Statecharts Specifications: A Family of Coverage Testing Criteria

Simone do R. S. de Souza[1], José Carlos Maldonado[2],
Sandra C. P. Ferraz Fabbri[3], and Paulo Cesar Masiero[2]

[1]Departamento de Informática, Universidade Estadual de Ponta Grossa -UEPG
Ponta Grossa, PR, Brazil
rocio@icmc.sc.usp.br
[2]Departamento de Ciências de Computação e Estatística, Universidade de São Paulo - ICMC-USP
C. Postal 668, 13560-970, São Carlos, SP, Brazil
{jcmaldon, masiero}@icmc.sc.usp.br
[3]Departamento de Computação, Universidade Federal de São Carlos -UFSCar
São Carlos, SP, Brazil
sfabbri@dc.ufscar.br

Abstract. This paper proposes a family of coverage testing criteria for specifications based on Statecharts. Statecharts are an extension of finite state machines with the capability of expressing parallelism and hierarchy. They have been used to specify the behavior of Reactive Systems. Recently, they have also been used in the context of object oriented software development. The two main approaches used for testing and validation of statecharts specifications are simulation and analysis of properties. However, these approaches do not essentially provide a mechanism for quantifying the specification testing activity and this compromises the testing quality assessment. The coverage criteria family proposed in this paper aims at complementing these approaches providing mechanisms either to evaluate the adequacy of test sequences generated by simulation, for example, 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 statecharts reachability tree. The concepts and criteria are illustrated using an environment for editing and simulating statecharts called StatSim.

Key Words: Software Engineering.

1. Introduction

Software Testing is one of the software quality assurance activities and a crucial issue is the establishment of a testing strategy to be applied during software development and maintenance. The main objective of the testing activity is to reveal errors. The earlier the errors are detected in the life cycle the less onerous is the process of removing them. Specification testing is in accordance with this fact and aim at validating system specifications against their requirements, as earlier as possible.

There are two main research issues in the testing area: how to select test cases and how to assure that a program P or a specification S has been tested enough. In general, there are three techniques to design and evaluate test cases: functional – test requirements are obtained from the software specification; structural – test requirements are obtained from a particular implementation; and error based – test requirements are obtained from typical and usual errors introduced during the development process. These techniques are complementary to each other as they exercise different characteristics of the software being tested.

For safety critical applications errors can produce disastrous consequences. Therefore, the use of formal techniques should be seriously considered, if not mandatory. Reactive Systems are a class of systems whose main feature is their interaction with the environment reacting to stimuli. Examples are metro control, patient hospital monitoring, and communication protocols. For these systems, software

quality is even more relevant as failures can provoke economic or human losses, making specification and testing activities much more critical. Finite State Machines (FSM) [12], Statecharts [13] and Petri Nets [17] 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 [16, 17], simulation [14] and, for FSMs, by several test sequences generation methods [6, 11, 18].

Statecharts have well defined syntax and semantics, allowing analysis of the dynamic aspects of the model. The system's components are modeled by a hierarchical set of extended finite state machines expressing parallelism and communication. According to Harel, the consistence and completeness verification of the model does not necessarily avoid the occurrence of logical errors [14]. Harel proposes some alternatives for validation of statecharts specifications: interactive, batch, programmed and exhaustive simulation [14]. Simulation allows observing how the model behaves as it reacts to the occurrence of pre-defined or randomly generated events. Analysis of properties is another alternative and allows evaluation of dynamic properties of the model, as deadlocks, configuration reachability, sequences of valid events, and firing of transitions. However, neither of them provides mechanisms for quantifying the testing and validation activities. The quality of the testing activity is by itself an issue in the software development process. One way to assess the quality of a test suite T is to use coverage measures based on a testing criterion.

Complementing current statecharts validation approaches, this paper proposes a *Statechart Coverage Criteria Family (SCCF)* for validation of statecharts based specifications. These criteria provide mechanisms to evaluate test sequences generated by simulation and mechanisms 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 statecharts called *reachability tree* [16]. The reachability tree shows the possible global states and the paths – sequences of configurations – that the system can take.

In Section 2 some related works are discussed. In Section 3 the basic concepts of statecharts that are the basis for understanding the criteria proposed herein are presented. In Section 4 the coverage criteria are defined. In Section 5 the hierarchy of the coverage criteria is analyzed. In Section 6 the reachability tree is used to establish the test requirements for the coverage criteria. In Section 7 we illustrate the application of the coverage criteria to assess the quality of a specification testing activity. Concluding remarks are presented in Section 8.

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 [7, 8, 9, 10, 15, 18, 19, 22, 23, 24, 25].

Yang and Chung introduced the path analysis testing of concurrent programs. Given a program, two models are proposed: 1) *task flowgraph* - corresponds to the syntactical view of the task execution behavior and models the task control flow; and 2) *rendezvous graph*: corresponds to the run-time view and models the possible rendezvous sequences among tasks [25]. 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* to support the debugging activity of concurrent programs is presented. 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 [22]. 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 (inclusion relation) 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. proposed four testing criteria for Ada programs: *all-entry-call, all-possible-entry-acceptance, all-entry-call-permutation* and *all-entry-call-dependency-permutation* [7]. 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)* [15] 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* [21] for parallel programs [24]. 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 with parallelism of threads constitute test requirements to be exercised. Threads are independent sequences of execution within a parallel program. The *della pasta* tool *(**Del**aware **Pa**rallel **S**oftware **T**esting **A**id*) automates their approach.

Probert and Guo [19] introduced the approach E-MPT (*Estelle-directed Mutation-based Protocol Testing)* that applies Mutation Testing 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 (variables, constants and mathematical operators) its syntactically correct alternatives based on the specification under testing.

Fabbri et al. defined Mutation Testing to validate specifications based on Petri Nets [9], Finite State Machines (FSM) [8] and Statecharts [10]. For each specification technique a mutation operator set has been defined inspired in the error classification suggested by Chow [6]. 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.

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 protocol has been discussed by Petrenko and Bochmann [18].

The SCCF testing coverage criteria family proposed in this article is based on the works discussed above, in the same line of Taylor et al and Chung et al's criteria [22, 7], but in the context of Statecharts, addressing its specific features. The authors are not aware of similar work for Statecharts, i.e., the proposition of coverage criteria, besides mutation testing, for statecharts-based specification testing and validation.

3. Statecharts

Statecharts are a visual formalism proposed by Harel for specification of complex reactive systems [13]. They provide an extension of Finite State Machines (FSM) to solve some aspects of FSM that make its application to complex systems difficult. Statecharts' outstanding features are hierarchy of states, ability to specify parallelism and communication mechanism via broadcasting, and a special notation set augmenting the representational power in relation to FSM. On the other hand, this increase of the representational power makes the validation activity more difficult [16].

The statecharts technique is summarized by Harel as follows: *statecharts = state diagrams + decomposition + orthogonality + broadcasting* [13]. This expression shows that statecharts are basically state diagrams and that its basic components correspond to Finite State Machines. States are of two types AND-states and OR-states. AND-states are called orthogonal components and have the distinctive feature that a system in an AND-state is also in all its orthogonal components. Thus, AND-states are composed by parallel substates that must stay simultaneously active. Figure 1 illustrates a statechart. The states *Tops_number, Receives_signal* and *Clicks_number* are components of the AND-state *Mouse_Hand*. In the other hand, OR-states specify that only one of its substates can be active at a certain time. In Figure 1, *Tops_number* is an OR-state that must have only one of its state *active.* A configuration of a statechart corresponds to the set of active states in a given time.

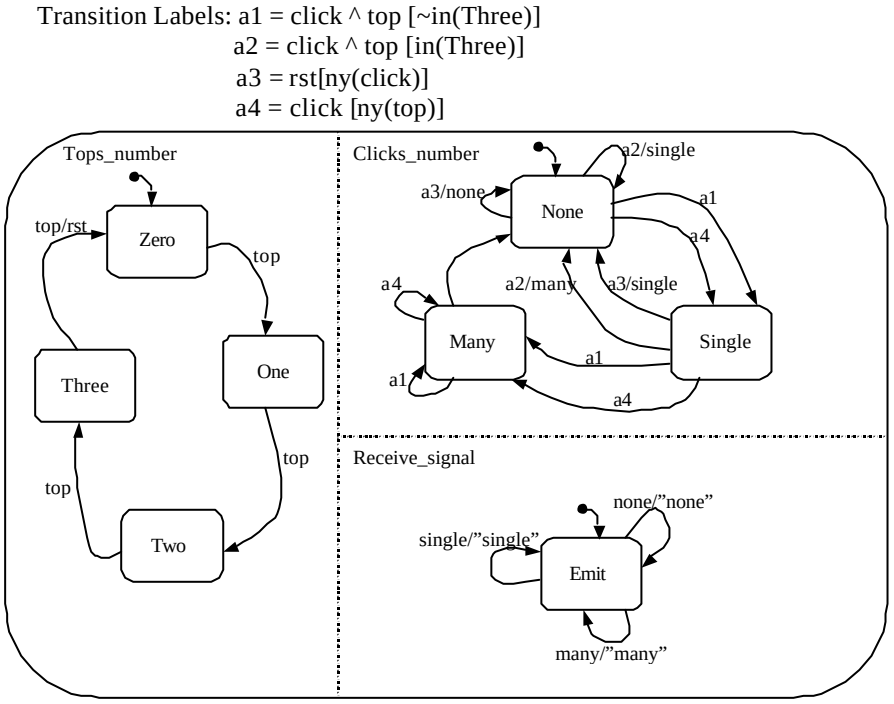

Figure 1. Statechart of a Mouse Handler [16].

States interact through transitions that consist of three basic parts: event expression, condition and action statements. These three components appear as a label attached to a transition according to the syntax: event-expression [condition]/{action}. The event expression may be null and the condition and action statements are also optional. An action associated to a fired transition may generate other events, which are broadcast to the parallel components. For example, in Figure 1 the event *rst* is broadcast when the transition *top/rst* of the component *Tops_number* fires.

A transition is relevant in a time step if its source states are active. It fires when both the event expression and the condition evaluate to true in this same time step. In this case, the actions, if any, are executed. Execution of an action may generate other events, which are broadcast to the orthogonal components, possibly firing new transitions. Assignments of arithmetic and logical expression to variables are also allowed as legal action statements. Conditions are formed by combination of logical and relational operators involving variables and special conditions, e.g., being in a state. Event expressions are formed from a basic set of primitive events, which can be combined using logical conjunctions and disjunctions. Some special events are allowed such as entering or leaving a state.

Transitions can also be annotated with a history symbol. This concept means that on reaching a composite state *A*, its substate to be activated will not necessarily be its default substate but the one visited the last time the state *A* was active. The default substate is always activated the first time A is active. The history may either be specified for just one level down or it may be recursively defined until the bottom level. History and recursive history are represented by the symbols H and H*, respectively. In our example, history symbols do not occur.

4. Statecharts Coverage Criteria Family (SCCF)

The success of the testing and validation activities is related to the quality of the test suite. As pointed out before, from this point of view, there are two important questions: *"How to select test cases?"* and *"How to assure that a program or a specification has been tested enough?"* The latter is usually addressed taking in consideration coverage measures based on testing criteria.

A testing criterion allows the systematization of the testing activity, providing mechanisms either to select test cases or to measure the adequacy of a given test suite. Some researchers have explored the definition of testing criteria to validate specifications, mapping the concepts of criteria defined to the program level to the specification level [8, 9, 10, 19, 23]. Fabbri et al [8,9,10] and Probert and Guo [19] explore the use of mutation testing while Ural and Yang [23] explore the use of data-flow based testing concepts. In the same vein, this paper presents the Statechart Coverage Criteria Family (SCCF) for validation of Statecharts-based specification. These criteria emphasize intrinsic features of the Statecharts technique, as parallelism, broadcasting and history. SCCF is based on the control flow based criteria [2] 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 states been reached?*
- *Have all possible parallelism among states been activated?*
- *Have all possible synchronization and communication among states been executed?*
- *Have all reachable states starting from a transition with history been traversed?*

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

- *Configuration:* a configuration C_i is a set of states that are active in one step of the computation. C_0 is the initial configuration. At each step we suppose that each event expression associated to the current configuration can evaluate to true and fire a new transition. In this way the configuration space can be constructed.
- *Path*: is a finite sequence of configurations $(C_0, C_i, C_j \ldots C_m, C_k)$, $k \geq 1$, such that the first configuration is the initial configuration (C_0), and exist a transition from C_0 to C_i, from C_i to C_j… and from C_m to C_k.
- *Simple path:* is a path *P* such that all configurations in the path, except possibly the first and the last, are distinct.
- *Loop-free path:* is a simple path *P* such that all the configurations are distinct.

- *Complete path:* is a path *P* whose first configuration is the initial configuration and whose last configuration is the terminal configuration. A configuration is terminal if it does not have an enabled transition.

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

Definition 1: The criterion *all-configurations* requests that all configurations of a statecharts be reached at least once by a test sequence.

Definition 2: The criterion *all-transitions* requests that all transitions of a statecharts be reached at least once by a test sequence.

Chow has shown that these criteria are not appropriate to reveal typical errors of finite state machine based specification [6]. This result can be considered in the context of statecharts since its basic components are extended finite state machines.

Definition 3: The criterion *all-paths* requests that all paths be reached at least once by a test sequence.

Observe that the *all-paths* criterion is not in general applicable because infinite paths may exist. Therefore, definitions 4, 5, and 6 introduce more rigorous criteria than the criteria *all-configurations* and *all-transitions* but less costly than the criterion *all-paths*. These criteria establish some constraints to the selection of paths and also establish a hierarchy among the criteria *all-paths*, *all-configurations*, and *all-transitions*:

Definition 4: The criterion *all-simple-paths* requests that all simple paths are traversed at least once by a test sequence.

Definition 5: The criterion *all-loop-free-paths* requests that all loop-free paths be traversed at least once by a test sequence.

Definition 6: The criterion *all-paths-k-configurations* requests that all paths containing at most *k* repetitions of each configuration be traversed at least once by a test sequence set.

The Statecharts technique has three features that need to be considered: parallelism, broadcasting and history. Parallelism is expressed by configurations that may have more than one active state. Considering this, the requirements that satisfy the criterion *all-configurations* also includes all parallelism among the states of the model. Broadcasting and history are treated by definitions 7 and 8.

Definition 7: The criterion *all-broadcasting* requests that all transitions with the action statement containing event generation be exercised at least once. Starting from a configuration, a broadcasting happens when a transition of the type *e[c]/a*, with *a* being an event, is fired. Therefore, all transitions of this type must be traversed at least once by a test sequence to satisfy this testing criterion.

Definition 8: The criterion *all-history* requests that for each transition annotated with a history symbol all possible configurations reachable from that transition be traversed at least once by a test sequence set.

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 *CR* (noted by CR-adequate) if *T* satisfies or executes every test requirements imposed by *CR* [21].

5. SCCF 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 inclusion relation [21]. In this section, based on subsume relation [21], we analyze the hierarchy among the SCCF criteria. According to Zhu [26], the subsume 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 subsume 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 CR_1 subsumes a criterion CR_2 if for any set of paths P that satisfies CR_1 implies P would also satisfy CR_2, for any specification S. CR_1 *strictly* subsumes CR_2 if CR_1 subsumes CR_2 but CR_2 does not subsume CR_1. CR_1 and CR_2 are *incomparable* if CR_1 does not subsume CR_2 and CR_2 does not subsume CR_1 [21].

Theorem 1: The Statecharts Coverage Criteria Family (SCCF) obeys the hierarchy of Figure 2.

Proof: Given that the inclusion relation is transitive, it is enough to demonstrate the following relations:

i. all-paths strictly subsumes all-paths-k-configurations.
ii. all-paths-k-configurations strictly subsumes all-simple-paths.
iii. all-paths-k-configurations strictly subsumes all-transitions.
iv. all-simple-paths strictly subsumes all-loop-free-paths.
v. all-simple-paths and all-transitions are incomparable.
vi. all-loop-free-paths strictly subsumes all-configurations.
vii. all-transitions strictly subsumes all-configurations.
viii. all-transitions strictly subsumes all-history-configurations.
ix. all-transitions strictly subsumes all-broadcasting.
x. all-configurations, all-history-configurations and all-broadcasting are incomparable.

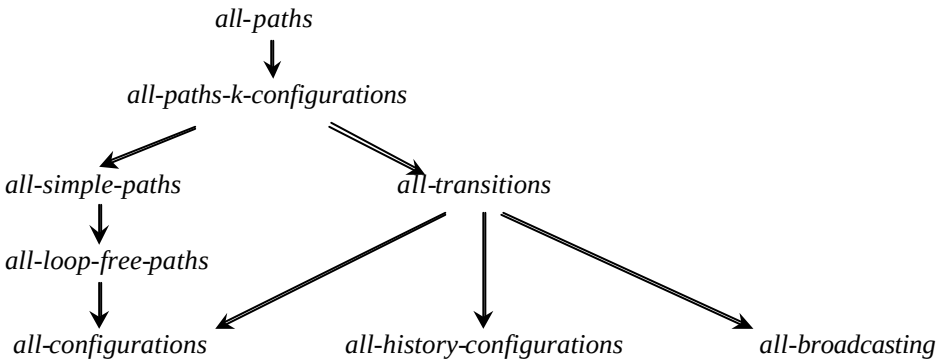

Figure 2. Hierarchy of SCCF Criteria.

All these relations are proved almost in the same way. Let us consider relation (i) to show the reasoning to prove these relations. Let us suppose that P_1 satisfies the all-paths criterion, e.g., P_1 is all-paths-adequate. Thus P_1 contains all possible paths characterized by the reachability tree. Observe that some of these paths may have infinite length. Now, let P_2 be all-paths-k-configurations-adequate. Thus, in each path $p \in P_2$ a given configuration C_i appears at most k times. For every path p $\in P_2$ there is at

least one path $m \in P_1$ that includes p; thus p is included in P_1. It may be concluded that P_1 is also all-paths-k-configuration-adequate, i.e., the all-paths criterion subsumes the all-paths-k-configurations criterion. On the other hand, P_2 does not satisfy the all-paths criterion because it would be enough to take a path with k_1 repetitions of a configuration C_i, $k_1 > k$. This path would not be necessarily included in P_2 . Therefore, the all-paths-k-configurations criterion does not subsume the all-paths criterion.

6. SCCF Criteria: Testing Requirements

The test requirements established by the SCCF criteria are easily obtained from the statecharts reachability tree [16]. The reachability tree definition is similar to the formal definition of a *transition system*. We can say that the reachability tree is a 4-uple $RT = <SC, T, ®, C_0>$. SC is the set of configurations, T is the set of transitions, the transition function ® is a partial function from SC x T to SC, and $C_0 \in$ SC is the initial configuration [20].

Figure 3 presents the reachability tree for the statechart of Figure 1. The tree has 12 possible configurations. The active states in the configurations are marked with the number 1 and the inactive states with the number 0. Consider the sequence of events *c, c, t, t.* When these events are fired the following configurations are reached C_0, C_2, C_6, $-C_7$, $-C_8$. When node $-C_7$ is reached the symbol - means that it is not a terminal node and that it is linked to the first occurrence of the same configuration in the tree; in this case, $-C_7$ is linked to the configuration C_7, which is the right-most son of C_2.

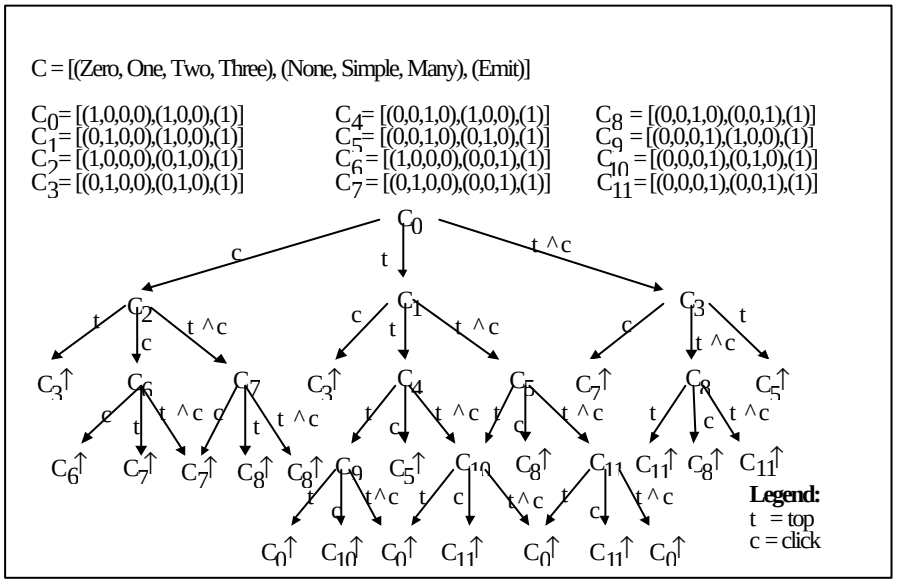

Figure 3. Reachability Tree for the Example of Figure 1.

Statecharts semantics allows many external events to happen at a time instant [13]. In the reachability tree it is considered each single transition rather than a combination of the different events which could happen at the time step, enabling transitions in orthogonal components [16]. As an example, we consider the reachability tree of Figure 3. Starting from C_0, the events sequence *t = top, click* can occur reaching the configuration C_3, which is the same result we would obtain with sequence *u = click, top*. It is described in the tree by the arc labeled $t \wedge c$.

To illustrate the use of the reachability tree to support the implementation of the SCCF criteria, the criteria *all-configurations* and *all-paths-k-configurations* are considered. The requirements established by the *all-configurations* criterion correspond to configuration set *SC* of *RT*. For the *all-paths-k-*

configurations, considering $k = 2$, the test requirements are obtained traversing in depth the reachability tree and selecting all paths where each configuration of SC appears at most twice in each path.

The procedure to obtain the test requirements is similar for all other SCCF coverage criteria. Table 1 presents a subset of the test requirements for the statechart of Figure 1 and the corresponding test sequences that would exercise those requirements.

Table 1. Subset of the Test Requirements (tr) and Test Sequences (ts) for the specification of Figure 1.

Criteria	Test Requirements and Test Sequences
*all-paths**	tr = {(C_0, C_2, C_3, C_5, C_{11}, C_0, C_2, C_6, C_6, C_7, C_8), (C_0, C_2, C_7, C_8, C_{11}, C_0, C_1, C_3, C_5, C_{11}), (C_0, C_1, C_4, C_9, C_0, C_3, C_5, C_{11}, C_{11}, C_0, C_1, C_4, C_9, C_0)…}
	ts = {(c, t, {t,c}, {t,c}, t, c, c, c, t, {t,c}), (c, {t,c}, t, t, {t,c}, t, c, t, {t,c}), (t, t, t, t, {t,c}, t, {t,c}, c, {t,c}, t, t, t, t) …}
all-paths-k-configuration	tr = {(C_0, C_2, C_3, C_5, C_{11}, C_0, C_1, C_5, C_{11}), (C_0, C_2, C_6, C_6, C_7, C_8, C_{11}, C_0), (C_0, C_1, C_5, C_1, C_4, C_5), (C_0, C_3, C_7, C_8, C_{11}, C_{11}, C_0, C_3, C_8)...}
	ts = {(c, t, t, {t,c}, t, t, {t,c}, {t,c}), (c, c, c, t, t, t, t), (t, {t,c}, t, t, c), ({t,c}, c, t, t, c, {t,c}, {t,c}, {t,c})...}
all-simple-paths	tr = {(C_0, C_2, C_3, C_8, C_{11}, C_0), (C_0, C_2, C_6, C_7, C_8, C_{11}, C_0), (C_0, C_1, C_3, C_5, C_{10}, C_0), (C_0, C_1, C_4, C_{10}, C_{11}, C_0), (C_0, C_3, C_7, C_8, C_{11}, C_0), (C_0, C_3, C_5, C_{10}, C_{11}, C_0) ...}
	ts = {c, t, {t,c}, t, {t,c}), (c, c, t, t, t, t), (t, c, t, t, t), (t, t, {t,c}, c, {t,c}), ({t,c}, c, t, {t,c}, t), ({t,c}, t, t, c, {t,c})...}
all- loop-free-paths	tr = {(C_0, C_2, C_3, C_8), (C_0, C_2, C_6, C_7, C_8, C_{11}), (C_0, C_1, C_3, C_5, C_{10}), (C_0, C_1, C_4, C_5, C_{11}), (C_0, C_1, C_5, C_{10}, C_{11}), (C_0, C_3, C_8, C_{11}), (C_0, C_3, C_7, C_8, C_{11})...}
	ts = {(c, t, {t,c}), (c, c, t, t, {t,c}), (t, c, t, t), (t, t, c, {t,c}), (t, {t,c}, t, c), ({t,c}, {t,c}, t), ({t,c}, c, t, t)...}
all-transitions	tr = {(C_0,C_2), (C_2,C_3), (C_2, C_6), (C_2, C_7), (C_3, C_7), (C_7, C_7), (C_7, C_8), (C_8, C_{11}), (C_{11}, C_0), (C_0,C_1), (C_1,C_3), (C_3,C_5), (C_5,C_{10}), (C_{10}, C_0), (C_1, C_5), (C_5, C_8), ...}
	ts = {(c), (c, t), (c, c), (c, {t,c}), ({t,c}, c), (c, {t,c}, c), (c, {t,c}, t), ({t,c}, {t,c}, t), ({t,c}, {t,c}, {t,c}, {t,c}) (t), (t, c), ({t,c}, t), ({t,c}, t, t), (t, t, {t,c}, t), (t, {t,c}), ...}
all-configurations	tr = {C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}}
	ts = {(t), (c), (c, t), (t, t), (t, {t,c}), (c,c), (c, {t,c}), (c, {t,c}, t), (t, t, t), (t, t, {t,c}), ({t,c}, {t,c}, t)}
all-broadcasting	tr = {(C_9,C_0), (C_{10},C_0), (C_{11},C_0), (C_9,C_0), (C_9,C_0), (C_{10},C_0), (C_{10},C_0), (C_{11},C_0)}
	ts = {(t, t, t, t), (t, t, t, {t,c}), (t, t, {t,c}, {t,c}), (t, t, {t,c}, t), (t, {t,c}, {t,c}, {t,c})}

*An infinite number of paths may be generated.

7. SCCF Criteria: Adequacy Analysis and Test Set Generation

The test requirements can guide the generation of the test sequences or be used to evaluate the adequacy of the test sequence set in relation to the corresponding coverage criterion. In this paper we address the use of SCCF as a coverage criteria. To illustrate this aspect we use the *StatSim (Statecharts Simulator)* environment. *StatSim* supports edition, simulation and analysis of statecharts. There are four approaches to simulate statecharts: *interactive, batch, programmed and exhaustive* [5]. These approaches are based on generating sequences of events according to several strategies and analyzing the statecharts behavior when these events are enabled. The exhaustive simulation is done based on the reachability tree and allows the verification of some statecharts dynamic properties, such as valid sequence of events, reachability, and deadlocks [5,16].

Let us consider Programmed Simulation to generate the event sequence set. In the programmed simulation a model is simulated controlled by a program that specifies the event occurrence probabilities and states what is to be accomplished at each step. This program is described using the

Execution Control Language (ECL). ECL enables to specify the number of simulation steps, the initial configuration, the type of simulation, probabilities of event occurrence, etc. As a result of the simulation, two reports are generated [5]: a statistical analysis of the simulation and a log file of all configurations reached during the simulation. The provided information can be used for coverage analysis for the criteria *all-configurations* and *all-transitions.*

Two simulations were carried out, generating two event sequences (S_a and S_b). In the first one, the probabilities of occurrence provided were 90% for event *top* and 30% for event *click*. In the second simulation, the probability was 80% for both events *top* and *click*. The results of the simulation were analyzed, computing the coverage of these event sequences in relation to the SCCF criteria.

Table 2 shows the number of the test requirements for some of the SCCF criteria. It should be noticed that the criterion *all-paths* determines infinite paths for this case. Table 3 summarizes the adequacy analysis results for the test sequences S_a, S_b and $S_a \cup S_b$.

Once we have the adequacy analysis we can improve the current test sequence sets used. For instance, sequence $S_a \cup S_b$ exercises 88.89% of the test requirements established by the criterion *all-transitions*, i.e., obtains coverage of 88.89%. In this situation we could randomly generate new test sequences and repeat the adequacy analysis until the desired coverage is reached or we could generate specific test sequences to exercise the not-yet-covered test requirements.

In the other way around, if an adequate test sequence set TS_i is obtained by construction it could be used to feed the statechart simulator, and be considered as minimum requirement for simulation.

It should be noticed the importance of automating the procedures to generate a set of test sequences adequate by construction to the criteria and to assess the adequacy of a given test sequence. Even for the weakest criterion and for simple cases as our example it would be error prone and time consuming to manually identify the test requirements.

Table 2. SCCF Criteria: Test Requirements.

Coverage Criteria	Test Requirements
All-simple-paths	170
All-loop-free-path	68
All-transitions	36
All-configurations	12
All-broadcasting	8

Table 3. SCCF Criteria: Coverage Analysis.

Coverage Criteria	Percentage of Coverage		
	S_a	S_b	S_a È S_b
All-simple-path	19.41	42.35	56.47
All-loop-free-path	11.76	48.53	57.35
All-transitions	55.50	58.33	88.89
All-configurations	75.00	75.00	100.00
All-broadcasting	100.00	25.00	100.00

8. Concluding Remarks

Our main contribution is the definition of a structural coverage criteria family for the statecharts language, named *Statechart Coverage Criteria Family (SCCF).* 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 statecharts specifications. These criteria take into account specific features of the statecharts technique such as parallelism, broadcasting and history and are purely based on control flow information. The inclusion relation among these criteria has been addressed providing information for the establishment of an incremental testing strategy for statecharts specifications. The concepts and the criteria are illustrated using a mouse handler specification [3, 16].

The SCCF testing requirements are discussed based on the reachability tree [16]. We illustrated the use of SCCF criteria family as coverage criteria taking test sequence sets generated by the simulation facilities available in the *StatSim* environment [5,16]. This information is used to improve the test sequence set if some of the test requirements are not exercised.

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 implementation 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 SCCF 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.

A well-known problem is the state explosion that occurs when modeling the number of possible global states by the reachability tree. Barnard describes four approaches to overcome this limitation during the construction of such trees, reducing its size to a manageable size [1]. Two of these approaches, identifying duplicate nodes and stubborn sets, were taken by Masiero and colleagues to implement the proposed statecharts reachablity tree [16]. These aspects ease the definition and implementation of the proposed SCCF criteria.

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

- To explore data flow based criteria in this context. Yang et al. have explored the all-du-paths criteria for parallel programs [24]. This would require aggregating data flow information to the reachability tree;
- To integrate into the StatSim facilities to use SCCF for testing and validating statecharts specifications. These facilities will be integrated with the simulation approaches already available in the environment;
- To conduct empirical studies to evaluate the cost and benefits of the proposed criteria. Compare the SCCF criteria against Mutation Testing, that we have also been exploring in the context of statecharts [10]; and
- To evaluate the use of the SCCF criteria and Mutation Testing in the context of other specification techniques such as and Estelle [4].

Acknowledgements

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

References

1. Barnard, J., COMX: a design methodology using communicating X-machines, *Information and Software Technology*, vol.40, pp.271-280, 1998.
2. Beizer, B., *Software Testing Techniques*, 2nd Edition, Van Nostrand Reinhold, New York, 1990.
3. Boussinot, F.; Simone, R.D., The Esterel Language, *in proc. IEEE*, vol.79(9), pp.1293-1303, 1991.
4. Budkowski, S.; Dembinski, P., An Introduction to Estelle: a specification language for distributed systems, *Computer Network and ISDN Systems*, vol.14(1), pp.3-23, 1987.
5. Cangussu, J.W.L.; Penteado, R.A.D.; Masiero, P.C.; Maldonado, J.C., Validation of Statecharts Based on Programmed Execution, *Journal of Computing and Information*, vol. 1(2), Special Issue of the 7th International Conference on Computer and Information – ICCI'95, Ontario, Canada, July, 1995.
6. Chow, T.S., Testing Software Design Modeled by Finite-State Machines, *IEEE TSE*, vol. 4(3), pp. 178-187, 1978.

7. Chung, C-M; Shih, T.K.; Wang, Y-H; Lin, W-C; Kou, Y-F., Task Decomposition Testing and Metrics for Concurrent Programs, *ISSRE'96 – International Symposium on Software Reliability Engineering*, pp. 122-130, 1996.
8. Fabbri, S.C.P.F.; Maldonado, J.C.; Delamaro, M.E.; Masiero, P.C., Mutation Analysis Testing for Finite State Machine, *ISSRE'94 – International Symposium on Software Reliability Engineering*, California, USA, pp.220-229, November, 1994.
9. Fabbri, S.C.P.F.; Maldonado, J.C.; Masiero, P.C.; Delamaro, M.E.; Wong, E., Mutation Testing Applied to Validate Specifications Based on Petri Nets, *FORTE'95 - International IFIP Conference on Formal Description Techniques for Distributed Systems and Communications Protocol*, Montreal, Canada, October, 1995.
10. Fabbri, S.C.P.F.; Maldonado, J.C.; Sugeta, T.; Masiero, P.C., Mutation Testing Applied to Validate Specifications Based on Statecharts, *ISSRE'99 – International Symposium on Software Reliability Engineering*, pp 210-219, Florida, USA, November 1999.
11. Fujiwara, S.; Bochmann, G.V.; Khendek,F.; Amalou, M.; Ghedamsi, A., Test Selection Based on Finite State Models, *IEEE TSE*, vol. 17(6), June, 1991.
12. Gill, A., *Introduction to the Theory of Finite-State Machines*, New York, McGraw-Hill, 1962.
13. Harel, D.; Pinnel, A.; Schmidt, J.P.; Sherman, R. On the Formal Semantics of Statecharts, *in proc. 2nd IEEE Symposium on Logic in Computer Science*, New York, USA, 1987.
14. Harel, D. Biting the Silver Bullet - Toward a Brighter Future for Systems Development, *IEEE Computer*, pp. 8-20, January, 1992.
15. Koppol, P.V.; Tai, K-C. An Incremental Approach to Structural Testing of Concurrent Software, *ISSTA'96 - International Symposium on Software Testing and Analysis, ACM-Software Engineering Notes*, pp. 14-23, 1996.
16. Masiero, P.C.; Maldonado, J.C.; Boaventura, I.G. A Reachability Tree for Statecharts and Analysis of some Properties, *Information and Software Technology*, vol.36(10), pp.615-624, 1994.
17. Peterson, J.L. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, 1981.
18. Petrenko, A.; Bochmann, G.v. On Fault Coverage of Tests for Finite State Specifications, available at <www.iro.umontreal.ca/labs/teleinfo/PubListIndex.html>, 1996.
19. Probert, R.L.; Guo, F. Mutation Testing of Protocols: principles and preliminary experimental results, *III IFIP TC6, Third International Workshop*, pp. 57-76, 1991.
20. Quemener, Y-M; Jéron, T. Finitely Representing Infinite Reachability Graphs of CFSMS with Graph Grammars, *publication n.994, Institut de Recherche en Informatique et Systèmes Aléatoires*, France, March, 1996.
21. Rapps, S.; Weyuker, E.J. Selecting Software Test Data Using Data Flow Information, *IEEE TSE*, vol.11(4), April, 1985.
22. Taylor, R.N.; Levine, D.L.; Kelly, C.D. Structural Testing of Concurrent Programs, *IEEE TSE*, vol.18(3), March, 1992.
23. Ural, H.; Yang, B. A Test Sequence Selection Method for Protocol Testing, *IEEE Trans. on Communications*, vol.39(4), April, 1991.
24. Yang, C-S; Souter, A.L.; Pollock, L.L. All-du-path Coverage for Parallel Programs, *ISSTA'98 - International Symposium on Software Testing and Analysis, ACM-Soft.Eng. Notes*, pp. 153-162, 1998.
25. Yang, R-D., Chung, C-G. Path Analysis testing of concurrent programs, *Information and Software Technology*, vol. 34(1), January, 1992.
26. Zhu, H., A Formal Analysis of the Subsume Relation between Software Test Adequacy Criteria, *IEEE TSE*, vol. 22(04), April, 1996.