.9671

Figure 10. Standardized Regression Coefficients (Literature Metrics)

Causal Modeling

We can use the results of our multiple regression to build equations for the measurements considered in the experiment, and these equations can account for a large part of the variance in the given measurement. Recall that this was the goal of the experiment - to quantify the value of each metric as a predictor of each of the measurements. The predictive power of each metric is reflected in the regression coefficient in each measurement equation. The measurement equation is in effect a causal model, though a very "shallow" one; all metrics are on the left, and the measurement is on the right. To better our understanding of the interactions between the metrics and their effect on the measurement, we can build a "deeper" causal model that more accurately reflects the effects of each metric on other metrics and the measurement. In this subsection we develop deeper causal models for deadlock check time based on the syntactic metrics and deadlock check time based on the literature metrics.

There are two problems we must consider when we form a causal model: the latent factor problem and the causal ordering problem. Because we have performed an observational experiment, we can not exert experimental control to account for latent factors. Pragmatically, however, we note that our multiple regressions above account for at worst 93% of the variance in the measurement variates. We therefore assert that, if there are latent factors, including them in our predictive equation will not effect our predictive power significantly (though doing so could certainly cause significant changes in the regression coefficients). The causal ordering problem is not very difficult in this experiment. We know that the TIGs for a program are used to develop the Petri net, so the causal direction is from TIG metrics to Petri net metrics. Similarly, the metrics from the literature are calculated from the set of TIGs for a program, so the causal direction is from TIGs metrics to the literature metrics. The Petri net is used to generate the reachability graph, so causal arrows lead from Petri net metrics to reachability graph measurements. The reachability graph is used to check for deadlock, so any correlation between reachability graph size measurements and deadlock check time are reflections of a causal effect of reachability graph size on deadlock check time.

Because we already understand the causal ordering in our model, our main effort is expended on quantifying the causal effects in the model to build a set of structural equations [Coh94] for each of the models. We loosely follow Cohen's approach using the ω statistic, which measures the proportion of indirect effect of one variate on another, though we have made some methodological changes. In Cohen's approach, all variables in the model are used at each step to develop causal links. In contrast, if we are developing causal effects on the number of Petri net places metric (for instance), we only include the TIG metrics in our analysis at this step; we perform this modification to the approach because of our a priori knowledge of causal ordering relationships. Our other modification concerns the selection of the variables to include in each structural equation. In Cohen's approach, variables are included if their ω score is less than a given threshold and the regression coefficient for that variable is above a certain threshold. We, however, include enough of the variables considered to give each developed structural equation an R^2 greater than 0.80. In other words, we concern ourselves more with the predictive power of each structural equation than with the direct effects of each variable in the equation.

The results of our modeling efforts for the selected metrics and measurements are shown in Figures 11 and 12. Note that a larger value (positive or negative) on a causal arrow indicates a stronger causal effect. When we use the metrics from the literature in the model, we do not include Petri net metrics, since the literature metrics are calculated directly from TIGs.

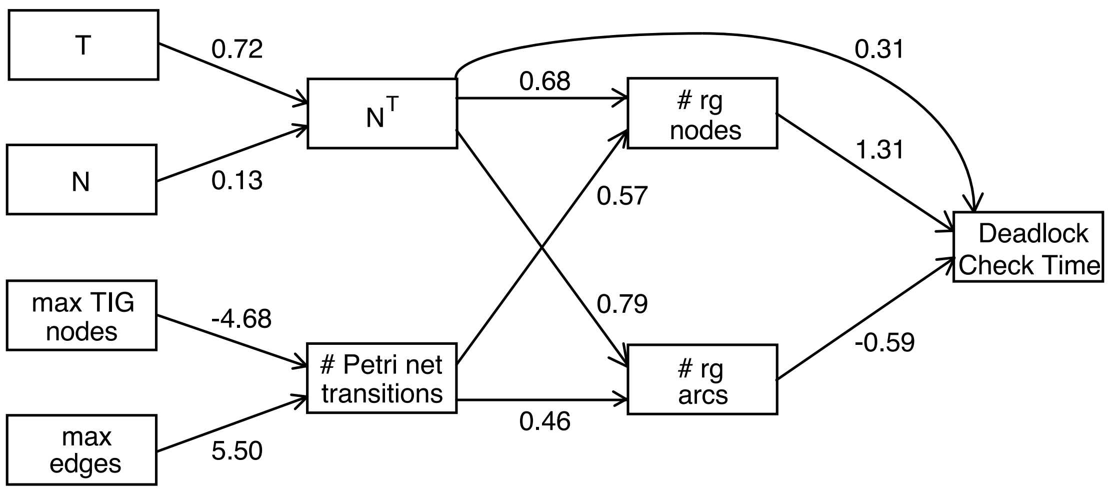
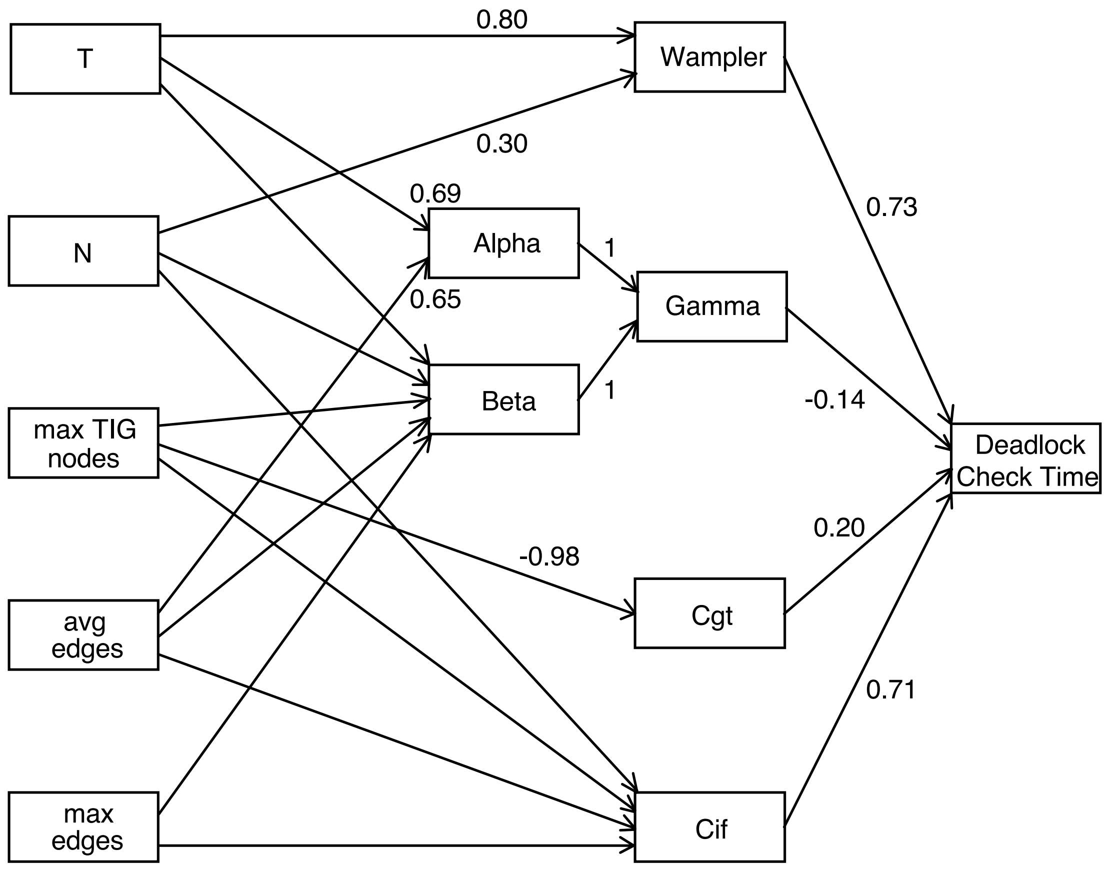


Figure 11. Causal Model for Syntactic Metrics



Beta = 0.51 T + 15.59 N - 10.39 (max TIG nodes) - 14.93 (avg edges) + 9.95 (max edges)
Cif = -7.05 N + 17.06 (max TIG nodes) + 3.54 (avg edges) - 12.98 (max edges)

Figure 12. Causal Model for Literature Metrics

In the causal model for the syntactic metrics, it is interesting to note the direct and indirect effects of N^T on deadlock check time. The indirect effect through reachability graph size is as we would expect from our multiple regression results, while the direct effect appears to be a consequence of our deadlock checking algorithm. The strength of the causal arrow from the number of reachability graph nodes to deadlock check time is unsurprising, since we check for deadlock on each node. More surprising is the fact that the number of Petri net transitions is the only Petri net metric included in the model. We would not expect this to be true, given the relatively low standardized regression coefficients for this metric compared to the standardized regression coefficients for the number of Petri net places; it is not clear why this occurs.

In the causal model for the literature metrics, we note that despite the strong effect of the Wampler and C_{if} metrics, all four metrics must be included as direct effects on deadlock check time to achieve $R^2 > 0.80$. We also point out that the structural equations for β and C_{if} are particularly complicated. One possible explanation for this is that the TIG metrics included in the experiment do not easily capture the characteristics measured by β (select nondeterminism) and C_{if} (information flow), so many of the TIG metrics must be included in the structural equations to achieve a sufficiently large R^2.

7 Conclusions

We have determined that certain sets of metrics, such as the syntactic and literature metrics examined here, can provide good predictive models for reachability analysis performance, at least for deadlock. We note that the metrics from the literature provide slightly weaker predictive power than the purely syntactic metrics for the programs used in this experiment. We have also developed several causal models for deadlock check time to better quantify the effects of our metrics on the other metrics and measurements considered in the experiment.

We intend to continue this work by experimenting with some "real" programs from the private sector and the military. Experimentation on academic problems can provide us with preliminary causal models, but since we do not know how well these problems represent the population of concurrent Ada programs, we can not immediately generalize our results. As we gain access to more programs, we can test our causal hypotheses on these programs and tune our causal models accordingly.

In this paper, we restrict our attention to using metrics to predict the feasibility of reachability analysis for checking deadlock. Many other properties of concurrent programs are also interesting, such as enforcement of mutual exclusion or freedom from critical races; these properties could be considered in follow-up experiments. It would be interesting to determine whether the type of property considered in the analysis has an effect on the multiple linear regression results or the causal models. There are also a wide variety of other static analysis techniques available for use on concurrent programs; it would be instructive to run similar experiments on each of these techniques as well. We may find that the metrics provide different predictive models for each technique, which could in turn lead to some heuristic guidance for selection of the most effective tool for analysis of a given program.

Acknowledgements

Many thanks are due to Lori Clarke, whose diligent reviews led to significant improvements in this paper.

References

[Coh94] Paul Cohen. *Empirical Methods for Computer Science*. To be published.

[DCN94] Matthew B. Dwyer, Lori A. Clarke, and Kari A. Nies. A compact petri net representation for concurrent programs. Technical Report TR 94-46, University of Massachusetts, Amherst, 1994.

[DS92] Srinivasarao Damerla and Sol M. Shatz. Software complexity and Ada rendezvous: Metrics based on nondeterminism. *Journal of Systems and Software*, 17(2):119-127, February 1992.

[LC89] Douglas L. Long and Lori A. Clarke. Task interaction graphs for concurrency analysis. In *Proceedings of the 11th International Conference on Software Engineering*, pages 44-52, Pittsburgh PA, May 1989.

[LT93] David L. Levine and Richard N. Taylor. Metric-driven reengineering for static concurrency analysis. In *Proceedings of the 1993 International Symposium on Software Testing and Analysis (ISSTA)*, pages 40-50, Cambridge MA, June 1993.

[Pet77] James L. Peterson. Petri nets. *Computing Surveys*, 9(3):223-252, September 1977.

[SC88] S.M. Shatz and W.K. Cheng. A petri net framework for automated static analysis of Ada tasking behavior. *The Journal of Systems and Software*, 8(5):343-359, December 1988.

[Tay83] Richard N. Taylor. Complexity of analyzing the synchronization structure of concurrent programs. *Acta Informatica*, 19:57-84, 1983.

[Wam85] Gordon Kent Wampler. Static concurrency analysis of Ada programs. Master's thesis, University of California, Irvine, 1985. As cited in [LT93].