A Family of Coverage Testing Criteria for Coloured Petri Nets

Adenilso da Silva Simão *
Simone do R. S. de Souza ‡
José Carlos Maldonado *

* Departamento de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

São Carlos — São Paulo
{adenilso,jcmaldon}@icmc.usp.br

‡ Departamento de Informática
Universidade Estadual de Ponta Grossa

Ponta Grossa — Paraná
srocio@uepg.br

Abstract

In this paper, we propose a family of specification-based coverage testing criteria for Coloured Petri Nets (CPNs). CPNs are an extension of Petri Nets with the capability of expressing, defining and handling datatypes and structured values. The main approaches used for testing and validation of CPNs based on specifications are simulation and analysis of properties. However, these approaches do not essentially provide a mechanism for quantifying the testing activity, compromising the testing quality assessment. The coverage criteria family proposed in this paper aims at complementing this scenario by providing mechanisms either to evaluate the adequacy of test sequences (e.g., sequences generated by simulation) or to guide the generation of adequate test sequences with respect to a given criterion. The underlying model to derive the requirements of the testing criteria is the occurrence graph. The concepts and criteria presented in this paper are supported by a prototyping tool. Some examples are provided to illustrate the main ideas.
Keywords: Coloured Petri Nets, Testing Criteria, Software Quality

1. Introduction

Outstanding as one of the most important software quality assurance activities, software testing is also one of the most expensive and most unsystematic. Moreover, despite the continuous and successful efforts of both academy and industry to devise methods and mechanisms to aid in this task, a general lackness of supporting tools can be observed. A crucial issue in this activity is the establishment of a testing strategy to be applied during software development and maintenance that trades off cost and efficacy in revealing critical errors. Since the earlier the errors are detected in the life cycle, the less onerous is the process of removing them, specification testing is an effective way of reducing the development costs. The specification testing aims at validating the system specifications against the requirements, as earlier as possible.

The authors would like to thank the Brazilian funding agencies FAPESP, CAPES and CNPq for their support to this research.

Two main research issues posed in the testing area are 1) how to select test cases and 2) how to assure that a program P or a specification S has been tested enough. These issues have been addressed by the establishment of testing criteria, which systematize the requirements to be concerned with in the testing and embody the desirable features of a "good" test case suit.

For safety critical applications, errors can produce disastrous consequences. Therefore, the use of formal techniques is strongly recommended, if not mandatory. Reactive Systems (RSs) are a class of systems whose main feature is their interaction with the environment reacting to stimuli. Typical examples of RSs are metro control, patient hospital monitoring, and communication protocols. Concerning these systems, software quality is even more relevant, for failures can provoke economic and/or human losses, making specification and testing activities much more critical. In this context, state transition based specification techniques, such as Finite State Machines (FSMs) [11], Statecharts [13] and Petri Nets [22], have been used for the specification of the behavioral aspect of these systems. Testing and validation of state transition based system specifications are done, in general, by reachability analysis [20, 22], simulation [14] and by test sequences generation methods [2, 9, 23, 26].

Coloured Petri Nets (CPNs) are a formal technique that has been intensively employed in the specification of systems in which the concurrency and parallelism among processes play an important role in the overall behavioral aspects [17, 18]. CPNs have well defined syntax and semantics, allowing analysis of the dynamic aspects of the model. The components of the system are modeled by a set of hierarchical pages, expressing the synchronization, parallelism and causal dependence or independence. According to Harel [14], the consistence and completeness verification of a model does not necessarily avoid the occurrence of logical errors. Simulation allows to observe how the model behaves, while it reacts to the occurrence of pre-defined or randomly generated events [17]. Analysis of properties is another alternative and allows evaluation of dynamic properties of the model, as deadlock freeness, marking reachability and validity of event sequences. However, the quality of the testing activity is by itself an issue in the software development process and neither simulation nor property analysis provides direct mechanisms for quantifying the testing and validation activities.

Considering software testing in general, one way to quantify the quality of a test case suit (and, thus, the testing activity) is to use coverage measures based on a testing criterion. Therefore, complementing current CPN validation approaches, we propose a CPN Coverage Criteria Family (CPNCCF) for validation of CPN based specifications. These criteria provide mechanisms both to evaluate test sequences (e.g. sequences generated by simulation) and to guide the generation of by-construction-adequate test sequences (with respect to the criteria). These criteria provide a coverage measure to quantify the testing activity and thus contribute to the improvement of the quality of this activity in the context of specifications. Application of these coverage criteria is done using a behavioral representation of the CPN called occurrence graph [17]. The occurrence graph shows the possible global states and the paths — sequences of markings — that can occur in a CPN.

This paper is organized as follows. In Section 2 related works are discussed. In Section 3 the basic concepts of CPN that are the basis for understanding the criteria proposed herein are presented. In Section 4 we define the coverage criteria and illustrate the application of the coverage criteria to assess the quality of a specification testing activity. Concluding remarks are presented in Section 5.

2. Related Work

Motivated by the fact that the traditional testing techniques are not adequate for testing some features introduced by concurrent/parallel programming such as non-determinism and concurrency, many researchers have developed specific testing techniques addressing these issues [3, 5, 6, 19, 23, 24, 27–33].

Yang and Chung [33] introduced the path analysis testing of concurrent programs. Given a program, two models are proposed:

(i) *task flowgraph* — corresponds to the syntactical view of the task execution behavior and models the task control flow; and

(ii) *rendezvous graph* — corresponds to the run-time view and models the possible rendezvous sequences among tasks.

An execution of the program will traverse one concurrent path of the rendezvous graph (*C-route*) and one concurrent path of the flowgraph (*C-path*). A method called *controlled execution* is presented to support the debugging activity of concurrent programs. They pointed out three research issues to be addressed to make practical their approach: C-path selection, definitive test generation and test execution.

Taylor et al. proposed a set of structural coverage criteria for concurrent programs based on the notion of concurrent states and on the concurrency graph [30]. Five criteria are defined: *all-concurrency-paths*, *all-proper-cc-histories*, *all-edges-between-cc-states*, *all-cc-states* and *all-possible-rendezvous*. The hierarchy (defined by the subsumption relation [8, 34]) among these criteria is analyzed. They stress that every approach based on reachability analysis would be limited in practice by state space explosion. They mentioned some alternatives to overcome the associated constraints.

In the same vein of Taylor and colleagues' work, Chung et al. [3] proposed four testing criteria for Ada programs: *all-entry-call*, *all-possible-entry-acceptance*, *all-entry--call-permutation* and *all-entry-call-dependency-permutation*. These criteria focus the rendezvous among tasks. They also present the hierarchy among these criteria.

Koppol and Tai introduced an incremental approach to structural testing of concurrent programs based on the hierarchy of processes. They claimed to alleviate the state explosion problem besides other advantages. Their underlying model is the Labeled Transition Systems (LTS) [19] that in fact is the reachability graph of each task of the concurrent program.

In another line of work, aiming at demonstrating that, with some extensions, sequential test data adequacy criteria are still applicable to parallel program testing, Yang et al. extended the data flow criterion all-du-path [25] for parallel programs [32]. A Parallel Program Flow Graph is constructed and is traversed to obtain the du-paths. All du-paths that have definition and use of variables related to parallelism of threads constitute test requirements to be exercised. Threads are independent sequences of execution within a parallel program. The *della pasta* tool (Delaware Parallel Software Testing Aid) automates their approach.

Probert and Guo [24] introduced the approach E-MPT (Estelle-directed Mutation-based Protocol Testing) that applies Mutation Testing [4] to validate the behavior of Estelle specifications. In fact, their approach addresses the validation of the Extended Finite State Machines defined by the specification. Two mutation types are defined: major

mutation — which tests the basic structures of Estelle — and minor mutation — which tests the correctness of the operations associated to the transitions. The generation of the mutant specifications is based on a Finite Complete Set of Alternatives, which possesses, for each element that can suffer a mutation (e.g., variables, constants and mathematical operators), its syntactically correct alternatives based on the specification under testing.

Souza et al. [28] extend the work of Probert and Guo, defining a mutation operator set for Estelle, which can be taken as a fault model for this technique. The mutation operator set takes in consideration the intrinsic features of Estelle, such as: parallelism, communication and dynamic structures. They are divided in three categories: Module Mutation, Interface Mutation and Structure Mutation. A strategy for application of the Mutation Testing is proposed, making possible to conduct the validation activity, giving priority for specific types of errors.

Fabbri et al. defined Mutation Testing to validate specifications based on Finite State Machines (FSMs) [5], Statecharts [7] and Petri Nets [6]. For each specification technique a mutation operator set has been defined inspired in the error classification suggested by Chow [2]. For the Statecharts technique, abstraction strategies were proposed to allow the selection of its basic components — EFSM-Extended Finite State Machines — at each hierarchical level. The mutation score defines a coverage measure for assessing the quality of a given test suite.

Souza et al. proposed coverage testing criteria for both specifications based on Statecharts — Statechart Coverage Criteria Family (SCCF) [29] — and specifications based on Estelle — Estelle Coverage Criteria Family (ECCF) [27]. These criteria emphasize intrinsic features of each specification technique and are based on control-flow information. SCCF and ECCF criteria can be used either to evaluate test sequences or to guide the generation of test sequences with respect to a given criterion. Reachability tree is the underlying model used to derive the requirements of these testing criteria. A well-known problem related with reachability tree is the state explosion. Some approaches to minimize this limitation can be used during the construction of these trees, in order to reduce their size to a manageable size [29]. These approaches are considered during the construction of statechart reachability tree [20] and estelle reachability tree [27].

In summary, all the works above stress the relevance of providing coverage measures for concurrent and parallel programs. For instance, the relevance of this kind of information in the context of communication protocols has been discussed by Petrenko and Bochmann [23]. The testing coverage criteria family proposed in this article is based on the works discussed above, in the same vein of the criteria defined by Taylor et al. [30], Souza et al. [27, 29] and Chung et al. [3], but in the context of Coloured Petri Nets, addressing its specific features.

3. Coloured Petri Nets

Coloured Petri Nets are a formal technique for system description, with a strong mathematical basis, specially suitable for modeling systems with discrete events [15]. Among the main features of this technique are:

- the distributed description of conditions in contrast to other techniques (e.g., Finite State Machines);
- the explicit representation of causal dependence or independence of the system elements; and

- the appealing graphical representation of the system and its dynamic aspects.

The CPNs extend the ordinary Petri Nets [21, 22] by adding capabilities for defining data types and manipulating values of these types. For historical reasons, the types in a CPN are called "colours", to contrast with the ordinary Petri Nets in which all the values are indistinguishable.

A CPN is a bipartite graph whose nodes are divided up in two disjoint sets: the places P (that usually represent the passive elements of the system) and the transitions T (that represent the active elements). Each place is associated with a colour and can contain multi-set[1] values of this colour. An arc is an edge in the graph that links a place $p \in P$ to a transition $t \in T$ (called *an input arc*) or transition t to place p (called *an output arc*). The arc can be annotated with an expression that can be composed by variables, values and operations. The expression annotated in an arc must, when evaluated, generate a multi-set of values from the colour of the adjacent place, assigning values to all its variables, if any.

A marking is a specific association of each place with a multi-set of elements from its respective colour and denotes a particular configuration of the CPN. A binding is a pair of a transition t and an assignment s of values to all variables, if any, that appear in any arc adjacent to t. In a given marking m, a binding $b = (t, s)$ is *enabled* if the multi-set resulting from evaluating the expression annotated in every input arc from the place p to t with the assignment s is less than multi-set associated with p in m. A binding b that is enabled in m can be *fired*, yielding another marking m' (denoted by $m[b\rangle m'$). m' is related to m in such a way that, for every input arc from a place p to t, the multi-set resulting from evaluate the annotated expression with s is removed from p, and, conversely, for every output arc from t to a place p, the resulting multi-set is included into p.

One of the most appealing features of CPN is its graphical representation. Figure 1 presents an example of the graphical representation of a CPN. This CPN was extracted from [15] and models a (simplified) Distributed Data Base. It is composed by nine states (represented by ellipses) and four transitions (represented by rectangles). The transitions are: Update and Send Messages (SM); Receive a Message (RM); Send an Acknowledgment (SA); and Receive all Acknowledgments (RA). The colour and initial contents of each place is positioned near the corresponding ellipse, in italics and in upright font, respectively. The annotations of each arc are near the respective arc.

The database is composed by n sites (with $n \geq 3$). Each site has its own copy of the data. After changing its copy, a site sends a message to all the others and waits for an acknowledgment message. When it receives all the acknowledgment messages, the system returns to the original state. The colour $DBM = \{d_1, d_2, \ldots, d_n\}$ represents the set of sites. The colour $MES = \{(s, r) \in DBM \times DBM \mid s \neq r\}$ represents all the messages that can be sent among the sites. Finally, the colour $E = \{e\}$ has a single element that is employed to switch between active (i.e., any site can make a change) and inactive (i.e., a site made a change and is waiting the acknowledgment messages). The variables that occur in this example are `r` and `s`. The auxiliary function $Mes : DBM \rightarrow 2^{MES}$ maps each site to the set of the other sites, i.e., $\forall d \in DBM$, $Mes(d) = \{(d, d') \in MES \mid d' \neq d\}$. In the

[1]Informally, a multi-set can be thought of as a set which allows multiple occurrence of the same element. For example, $1`e + 2`b$ is a multi-set in which the element e occurs once and the element b occurs twice. The relational comparison of two multi-sets is made by a elementwise comparison the quantity of each element in both multi-sets.

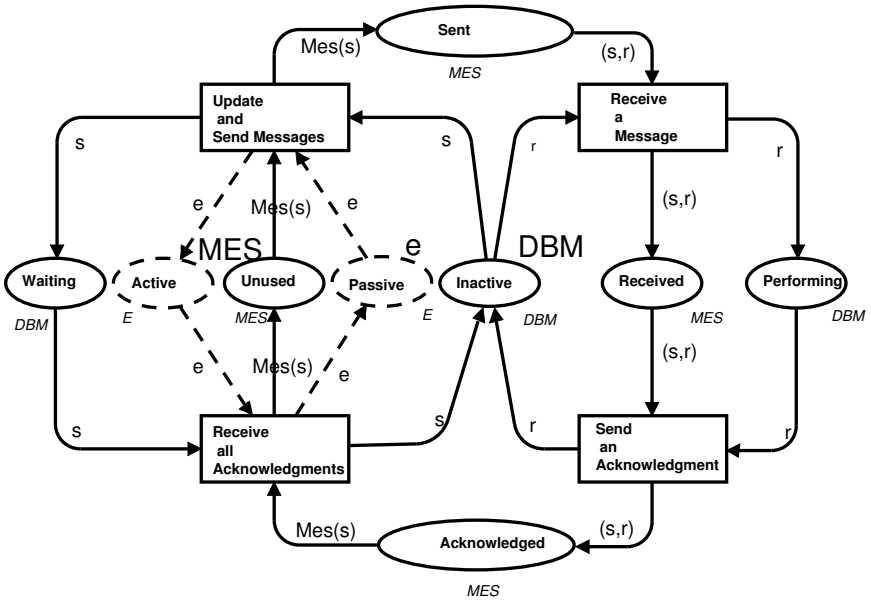

Figure 1: An Example of a Coloured Petri Net.

initial marking, the enabled bindings are $(SM, \langle \mathtt{s} = d_1 \rangle)$, $(SM, \langle \mathtt{s} = d_2 \rangle)$, $(SM, \langle \mathtt{s} = d_3 \rangle)$ and $(SM, \langle \mathtt{s} = d_4 \rangle)$.

In order to ease the definition and manipulation of large CPNs, a hierarchical tree of CPN pages can be used. A CPN page can be thought of as a simple CPN whose meaning, however, depends on other sub- and/or super-pages. The linking between a page and its sub-page is made through hierarchical transitions and port places. A hierarchical transition is associated with another page. Every place with arcs from or to a hierarchical transition is a port place. A hierarchical transition should associate a place in the subpage (called, socket place) to each of its port places. The pair of socket/port places should be compatible, that is, should have the same colour.

The semantics of a hierarchical CPN is the same of a non-hierarchical CPN, regarding that every pair of socket/port places behaves as if both are the same place. In other words, the contents of both places are always the same. This implies that, whenever an element is added to (respectively, removed from) either place, the same element is added to (respectively, removed from) the other place.

4. CPN Coverage Criteria Family (CPNCCF)

The efficacy of the testing and validation activities depends greatly on the quality of the test case suite employed. As pointed out before, from this viewpoint, there are two important questions that can be posed: "How to select test cases?" and "How to assure that a program or a specification has been tested enough?" The latter is usually addressed by taking in consideration coverage measures based on testing criteria.

A testing criterion allows the systematization of the testing activity, providing mechanisms either to select a test case suite or to measure the adequacy of a given one. Some researchers have explored the definition of testing criteria to validate specifications, mapping the concepts of criteria defined for the program level to the specification level [5, 6, 24, 27, 29, 31]. Fabbri et al. [5, 6] and Probert and Guo [24] explore the use of mutation testing, while Ural and Yang [31] explore the use of data-flow testing concepts.

Souza et al. [27, 29] propose a coverage Criteria for Statecharts and Estelle, respectively. In the same vein, this paper presents the CPN Coverage Criteria Family for validation of CPN based specification. These criteria emphasize intrinsic features of the CPN technique, as causal dependency and independency. CPNCCF is based on the control flow criteria [1, 25] and provides mechanisms to assess whether the specification satisfies the system's requirements. For instance, using these criteria the following questions can be addressed:

- Have all possible interleaving of markings been reached?
- Have all possible parallelism among markings been activated?
- Have all casual dependency (and independency) been exploited?

Next we define some concepts that will be used to define the CPNCCF criteria.

Path is a finite sequence of markings $P = \langle m_0, \ldots, m_k \rangle$, $k > 1$, such that the first marking is the initial marking m_0, and for each pair of marking m_i, m_{i+1}, with $0 \leq i < k$, there exists a binding b_i that is enabled in m_i and $m_i[b_i\rangle m_{i+1}$.

Simple path is a path $P = \langle m_0, \ldots, m_k \rangle$ such that all markings in the path, except possibly the first and the last, are distinct, i.e., $\forall j, i\ 0 < i \leq k \wedge 0 \leq j < k$ such that $i \neq j \rightarrow m_i \neq m_j$.

Loop-free path is a simple path P such that all the markings are distinct, i.e., $\forall j, i\ 0 \leq i, j \leq k$ such that $i \neq j \rightarrow m_i \neq m_j$.

The minimum coverage desirable for systems specified by CPN is to exercise all markings and all transitions. Thus, two coverage criteria are defined:

Definition 1 *The criterion* all-markings *requires that all markings of a CPN be reached at least once by a test sequence.*

Note that in the case of unbounded CPNs, this criterion is infeasible, since infinitely many markings can exist.

Definition 2 *The criterion* all-transitions *requires that all transitions of a CPN be fired at least once by a test sequence, regardless the binding.*

Additionally, since the same transition can be fired with distinct assignment (i.e., distinct bindings), the next coverage criterion is defined to consider this situation.

Definition 3 *The criterion* all-bindings *requires that all bindings be fired at least once by a test sequence.*

Note that in the case of CPNs with variables whose colours are infinite, this criterion is infeasible, since infinitely many bindings can exist.

Chow has shown that these criteria are not appropriate to reveal typical errors of finite state machine based on specification [2], and, made the necessary analogy, are not suitable for typical errors of CPNs as well. Therefore, we define other "stronger" criteria.

Definition 4 *The criterion* all-paths *requires that all paths be reached at least once by a test sequence.*

Observe that, in general, the *all-paths* criterion is not applicable because infinite paths may exist. Therefore, definitions 5, 6, and 7 introduce more rigorous criteria than the criteria *all-markings*, *all-transitions* and *all-bindings*, but less costly than the criterion *all-paths*. These criteria establish some constraints to the selection of paths.

Definition 5 *The criterion* all-simple-paths *requires that all simple paths are traversed at least once by a test sequence.*

Definition 6 *The criterion* all-loop-free-paths *requires that all loop-free paths be traversed at least once by a test sequence.*

Definition 7 *The criterion* all-paths-k-markings *requires that all paths containing at most k repetitions (with $k \geq 2$) of each marking be traversed at least once by a test sequence.*

These coverage criteria establish test requirements that need to be exercised by a test sequence to be considered adequate with respect to these criteria. A test sequence set T is adequate in relation to a given test criterion $\mathcal{C}$ (noted by $\mathcal{C}_{\text{adequate}}$) if T satisfies or executes every test requirements imposed by $\mathcal{C}$ [35].

4.1. CPNCCF Property Analysis

There are three meaningful bases against which test adequacy criteria can be compared: cost, effectiveness and strength. From the theoretical point of view, strength can be analyzed by the subsumption relation [25]. In this section, based on subsumption relation [25], we analyze the hierarchy among the CPNCCF criteria. According to Zhu [35], the subsumption relation is perhaps the property that we know best about adequacy criteria, although not all of them can be easily placed in the hierarchy, such as specification-based criteria. Zhu has also shown that under certain circumstance the subsumption relation can provide information to compare the effectiveness of the criteria. We consider the addressing of this aspect at the specification level to be a contribution aiming at the comparison of specification testing criteria. A criterion $\mathcal{C}_1$ subsumes a criterion $\mathcal{C}_2$ if for any set of paths P that satisfies $\mathcal{C}_1$ implies P would also satisfy $\mathcal{C}_2$, for any specification S. $\mathcal{C}_1$ strictly subsumes $\mathcal{C}_2$ if $\mathcal{C}_1$ subsumes $\mathcal{C}_2$ but $\mathcal{C}_2$ does not subsume $\mathcal{C}_1$. $\mathcal{C}_1$ and $\mathcal{C}_2$ are incomparable if $\mathcal{C}_1$ does not subsume $\mathcal{C}_2$ and $\mathcal{C}_2$ does not subsume $\mathcal{C}_1$ [25].

Theorem 1 *Considering the criteria proposed in this paper, the following relation hold:*

(1) all-paths *strictly subsumes* all-paths-k-markings.

(2) all-paths-k-markings *strictly subsumes* all-simple-paths.

(3) all-simple-paths *strictly subsumes* all-loop-free-paths.

(4) all-bindings *strictly subsumes* all-transitions.

Proof 1 All these relations are proved almost in the same way. For sake of space, we do not include all the proofs in this paper. Consider relation (3) to show the reasoning to prove these relations. Let P_1 be a set of path that satisfies the *all-simple-paths* criterion, e.g., P_1 is *all-simple-paths*$_{\text{adequate}}$. Thus, by definition, P_1 contains all possible paths

whose markings are all distinct, excepting, eventually, the first and last ones. Now, let P_2 be *all-loop-free-paths*$_{\text{adequate}}$. Thus, by definition, P_2 contains all possible paths whose markings are all distinct. Therefore, it can be verified that $P_2 \subseteq P_1$. It may be concluded that P_1 is also *all-loop-free-paths*$_{\text{adequate}}$, i.e., the *all-simple-paths* criterion subsumes the *all-loop-free-paths* criterion. On the other hand, P_2 does not satisfy the *all-simple-paths* criterion because. Consider a path p whose first and last markings are the same. Therefore, $p \notin P_2$ and the *all-loop-free-markings* criterion does not subsume the *all-simple-paths* criterion.

4.2. CPNCCF Criteria: Establishing Testing Requirements

The test requirements established by the CPNCCF criteria are obtained from the occurrence graph (OG) [16, 18]. The OG summarizes the possible sequences of firing from the initial marking m_0. An OG is a graph whose nodes are the reachable marking of the CPN and whose edges are the bindings that lead from one marking to another. Formally, the graph $G = (N, E)$ is the occurrence graph for a CPN C if and only if,

- $m_0 \in N$ (i.e., the initial marking of C is a node of the graph);
- for every $m \in N$ and every binding b enabled in m such that $m[b\rangle m'$, then $m' \in N$ and $(m, b, m') \in E$; and conversely,
- for every $(m, b, m') \in E$, $m[b\rangle m'$.

A problem that severely hinders the usage of occurrence graphs is the so called *state explosion problem*. The state explosion problem occurs when the number of nodes increases exponentially, as the lengths of the sequences increase. In some cases, the number of nodes can be infinite, especially with unbound CPN or with CPN that has colour with infinite number of elements. To overcome the state explosion problem, there are some approaches that reduce the occurrence graph, such as Equivalence Classes and Stubborn Sets. The Equivalence Classes employs the concept that some markings are, in some sense, similar to each other [18]. These markings can, indeed, be thought of as equivalent. Instead of constructing an occurrence graph whose nodes are every reachable marking, the nodes of the occurrence graph represent the classes of equivalent markings. In order to apply this approach, it is necessary to define an equivalence relation for the set of markings and an equivalence relation for the set of bindings. Such relations are dependent both on a particular CPN and on the properties which the user may want to consider. Note that the equivalence relation can be defined in such a way that an infinite number of markings (respectively, bindings) is represented by a single equivalence class. Therefore, the problem of unbound CPNs (i.e., CPNs whose set of reachable markings is infinite) can also be coped with proper equivalence relations.

The Stubborn Sets are sets of enabled bindings that are mutually independent, i.e., bindings that can be fired in any order, reaching the same marking. In this approach, instead of considering all the possible combinations of these bindings, only an element, called *stubborn set* is included in the graph.

Figure 2 presents the occurrence graph of CPN in Figure 1 with $n = 3$. In this example, neither Equivalence Classes nor Stubborn Sets were employed. For the sake of space, the markings are represented in a condensed way, indicating only the list of those sites that have a message addressed to them in the places *Sent*, *Received* and *Acknowledged*,

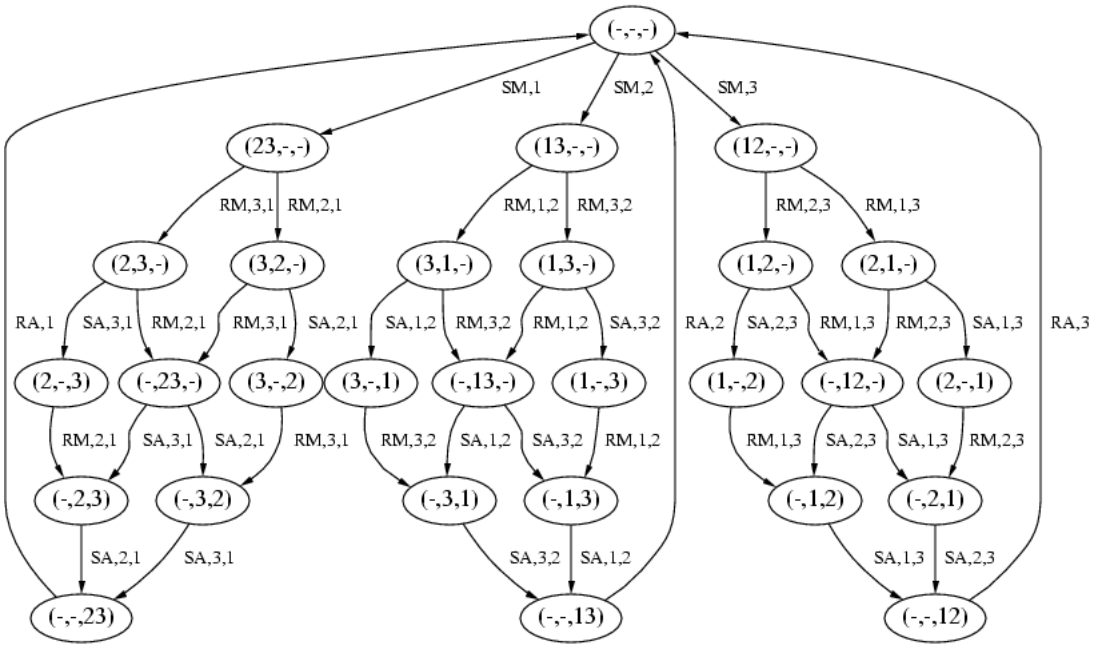

Figure 2: Occurrence Graph.

respectively [18]. For example, (-,-,-) represents that all sites are in *Inactive* and (2,3,-) represents that site d_1 is in *Waiting*, site d_2 is in *Inactive* and d_3 is in *Performing*. Analogously, (23,-,-) denotes a marking in which d_2 and d_3 are *Inactive* and d_1 is Waiting. It can be verified that the contents of all the other places can be derived from this condensed information. Analogously, the bindings are condensedly represented by the acronym of the transition and the sites assigned to the variables.

To illustrate the use of the OG in the implementation of the CPNCCF criteria, we will consider the *all-markings* and *all-loop-free-paths* criteria. The requirements established by the *all-markings* criterion correspond to the set of markings in the OG (i.e., the set of nodes). For the *all-loop-free-paths*, the testing requirements are obtained traversing the OG and collecting all paths that are loop free. The other criteria can be implemented in a similar way.

Table 1 presents the number of testing requirements established by each of the criterion of CPNCCF for the CPN in Figure 1, as well as some examples of them. These requirements can be checked against the OG in Figure 2.

In order to illustrate how the occurrence graph can be reduced and the impact of this reduction in the testing requirements, consider the application of Equivalence Classes to this example. The equivalence relation for markings is defined such that two markings are considered equivalent if, and only if, there exists a bijective function $\varphi : DBM \rightarrow DBM$ that can "convert" one marking in the other. For example, the marking (2,3,-) is equivalent to the marking (1,2,-), since the latter can be obtained from the former by means of the bijection function that maps d_1 into d_3, d_2 into d_1 and d_3 into d_2. The equivalence relation for bindings is defined analogously. Intuitively, these equivalence relations mean that two markings are equivalent if they differ only in the names of the DBMs. The reduced occurrence graph is presented in Figure 3. It can be noted that the graph is fairly smaller than the one presented in Figure 2.

Table 1: Subset of the Test Requirements (tr) for the CPN of Figure 1.

Criteria	Number of Test Requirements and Some Examples	
all-paths	∞	tr = {⟨(-,-,-), (12,-,-), (1,2,-), (-,12,-), (-,1,2), (-,-,12), (-,-,-), (13,-,-), (1,3,-)⟩, ⟨(-,-,-), (23,-,-), (3,2,-), (3,-,2)⟩, ...}
*all-paths-k--markings**	1102	tr = {⟨(-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2), (-,-,23), (-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2)⟩, ⟨(-,-,-), (23,-,-), (3,2,-), (-,23,-), (-,3,2), (-,-,23), (-,-,-), (13,-,-), (3,1,-), (-,13,-), (-,1,3), (-,-,13)⟩, ...}
all-simple-paths	76	tr = {⟨(-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2), (-,-,23), (-,-,-)⟩, ⟨(-,-,-), (23,-,-), (2,3,-), (-,23,-), (-,3,2), (-,-,23)⟩, ...}
all-loop-free-paths	58	tr = {⟨(-,-,-) (23,-,-) (2,3,-)⟩, ⟨(-,-,-) (13,-,-) (3,1,-) (3,-,1) (-,3,1) (-,-,13)⟩, ⟨(-,-,-) (23,-,-) (3,2,-) (3,-,2)⟩, ...}
all-transitions	4	$tr = \{SM, RM, SA, RA\}$
all-bindings	18	tr = { "`RA,1`", "`RA,2`", "`RA,3`", "`RM,1,2`", "`RM,1,3`", "`RM,2,1`", "`RM,2,3`", "`RM,3,1`", "`RM,3,2`", "`SA,1,2`", "`SA,1,3`", "`SA,2,1`", "`SA,2,3`", "`SA,3,1`", "`SA,3,2`", "`SM,1`", "`SM,2`", "`SM,3`" }
all-markings	28	tr = {(-,-,-), (23,-,-), (13,-,-), (12,-,-), (3,2,-), (2,3,-), (3,-,2), (-,23,-), (-,3,2), (-,-,23), (-,2,3), (2,-,3), (3,1,-), (1,3,-), (3,-,1), (-,13,-), (-,3,1), (-,-,13), (-,1,3), (1,-,3), (2,1,-), (1,2,-), (2,-,1), (-,12,-), (-,2,1), (-,-,12), (-,1,2), (1,-,2) }

* With k = 2.

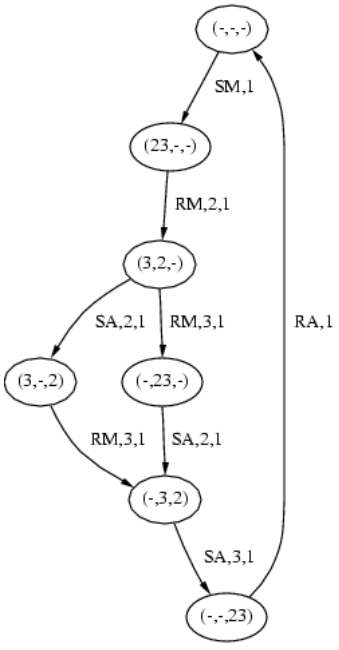

Figure 3: Occurrence Graph Reduced with Equivalence Classes.

Table 2: Subset of the Test Requirements (tr) for the CPN of Figure 1 with Equivalence Classes.

Criteria	Number of Test Requirements and Some Examples	
all-paths	∞	$tr = \{\langle$(-,-,-), (12,-,-), (1,2,-), (-,12,-), (-,1,2), (-,-,12), (-,-,-), (13,-,-), (1,3,-)$\rangle$, $\langle$(-,-,-), (23,-,-), (3,2,-), (3,-,2)$\rangle$, ... $\}$
*all-paths-k--markings**	27	$tr = \{\langle$(-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2), (-,-,23), (-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2)$\rangle$, $\langle$(-,-,-), (23,-,-), (3,2,-), (-,23,-), (-,3,2), (-,-,23), (-,-,-)$\rangle$, ... $\}$
all-simple-paths	11	$tr = \{\langle$(-,-,-), (23,-,-), (3,2,-), (3,-,2), (-,3,2), (-,-,23), (-,-,-)$\rangle$, $\langle$(-,-,-), (23,-,-), (2,3,-), (-,23,-), (-,3,2), (-,-,23)$\rangle$, ... $\}$
all-loop-free-paths	9	$tr = \{\langle$(-,-,-) (23,-,-) (2,3,-)$\rangle$, $\langle$(-,-,-) (23,-,-) (3,2,-) (3,-,2)$\rangle$, ... $\}$
all-transitions	4	$tr = \{SM, RM, SA, RA\}$
all-bindings	6	$tr = \{$"`RA,1`", "`RM,2,1`", "`RM,3,1`", "`SA,2,1`", "`SA,3,1`", "`SM,1`"$\}$
all-markings	7	$tr = \{$(-,-,-), (23,-,-), (3,2,-), (2,3,-), (3,-,2), (-,23,-), (-,3,2), (-,-,23), (-,2,3), (2,-,3) $\}$

* With k = 2.

Table 2 presents the number of testing requirements, taking into account the equivalence relations. The cardinality of the sets of test requirements are sensibly more manageable than in the case where no equivalence relation were considered. However, it will not be sensible to possible errors that are dependent on a particular DBM.

4.3. CPNCCF Criteria: Adequacy Analysis and Test Set Generation

The testing requirements can be used for two distinct but related purpose: i) to guide the generation of the test sequences or ii) to evaluate the adequacy of the test sequence set in relation to the corresponding coverage criterion. In this paper, we address the use of CPNCCF as a coverage criteria. We have developed a prototyping tool for supporting this task. This tool, named $\mathcal{T}es\mathcal{CC}_{CPN}$, is part of a large environment we are developing to supporting the testing and validation of CPNs with the coverage criteria of CPNCCF.

The overall schema of execution is presented in Figure 4. Before being able to execute $\mathcal{T}es\mathcal{CC}_{CPN}$, the CPN is input to an external module, named `cpn-ml2sml` (step a), which compiles it, generating SML code [12] for the simulation/execution of the net (step b). The CPN SML code is an intermediate representation of the CPN which embodies the firing semantics. The CPN SML code is, then, linked to $\mathcal{T}es\mathcal{CC}_{CPN}$ (step c).

One of the functionality of $\mathcal{T}es\mathcal{CC}_{CPN}$ is the generation of the occurrence graph. The occurrence graph is used internally to calculate the testing requirements. The tool can output the graph (step f), so that the user can inspect it. Currently, $\mathcal{T}es\mathcal{CC}_{CPN}$ outputs the graph in digraph format of GraphViz [10]. GraphViz is a powerful tool for drawing graphs and is able to generating several different kinds of image (e.g., JPEG, PNG, EPS)[2].

The $\mathcal{T}es\mathcal{CC}_{CPN}$ can be controlled (step d) with data about the testing criteria the user is interested in, as well as equivalence relations, if any. The equivalence relations must be input as SML functions that determine whether two markings (respectively, two bindings) are equivalent or not. There are two possible execution mode:

[2]The occurrence graphs in Figures 2 and 3 were generated in this way.

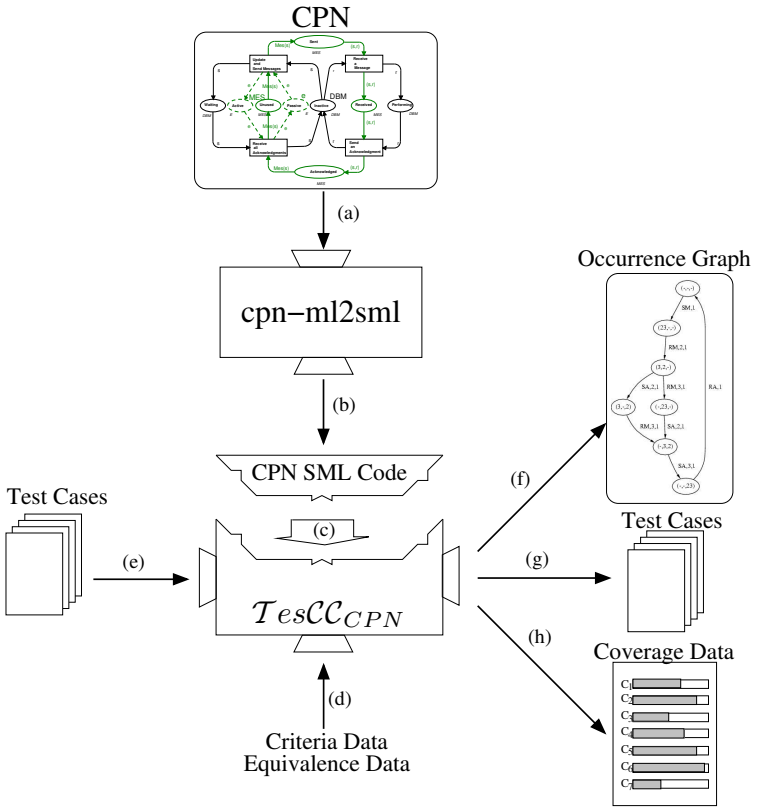

Figure 4: Overall of the Execution Schema of $\mathcal{T}esCC_{CPN}$.

(i) the set of test cases is input (step e) and the coverage data is output (step h). The coverage data include how many testing requirements are defined by a given criterion, how testing requirements were exercised by the test cases, and, if requested, which testing requirements were not exercised.

(ii) no input is furnished and a set of test cases is yielded (step g). In this case, the tool will produce a set of test cases that is, by construction, adequate to a given criterion.

5. Concluding Remarks

Our main contribution is the definition of a structural coverage criteria family for the Coloured Petri Nets, named CPN Coverage Criteria Family (CPNCCF). To our knowledge this is the first effort to provide coverage criteria to assess the quality of the testing and validation activities in the context of CPN specifications. The inclusion relation among these criteria has been addressed providing information for the establishment of an incremental testing strategy for CPN specifications.

The CPNCCF testing requirements are derived based on the occurrence graph. This information can be used to assess, considering a given criterion, how complete the set of test sequences is, by comparing the number of testing requirements that were derived with the number of testing that were exercised. Moreover, it can be used to generate test sequences.

It should be observed that the adequate test set obtained to test and validation the specification is in fact a mechanism to conduct the conformance testing for an imple-

mentation under test. In this scenario it would be worthwhile to further investigate the relationship between these abstraction levels: specification and implementation.

Two other aspects that need to be further explored are the cost and the effectiveness of the CPNCCF criteria family. The cost of the application evaluates the necessary effort to apply the criterion while the effectiveness evaluates the capability of the criterion in revealing errors. These aspects are currently under investigation.

The evolution of our work on this subject is directed to three lines of research:

- To conduct empirical studies to evaluate the cost and benefits of the proposed criteria. Although the preliminary results we presented in this paper indicates that the criteria can be used in practice, it is very important to collect empirical evidences of its effectiveness in revealing errors in real, industrial-scaled case studies. Currently, we are identifying such case studies.

- To conduct theoretical and empirical studies to compare the proposed criteria with other CPN validation techniques (e.g., Mutation Testing).

- To incorporate supporting to Stubborn Sets in the $\mathcal{T}es\mathcal{CC}_{CPN}$. Currently, it supports only Equivalence Classes as an approach to reduce the size of occurrence graph and to avoid the state explosion problem.

- To allow that the equivalence relations be input as functions defined in terms of elements from the CPN model. Currently, to input an equivalence relation in the $\mathcal{T}es\mathcal{CC}_{CPN}$, the user must have knowledge of the SML internal representation of the CPN.

- To investigate the impact of different equivalence relations in the cost and effectiveness of the criteria. The usage of a proper equivalence relation can drastically reduce the size of the occurrence graph and, thus, the number of testing requirements. However, it actually builds an abstraction of the system and may lose some information that may be useful to reveal errors.

References

[1] Beizer, B. (1990). *Software Testing Techniques*. Van Nostrand Eeinhold, New York, 2 edition.

[2] Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187.

[3] Chung, A., Sidhu, D., Wang, Y., Lin, W., and Kou, F. (1996). Task decomposition testing and metrics for concurrent programs. In *ISSRE'96 — International Symposium on Software Reliability Engineering*, pages 122–130.

[4] DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41.

[5] Fabbri, S. C. P. F., Maldonado, J. C., Masiero, P. C., and Delamaro, M. E. (1994). Mutation analysis testing for finite state machines. In *Fifth International Symposium on Software Reliability Engineering*, pages 220–229, Monterey, California, USA.

[6] Fabbri, S. C. P. F., Maldonado, J. C., Masiero, P. C., Delamaro, M. E., and Wong, E. (1995). Mutation testing applied to validate specifications based on Petri nets. In

FORTE'95 - 8th International IFIP Conference on Formal Description Techniques for Distributed Systems and Communications Protocol, Montreal, Canada.

[7] Fabbri, S. C. P. F., Maldonado, J. C., Sugeta, T., and Masiero, P. C. (1999). Mutation testing applied to validate specifications based on statecharts. In *ISSRE — International Symposium on Software Reliability Systems*, pages 210–219.

[8] Frankl, P. G. and Weyuker, E. J. (1993). A formal analysis of the fault-detecting ability of testing methods. *IEEE Transactions on Software Engineering*, 19(3):202–213.

[9] Fujiwara, S., Bochmann, G. V., Khendek, F., Amalou, M., and Ghedamsi, A. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603.

[10] Gansner, E. R. and North, S. C. (2001). An open graph visualization system and its applications to software engineering. *Software — Practice & Experience*, 30(11):1203–1233.

[11] Gill, A. (1962). *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York.

[12] Hansen, M. R. and Rischel, H. (1999). *Introduction to Programming using SML*. Addison-Wesley.

[13] Harel, D. (1987). Statecharts: On the formal semantics of statecharts. In *The 2nd IEEE Symposium on Logic in Computer Science*, pages 54–64, Ithaca, New York.

[14] Harel, D. (1992). Bitting the silver bullet — toward a brighter future for systems development. *IEEE Computer*, 25(1):8–20.

[15] Jensen, K. (1986). Coloured Petri nets. In *Application and Theory of Petri Nets*, volume 254 of *Lecture Notes on Computer Science*, pages 248–299, Berlin. Springer.

[16] Jensen, K. (1997a). A brief introduction to coloured petri nets. In Brinkma, E., editor, *Tools and Algorithms for Construction and Analysis of Systems*, volume 1217 of *Lecture Notes on Computer Science*, pages 203–208. Springer.

[17] Jensen, K. (1997b). *Coloured Petri Nets: Analysis Methods*. Springer, Berlin, 2 edition.

[18] Jensen, K. (1997c). *Coloured Petri Nets: Basic Concepts*. Springer, Berlin, 2 edition.

[19] Koppol, P. V. and Tai, K. C. (1996). An incremental approach to structural testing of concurrent software. In *International Symposium on Software Testing and Analysis*, pages 14–23.

[20] Masiero, P. C., Maldonado, J. C., and Boaventura, I. G. (1994). A reachability tree for statecharts and analysis of some properties. *Informations and Software Technology*, 36(10):615–624.

[21] Murata, T. (1984). Modeling and analysis of concurrent systems. In *Handbook of Software Engineering*. Van Nostrand Reinhold Electrical, New York.

[22] Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

[23] Petrenko, A., v. Bochmann, G., and Yao, M. (1996). On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, 29(1):81–106.

[24] Probert, R. L. and Guo, F. (1991). Mutation testing of protocols: Principles and preliminary experimental results. In *Proceedings of the IFIP TC6 Third International Workshop on Protocol Test Systems*, pages 57–76, North-Holland.

[25] Rapps, S. and Weyuker, E. J. (1985). Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375.

[26] Simão, A. S. and Maldonado, J. C. (2000). Mutation based test sequence generation for Petri nets. In *Proceedings of III Workshop of Formal Methods*, pages 68–79, João Pessoa, PB.

[27] Souza, S. R. S., Maldonado, J. C., and Fabbri, S. C. P. F. (2001). FCCE: Uma família de critérios de teste para validação de sistemas especificados em estelle. In *Anais do XV Simpósio Brasileiro de Engenharia de Software*, pages 256–271, Rio de Janeiro, Brasil.

[28] Souza, S. R. S., Maldonado, J. C., Fabbri, S. C. P. F., and Lopes de Souza, W. (2000a). Mutation testing applied to Estelle specifications. *Software Quality Journal*, 8(4):285–301.

[29] Souza, S. R. S., Maldonado, J. C., Fabbri, S. C. P. F., and Masiero, P. C. (2000b). Statecharts specifications: A family of coverage testing criteria. In *CLEI2000 - Conferncia Latino Americana de Informtica*, Cidade do Mexico, Mexico.

[30] Taylor, R. N., Levine, D. L., and Kelly, C. D. (1992). Structural testing of concurrent programs. *IEEE Transactions on Software Engineering*, 18(3):206–215.

[31] Ural, H. and Yang, B. (1991). A test sequence selection method for protocol testing. *IEEE Transactions on Communications*, 39(4):514–523.

[32] Yang, C., Souter, A. L., and Pollock, L. L. (1998). All-du-path coverage for parallel programs. In *International Symposium on Software Testing and Analysis*, pages 153–162.

[33] Yang, R. D. and Chung, C. (1992). Path analysis testing of concurrent programs. *Information and Software Technology*, 34(1):43–56.

[34] Zhu, H. (1996). A formal analysis of the subsume relation between software test adequacy criteria. *IEEE Transactions on Software Engineering*, 22(4):248–255.

[35] Zhu, H., Hall, P. A. V., and May, J. H. R. (1997). Software unit test coverage and adequacy. *Computer Survey*, 29(4):367–427.