Towards Test Purpose Generation from CTL Properties for Reactive Systems

Daniel Aguiar da Silva\(^1,2\) and Patrícia D. L. Machado\(^3\)

Grupo de Métodos Formais  
Universidade Federal de Campina Grande  
Campina Grande, Brazil

Abstract

This paper presents an approach for the generation of test purposes in the form of labelled transition systems from specifications of properties in CTL. The approach is aimed at adapting the model checking process, by extending search algorithms to perform further analysis so that examples and counter-examples can be extracted. An algorithm for the generation of test purposes through analysis over the examples and counter-examples is presented, along with a case study to show the correspondence between the CTL properties and the generated test purposes.

Keywords: Test Purpose, Testing, Formal Testing, Formal Verification, Model Checking.

1 Introduction

Specifying systems that are reactive, distributed and concurrent can be complex and error-prone. Many formalisms (e.g. \([16,13,14]\)) have been defined to support this task, producing more precise and correct models. Thus, the verification of properties against models may be done through formal methods, becoming automated and more rigorous.

The success of applying formal verification techniques (e.g. model checking \([4]\)) to software development is increasing with the evolution of algorithms and tools. Properties are verified in an efficient and automated way, even against the complex and huge models of such systems, becoming essential to the correctness assurance of the models. Despite the important contribution of formal verification techniques to produce more reliable software systems, it does not assure conformance between

\(^1\) The author is supported by CAPES  
\(^2\) Email: daguiar@dsc.ufcg.edu.br  
\(^3\) Email: patricia@dsc.ufcg.edu.br
implementation and models. Thus, a validation technique, like conformance testing, is necessary to complement the software verification and validation process.

Testing is a popular validation technique recognized as a complement to verification techniques [25] (e.g. model checking). Conformance testing is a black box functional testing technique [15] that consists of checking the conformance between the implementation under test (IUT) and the specification. The IUT is a black box, so, its behaviour may only be visible through interactions with the tester. Such interactions are performed through the system’s boundaries with the environment, called points of control and observation (PCO’s). An approach to test case generation is based on the explicit specification of properties to be tested. Such properties are called test purposes, and they focus on specific parts of the specification [15]. In model-based testing the specification is given as a model [7], so, we use the terms as synonyms here.

Applying conformance testing to the testing of reactive, distributed and concurrent systems is a laborious and difficult task due to the nondeterminism of these systems. The testing process can become too expensive and inefficient.

The application of conformance testing from formal specifications represents an important branch in the efforts to make testing more rigorous and efficient. Many tools (e.g. [15,5,23]) have been developed and applied to industry experiments (e.g. [9]). However, we believe that the lack of techniques and tools for the specification of test purposes has been a great barrier to the application of conformance testing tools. Like specifications, they are usually written based on low level abstract formalisms, therefore, difficult to understand. Moreover, maintaining them based on the commonly huge systems specifications is laborious and error-prone.

As the IUT should conform to the model, properties that need to be verified against the model, also needs to be tested against the IUT. Thus, test cases must be generated based on such properties, making test purposes correspondent to them. Based on this correspondence, test purpose generation may be based on model checking, which provides efficient mechanisms to perform model analysis.

This paper aims to present an approach for the automatic generation of test purposes for reactive distributed systems based on verification techniques. Our approach uses a model checker to perform the test purpose generation. The test purpose generation consists on the specification of the properties to be verified, with later synthesis from the extracted examples and counter-examples through the model checker. This paper focus on CTL formulas, more specifically, on the EU connective. The test purposes are generated as labelled transition systems (LTS).

As main contribution to testing, we provide a rigorous automated procedure for test purpose specification and generation. The properties to be tested can be specified as an abstract formal language, more suitable for human reasoning. Moreover, the formal verification and testing processes may be linked, providing more consistency to the verification and validation based on formal methods.

The paper is organized as follows: Section 2 presents the theoretical background; Section 3 presents the approach for test purpose generation; Section 4 presents a case study for a test purpose and test case generation; Section 5 presents some
related works; Section 6 presents concluding remarks.

2 Background

As theoretic background we use the formal framework proposed in [25] and its extension presented in [6]. This framework presents the basic formal concepts used in conformance testing and provides mechanisms to test cases evaluation. The extension presented introduces the formal concept of test purposes, called observation objectives. We present some formal concepts related to the observation objectives presented in [6] relating them to the model checking theory.

2.1 Formal Test Purposes

Test purposes describe desired behaviour that must be observed during test execution. The test cases related to the test purposes are generated and executed aiming at the exhibition of the desired behaviour by the implementation. Thus, we define a relation \( \text{exhibits} \subseteq \text{IMPS} \times \text{TOBS} \), where \( \text{IMPS} \) is the domain of implementations and \( \text{TOBS} \) is the domain of test purposes.

However, implementations are not suitable for formal reasoning, making it difficult to give a formal definition to this relation. Based on the test hypothesis [1], we assume the existence of a model \( i_{IUT} \in \text{MODS} \) for the IUT, where \( \text{MODS} \) is the universe of models. Now, we can establish a relation in the formal domain making it possible to reason about exhibition. This relation is called the reveal relation, defined as \( \text{rev} \subseteq \text{MODS} \times \text{TOBS} \). Thus, for an implementation \( IUT \in \text{IMPS} \), a model of the IUT \( i_{IUT} \in \text{MODS} \) and a test purpose \( e \in \text{TOBS} \):

\[
IUT \text{ exhibits } e \iff i_{IUT} \text{ rev } e.
\]

A verdict function decides whether a test purpose is exhibited by an implementation: \( H_e : \mathcal{P}(\text{OBS}) \rightarrow \{\text{hit}, \text{miss}\} \). Then considering a test suite \( T_e \), \( IUT \text{ hits } e \text{ by } T_e \) if and only if \( i_{IUT} \text{ hits } e \text{ by } T_e \).

A test suite that is \( e \)-complete can distinguish among all exhibiting and non-exhibiting implementations, such that, \( IUT \text{ exhibits } e \) if and only if \( IUT \text{ hits } e \text{ by } T_e \). A test suite is \( e \)-exhaustive when it can only detect non-exhibiting implementations (\( IUT \text{ exhibits } e \) implies \( IUT \text{ hits } e \text{ by } T_e \)), whereas a test suite is \( e \)-sound when it can only detect exhibiting implementations (\( IUT \text{ exhibits } e \) if \( IUT \text{ hits } e \text{ by } T_e \)).

2.2 Relating Formal Test Purposes to Model Checking Theory

The model checking problem is defined in [4] as: given a kripke structure \( M \), which models a concurrent finite state system and a temporal logic formula \( f \) expressing a property \( p \), identify the set of states \( S \) of \( M \) that satisfy \( f \). Formally: \( \{s \in S \mid M, s \models f\} \).

Consider a given specification \( m_{IUT} \) as a kripke structure and a model \( i_{IUT} \in \text{MODS} \) that implements it. If there is a set of states in \( m_{IUT} \) that satisfies a

\[\text{For sake of clarity, we use the well known term test purposes throughout the paper}\]
given property $p$, then $i_{IUT}$ is able to reveal $p$. Assuming that $p$ can be expressed as a temporal logic formula $f$ and by a test purpose $e$, we can establish that: $i_{IUT} \text{ rev } e \iff \exists s \in S : m_{IUT}, s \models f$.

The states satisfying $f$ form sets of states that represent the property $p$ w.r.t. the specification $m_{IUT}$. These sets contain states related by a predecessor/successor relation, i.e., traces of the kripke structure representing $p$. As these traces correspond to abstract specifications of $p$, they may be used to guide the generation of test purposes.

### 3 Test Purpose Generation

The verification of properties through model checking has been successfully done against realistic size concurrent systems [4]. However, the same rigour is not usually applied to testing implementations, creating a large gap between these processes and making possible the presence of failures on the implementation in points where the specification was successfully corrected. Therefore, we aim to reduce this gap through the generation of test purposes from such properties specified in temporal logic formulas, based on the similarity of them.

To achieve this goal, we aim to perform analysis over the model through its state space, like model checking does. However, the process is adapted to get enough information for the test purpose generation in addition to the correctness verification of the model. The approach consists of an adaptation of a model checker algorithm [22] to extract model traces representing examples and counter-examples (if there are any) from the state space and later analysis over these traces to generate an abstract graph representing the test purpose (Fig. 1).

![Fig. 1. Test purpose generation process.](image)

The test purpose is given as an LTS. Formally, a test purpose is a tuple $e = (Q, A, \rightarrow, q_0)$, where $Q$ is a finite set of states, $A$ the alphabet of actions, $\rightarrow \subseteq Q \times A \times Q$ the transition relation and $q_0 \in Q$ the initial state. The test purpose is equipped with two sets of special states accept and refuse for sequences to be selected or not to compose test cases, respectively.
3.1 Extracting Examples and Counter-examples

The adaptation of the model checking technique consists of changes on the search algorithms of a model checker, making it possible to extract a larger number of model traces (i.e. examples and counter-examples). These traces are aimed to provide sufficient information about the model for the test purpose generation. This information is obtained from analysis over the examples and counter-examples that are made to identify the relevant transitions w.r.t. the specified property. Such transitions compose the LTS of the test purpose.

The examples are used to provide information about the accepted behaviour defined by the test purpose. The relevant transitions are then taken to construct the accept traces of the test purpose. The irrelevant ones are abstracted, usually by ”*-transitions”. Such *-transitions replace any occurring transition, except the transitions leading to another states.

Since the model checking technique is defined over transitions and states in terms of the kripke model, the use of LTS may lead to a misrepresentation of the property. The abstraction by *-transitions may be higher than the necessary to make the LTS correspondent to the formula, making possible the generation of test cases with transition sequences that may lead to property violation. To solve this problem, counter-examples of the formula, containing such undesirable transitions, are used to restrict the LTS to be generated. These transitions compose the traces leading to the refuse states of the test purpose. These states are interesting to the non-determinism problem of reactive systems too. It provides constraints on the test case generation algorithm.

3.2 Analysis and Abstraction

To simplify the analysis of the model traces, we define a simplified representation of its states in an abstract way. Thus, we represent these traces by a basic finite state machine defined by the tuple \((Q, \Sigma, \delta, q_0, F)\), where \(Q\) is a finite non-empty set of states, \(\delta\) a finite set of alphabet symbols accepted by the machine, \(\delta : Q \times \Sigma \rightarrow Q\) a transition function, \(q_0 \in Q\) an initial state and \(F \subseteq Q\) a set of final states, called accept states. The states of each trace are classified into sets defined by the propositions of the respective CTL formula, based on the satisfiability of the states w.r.t. to the formula propositions. Fig. 2 shows model traces (Fig. 2(a) and Fig. 2(b)) related to a CTL formula \(EU(p, q)\). The states of the example (Fig. 2(a)) are classified into two sets of state types, \(p\) and \(q\). The states satisfying the proposition \(p\) are called \(p\)-states and the state satisfying the proposition \(q\) is called \(q\)-state. For \(EU(p, q)\) formulas, the \(q\)-state represents the accept state of the machine.

![Fig. 2. Simplified representation of traces of the \(EU(p, q)\) formula.](image-url)
The analysis algorithm classifies the relevant and the irrelevant transitions of the traces w.r.t. to the property based on the detection of the state changes over the simplified representation of them. This is done by identification of the transition and/or sequence of transitions necessary to the state changes. This identification consists in classifying the transitions of each trace extracted into the two sets (the relevant and the irrelevant). To detect a sequence of transitions necessary to cause a state change, the algorithm performs an intersection operation over the two sets. Only sequences of transitions that can occur in alternate orders are detected, i.e. if a set of transitions causes state changes jointly, in alternate orders, traces containing a transition of such set that cause a state change must contain all of them. Therefore, two subsets of the relevant set are created, one for the transitions identified in the intersection operation and one for the others.

After the examples and counter-examples analysis and transitions classification steps, the next step performed is the test purpose generation. The transitions of the two subsets of the relevant ones are used to construct the test purpose graph: (i) leading to accept states in case of the transitions obtained from examples and (ii) leading to refuse states in case of the transitions obtained from counter-examples. Fig. 3 shows a test purpose generated from the examples of Figures 2(a) and 2(b).

3.3 The EU Test Purpose Generation Algorithm

The algorithm, shown in Algorithm 1, is based on the state changes. A partition over the transitions of the model traces must be made over two sets, L and N (lines 2-8). This partition is performed with the aid of the function leadsToQ(t,e) (line 3). Transitions that lead to a q-state, i.e. the relevant transitions, are added to the L set (line 6). Transitions that do not lead to a q-state, i.e. the irrelevant ones, are added to the N set (line 4). Fig. 4 shows a set of traces related to a given EU(p,q).
The resulting sets of examples transitions are $L = \{z, f, g\}$ and $N = \{x, y, a, b, c, d, f, g\}$. Some transitions belong to both sets (e.g., $f$ and $g$). We can conclude that these transitions cause state changes in a joint way. Thus, we intersect the two sets to obtain a third set $I = \{f, g\}$ to group such transitions in order to create the abstract graph (lines 9-16). For each example, the combination of the transitions sequences of the $I$ set must be regarded by the test purpose. Subsets based on these combinations are created, based on a predecessor relation over the transitions, to define the correspondent traces of the test purpose. The function $\text{predecessor}(t, e)$ (line 12) returns the transitions occurring earlier than a given transition $t$, in a given example $e$, regarding their orders.

The traces of the graph are created between the lines 21-37. A special set $S = \{z\}$ containing the transitions belonging only to the $L$ set is created (line 18). These transitions are used to make traces linking the initial state of the graph to the accepting state (lines 21-23). The traces containing transitions belonging to the $I$ set are made based on the sequences of transitions defined through the subsets created to regard such sequences. For each subset a trace must be made (lines 26-30). Transitions from the $I$ and $S$ sets related by the predecessor relation are verified between lines 31-35. If a transition $t$ from the $I$ set is a predecessor of a transition $j$ from the $S$ set, a trace containing such transitions must be made (lines 32-34).

The graph created from the relevant transitions of the examples is called accepting graph (Fig. 5(a)). The same procedure applied over the examples in order to create the accepting graph is applied to the counter-examples (Fig. 4(b)), generating a graph called refuse graph (Fig. 5(b)). The test purpose graph must contain the information of both graphs. The test purpose resultant from the procedure is shown in Fig. 5(c).

(a) Accepting graph  
(b) Refuse graph  
(c) Resultant test purpose

Fig. 5. Graphs obtained through the process

4 Case Study

A case study was performed with a specification of the Mobile IP protocol [19]. A test purpose was generated based on our approach and test cases were generated with the TGV tool [15].

The internet protocols do not provide dynamic addressing to mobile devices, called mobile nodes, that can migrate over the network. A migration could cause
Algorithm 1 EU Test Purpose Generation Algorithm

1: for all $e \in \text{Examples}$ do
2:     for all $t \in e$ do
3:         if $\neg \text{leadsToQ}(t,e)$ then
4:             add($t,N$)
5:         else
6:             add($t,L$)
7:         end if
8:     end for
9: $I = L \cap N$
10: for all $t \in I$ do
11:     $\text{SUB}_I = \emptyset$
12:     for all $p \in \text{predecessors}(t,e)$ do
13:         add($p,\text{SUB}_I$)
14:     end for
15:     add($t,\text{SUB}_I$)
16: end for
17: end for
18: $S = L - I$
19: $\text{TestPurpose} = \emptyset$
20: $i = 0$
21: for all $t \in S$ do
22:     add($(i,t,\text{"accept"}), \text{TestPurpose}$)
23: end for
24: for all $t \in I$ do
25:     for all $s \in \text{SUB}_I$ do
26:         if $s \neq t$ then
27:             add($(i,s,i+1),\text{TestPurpose})$
28:         else
29:             add($(i,s,\text{"accept"}),\text{TestPurpose})$
30:         end if
31:     for all $j \in S$ do
32:         if $s \in \text{predecessors}(j)$ then
33:             add($(i+1,j,\text{"accept"}),\text{TestPurpose})$
34:         end if
35:     end for
36:     $i = i + 1$
37: end for
38: end for

the connection to get lost. The Mobile IP protocol was developed to solve this problem, providing transparent migration and new IP address assignments.

To provide the transparency to the migrations the protocol provides two addresses to the mobile nodes. A home address and a foreign address, called care-of-address (COA). The home address is obtained from the home network, while the COA is obtained from the foreign network for which the mobile node is migrating to. While the mobile node (MN) is within the foreign network, the messages addressed to it are delivered by the foreign router, called foreign agent (FA). Messages sent from a host to a mobile node are addressed to the home address. The home network router, called home agent (HA), encapsulates the message within another one addressed to the COA and sends it to the foreign agent. This process is known as tunnelling. When the mobile node migrates, a COA is assigned to it and the foreign agent sends an advertisement message to the home agent.

4.1 Test Purpose Generation

The formalism used to model the protocol was RPOO [22]. RPOO is an object-oriented modelling language based on Petri Nets [16]. The model checker used to verify properties over RPOO models, and adapted to our case study, was Veritas [22], a CTL based model checking tool.

As a test purpose, we wish to reason about the conformance between IUT and model in cases messages are sent to the mobile node. While the mobile node is
home, the messages must be delivered by the home agent. We specify a simple $EU(NOT(p), q)$ formula, where $p$ means "the mobile node has migrated to the foreign network" and $q$ means "the home agent delivers the messages to the mobile node". The extraction of the model traces was based on the depth-first search algorithm, producing traces containing many states in common. We aimed to cover all $q$ states, with only one example for each $q$ state. So, examples leading to $q$ states previously selected are not extracted. Fig. 6 illustrates the depth-first selection of traces. States 4 and 6 are $q$ states, covered by the examples marked as a thick arrow. The dashed arrows indicate examples that should not be extracted. However, this strategy would miss some relevant transitions contained by the "dashed examples", not contained by the other traces. To solve this problem, in such cases, these dashed examples must be extracted. Thus, all relevant transitions related to the specified property are covered, providing a complete information for the generation of a test purpose consistent w.r.t. the CTL formula.

![Fig. 6. Depth search](image)

The analysis of the examples selected concluded that only one action was necessary to reach the $q$ states. Thus, the accepting graph only specifies it, abstracting the others with the *-transition (Fig. 7(a)). As we are not interested in cases the mobile node migrates, the counter-examples obtained represent the violation of the proposition $NOT(p)$, with the migration of the mobile node and the send of an advertisement message from the foreign agent to the home agent. The transitions representing such violation compose the refuse graph (Fig. 7(b)). The test purpose resultant from the abstraction process is shown in Fig. 8.

![Fig. 7. Graphs obtained through the process](image)

The TGV tool was used to generate the test cases from the test purpose of Fig. 8. It produced a complete test graph (CTG) through a synchronous product between the model and the test purpose. The CTG contained all the examples selected, covering all $q$ states. However, the CTG covered all the possibilities leading to the $q$ states (e.g. the dashed examples as in Fig. 6 were covered too). Therefore,

---

5 The kripke model was converted into an LTS one in the format required by TGV.
all the model traces corresponding to the CTL formula were covered by the test cases, showing the correspondence between the generated test purpose and the CTL formula w.r.t. to the model.

The generated CTG is e-exhaustive, containing infinite number of test cases. The TGV guarantees the e-soundness of the generated test cases. So, we can call the test suite composed by the CTG e-complete.

5 Related Works

Test generation using model checkers is a well explored research area. Many approaches have been proposed (e.g. [18,11]) so that model checkers are used to generate test cases directly from model traces. In these cases the test purposes are formalized as temporal logic formulas and applied to the process. However, these approaches are not based on a clear testing theory and are not appropriate to non-deterministic systems [15]. The adaptation of model checking techniques and tools to test case generation is explored in [15,5]. Based on clear theory of conformance testing they provide an exclusive process for test case generation.

In [15] the test purposes are given as an LTS, however, the technique does not provide ways to its generation. Another LTS approach to automatically produce test cases allowing checking of satisfiability of a linear property on a given implementation is discussed in [8]. This approach is based on a partial specification and an observer specified as a Rabin automata [21] to recognize the desired execution sequences. A concept of bounded properties is introduced to limit the infinite execution sequences. The partial specifications provide more flexibility to the test case generation and execution. Aiming to solve the state space explosion problem [4] of the explicit state space enumeration techniques like the based on LTS [15,8,24], symbolic approaches have been proposed [3,10].

An algorithm for the test purpose generation is presented in [12]. The approach is aimed at the identification of the significant behaviours of a system modelled as labelled event structures to generate the test purposes in form of MSC’s. Each significant behaviour is to be converted into a test purpose aiming at the generation of a test case for each one. Despite the characteristic of automation of this technique, the test purposes do not provide a higher level of abstraction w.r.t. the model. The test suite tends to be small and not exhaustive.
6 Conclusion

The presented approach makes possible the straight use of CTL properties to test purpose generation. Also, it promotes the integration of the verification and validation processes, providing a link between the model checking and conformance testing techniques. The test purposes generated through our approach represent rigorous specifications of properties to guide the generation of conformance test cases. The test case generation from such test purposes through the related theory presented in [24,25] may lead to e-complete test suites.

However, the use of temporal logic properties in the test case generation suffers from some restrictions related to the length of the test cases. Infinite executions, usually represented by liveness properties [17], are not practical to testing. Thus, test case generation techniques based on such properties must provide ways to limit the test case execution (e.g. [8]).

The generalization of the presented approach may be reached through its adaptation to nested formulas and $EG$ connective based formulas. Covering the $EU$ and $EG$ connectives suffices, once any CTL formula can be expressed in terms of these connectives. Such generalization may be obtained through a definition of special representation of examples and counter-examples for the $EG$ connective. However, we are investigating a more general representation, covering any kind of CTL formula, on which examples and counter-examples are analyzed in a joint way, distinguished only by the final states accept and refuse, respectively. The algorithms must be adapted to perform the analysis based on the new representation.

The application of the presented approach using linear temporal logic descriptions using automata on infinite words (e.g. [21,2]), like in [8], may be aimed at future works. Applying the proposed approach to finite state machines testing approach [20] constitutes another important research line. It is important to provide techniques to the problem of test case selection as well. As test case generation usually produces an infinite number of test cases, it is necessary to provide ways to select among them (e.g. coverage strategies and/or heuristics to select the most promising execution sequences [8]).

References


