MODEL CHECKING IS REFINEMENT
From Computation Tree Logic to Failure Trace Testing

Stefan D. Bruda and Zhiyu Zhang
Department of Computer Science, Bishop's University, Sherbrooke, Quebec J1M 1Z7, Canada

Keywords: Formal methods, Verification and validation, Failure trace testing, Computation tree logic, Model checking, Model-based testing, Stable failure, Temporal logic.

Abstract: Two major systems of formal conformance testing are model checking and algebraic model-based testing. Model checking is based on some form of temporal logic. One powerful and realistic logic being used is computation tree logic (CTL), which is capable of expressing most interesting properties of processes such as liveness and safety. Model-based testing is based on some operational semantics of processes (such as traces, failures, or both) and associated preorders. The most fine-grained preorder beside bisimulation (mostly of theoretical importance) is based on failure traces. We show that these two powerful variants are equivalent, in the sense that for any CTL formula there exists a set of failure trace tests that are equivalent to it. Combined with previous results, this shows that CTL and failure trace tests are equivalent.

1 INTRODUCTION

We refer to conformance testing as the process of formally proving or disproving the correctness of a system with respect to a certain specification or property. In (formal) model-based testing test cases are derived systematically and automatically from the specification; we then run the test cases against the system under test and observe the final results of the run. In model checking the specification of the system is described by temporal logic formulae; we then construct a model of the system and we check whether the model satisfies the given specification formula.

In this paper we focus on CTL, one particular temporal logic. We also focus on arguably the most powerful method of model-based testing, namely failure trace testing (Langerak, 1989).

Some properties of a system may be naturally specified using temporal logic, while others may be specified using finite automata or labelled transition systems. Such a mixed specification could be given by somebody else, but most often algebraic specifications are just more convenient for some components while logic specifications are more suitable for others. Consider some properties expressed as CTL formulae that hold for some part A of a larger system. They could be model checked so we know that part A is correct. The specification of a second part B of the same system is algebraic. We can do model-based testing on it and once more be sure that part B is correct. Now we put them together. The result is not automatically correct. We do not even have a global formal specification: part of it is logic, and other part is algebraic. Before everything else we thus need to convert one specification to the form of the other. We describe here precisely such a conversion: We show that for each CTL formula there exists an equivalent failure trace test suite. Combined with previous results (Bruda and Zhang, 2009)—where the conversion the other way around is established—we effectively show that CTL and failure trace testing are equivalent.

We are thus opening the domain of combined (algebraic and logic) methods of conformance testing. While the amount of work in both logic and algebraic areas of conformance testing is huge, work in this combined area is scarce. Specifically, we are aware of some work in relating LTL and De Nicola and Henessy testing (Cleaveland and Lüttgen, 2000) but we are not aware of any work relating CTL and failure trace testing.

2 PRELIMINARIES

A *Kripke structure* over a set AP of atomic propositions is a tuple $K = (S, S_0, \rightarrow, L)$. S is the set of states, $S_0 \subseteq S$ is the set of initial states, $\rightarrow \subseteq S \times S$

D. Bruda S. and Zhang Z. (2010).
MODEL CHECKING IS REFINEMENT - From Computation Tree Logic to Failure Trace Testing.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 173-178
DOI: 10.5220/0003006801730178
Copyright © SciTePress

is the transition relation ($s \to t$ is short for $(s,t) \in \to$), $L : S \to 2^{\mathsf{AP}}$ specifies which propositions are true in each state. A *path* in a Kripke structure is a sequence $q_0 \to q_1 \to q_2 \to \cdots$ such that $q_i \to q_{i+1}$ for all $i \geq 0$.

Temporal logics include CTL*, CTL and LTL (Clarke et al., 1999). LTL and CTL are strict subsets of CTL*. CTL* features two path quantifiers A and E (for all/some computation paths) and five basic temporal operators: X "next", F "eventually", G "always" or "globally", U "until", and R "releases"; we have state formulae (that hold in a state) and path formulae (that hold along a path). In CTL the temporal operators must be immediately preceded by a path quantifier. The syntax of CTL formulae can thus be defined as follows: With a ranging over AP, and f, f_1, f_2 ranging over state formulae,

$$\begin{aligned} f \quad = \quad & \top \mid \bot \mid a \mid \neg f \mid f_1 \wedge f_2 \mid f_1 \vee f_2 \mid \\ & \mathsf{AX}\ f \mid \mathsf{AF}\ f \mid \mathsf{AG}\ f \mid \mathsf{A}\ f_1\ \mathsf{U}\ f_2 \mid \mathsf{A}\ f_1\ \mathsf{R}\ f_2 \mid \\ & \mathsf{EX}\ f \mid \mathsf{EF}\ f \mid \mathsf{EG}\ f \mid\ \mathsf{E}\ f_1\ \mathsf{U}\ f_2 \mid \mathsf{E}\ f_1\ \mathsf{R}\ f_2 \end{aligned}$$

The semantics of CTL formulae is defined by the satisfaction operator $\vDash$. The notation $K,s \vDash f$ $[K,\pi \vDash f]$ means that in a Kripke structure K, formula f is true in state s [along path π]. The meaning of $\vDash$ is defined inductively. f and g are state formulae unless stated otherwise. We use π^i to denote the i-th state of a path π (with the first state being state 0, or π^0).

1. $K,s \vDash \top$ is true and $K,s \vDash \bot$ is false.
2. $K,s \vDash a$, $a \in \mathsf{AP}$ iff $a \in L(s)$.
3. $K,s \vDash \neg f$ iff $\neg(K,s \vDash f)$.
4. $K,s \vDash f \wedge g$ iff $K,s \vDash f$ and $K,s \vDash g$.
5. $K,s \vDash f \vee g$ iff $K,s \vDash f$ or $K,s \vDash g$.
6. $K,s \vDash \mathsf{E}\ f$ for some path formula f iff there is a path $\pi = s \to s_1 \to s_2 \to \cdots \to s_i$ s.t. $K,\pi \vDash f$.
7. $K,s \vDash \mathsf{A}\ f$ for some path formula f iff $K,\pi \vDash f$ for all paths $\pi = s \to s_1 \to s_2 \to \cdots \to s_i$.
8. $K,\pi \vDash \mathsf{X}\ f$ iff $K,\pi^1 \vDash f$.
9. $K,\pi \vDash f\ \mathsf{U}\ g$ iff there exists $j \geq 0$ such that $K,\pi^k \vDash g$ for all $k \geq j$, and $K,\pi^i \vDash f$ for all $i < j$.
10. $K,\pi \vDash f\ \mathsf{R}\ g$ iff for all $j \geq 0$, if $K,\pi^i \nvDash f$ for every $i < j$ then $K,\pi^j \vDash g$.

The common model used for system specifications in model-based testing is the labelled transition system (LTS), where the labels or formulae are associated with the transitions instead of states. In model-based testing (De Nicola and Hennessy, 1984; Tretmans, 1996) sound and complete test cases are derived from a model (an LTS) that describes some functional aspects of the system under test. The system under test is also modelled as an LTS.

An LTS is a tuple $M = (S,A,\to,s_0)$. S is a countable set of states, $s_0 \in S$ is the initial state. A is a set of labels denoting visible (or observable) events (or actions). $\to \subseteq S \times (A \cup \{\tau\}) \times S$ is the transition relation, where $\tau \notin A$ is the internal action that cannot be observed by the external environment. We often use $p \xrightarrow{a} q$ instead of $(p,a,q) \in \to$; $p \xrightarrow{a}$ is a shorthand for $\exists\ q : p \xrightarrow{a} q$. We blur the distinction between an LTS and a state, calling them both "processes" (since a state defines completely an LTS under a global $\to$).

A *path* (or *run*) π starting from state p is a sequence $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots p_{k-1} \xrightarrow{a_k} p_k$, $k \in \mathbb{N}$ such that $p_0 = p$ and $p_{i-1} \xrightarrow{a_i} p_i$, $0 < i \leq k$; $|\pi|$ is k, the length of π. The trace of π is the sequence $\mathrm{trace}(\pi) = (a_i)_{0<i\leq|\pi|, a_i \neq \tau} \in A^*$. $\Pi(p)$ denotes the set of all the paths starting from state p. $p \xRightarrow{w} p'$ states that there exists a sequence of transitions whose initial state is p, whose final state is p', and whose visible transitions form the sequence w. The notation $p \xRightarrow{w}$ stands for $\exists\ p' : p \xRightarrow{w} p'$. The traces of a process p are $\mathrm{traces}(p) = \{w : p \xRightarrow{w}\}$. The finite traces of a process p are defined as $\mathrm{Fin}(p) = \{w : p \xRightarrow{w}, |w| \in \mathbb{N}\}$.

A process p which can make no internal progress (i.e., has no outgoing internal actions) is said to be *stable* (Schneider, 2000): $p\downarrow = \neg(\exists\ p' \neq p : p \xrightarrow{\tau} p')$. If there is no action $a \in X$ to which a process p can react then p will *refuse* X: p ref X iff $\forall a \in X : \neg(\exists\ p' : p \xRightarrow{\varepsilon} p' \wedge p' \downarrow \wedge p' \xrightarrow{a})$. (w,X) is called a *stable failure* (Schneider, 2000) of p whenever $\exists\ p^w : p \xRightarrow{w} p^w \wedge p^w \downarrow \wedge p^w$ ref X. The set of stable failures of p is then $\mathrm{SF}(p) = \{(w,X) : \exists\ p^w : p \xRightarrow{w} p^w \wedge p^w \downarrow \wedge p^w \text{ ref } X\}$. Then $p \sqsubseteq_{\mathrm{SF}} q$ iff $\mathrm{Fin}(p) \subseteq \mathrm{Fin}(q)$ and $\mathrm{SF}(p) \subseteq \mathrm{SF}(q)$. We call $\sqsubseteq_{\mathrm{SF}}$ the *stable failure preorder*.

Systems and tests can be concisely described using the testing language TLOTOS (Brinksma et al., 1987; Langerak, 1989). A is the countable set of observable actions, ranged over by a and excluding the three special actions $\tau, \theta, \gamma \notin A$. The set of processes or tests is ranged over by t, t_1 and t_2; T ranges over the sets of processes or tests. The syntax of TLOTOS is then:

$$t = \mathrm{stop} \mid a;t_1 \mid \mathbf{i};t_1 \mid \theta;t_1 \mid \mathrm{pass} \mid t_1 \,\square\, t_2 \mid \Sigma T.$$

The semantics of TLOTOS is defined as follows:

1. inaction (stop): no rules.
2. action prefix: $a;t_1 \xrightarrow{a} t_1$ and $\mathbf{i};t_1 \xrightarrow{\tau} t_1$
3. deadlock detection: $\theta;t_1 \xrightarrow{\theta} t_1$.
4. successful termination: $\mathrm{pass} \xrightarrow{\gamma} \mathrm{stop}$.
5. choice: with $g \in A \cup \{\gamma,\theta,\tau\}$, $\dfrac{t_1 \xrightarrow{g} t_1'}{t_1 \,\square\, t_2 \xrightarrow{g} t_1',\ t_2 \,\square\, t_1 \xrightarrow{g} t_1'}$.

6. generalized choice: with $g \in A \cup \{\gamma,\theta,\tau\}$, $t_1 \xrightarrow{g} t_1'$: $\Sigma(\{t_1\} \cup T) \xrightarrow{g} t_1'$.

γ signals the successful completion of a test and θ is the deadlock detection label. Any *process* (or LTS) can be described as a TLOTOS process not containing γ and θ. A *failure trace test* on the other hand is a full TLOTOS process, i.e., may contain γ and θ. A test runs in parallel with the system under test according to the parallel composition operator $\|_\theta$, which defines the semantics of θ as the lowest priority action:

$$\frac{p \xrightarrow{\tau} p'}{p\|_\theta t \xrightarrow{\tau} p'\|_\theta t} \qquad \frac{t \xrightarrow{\tau} t'}{p\|_\theta t \xrightarrow{\tau} p'\|_\theta t}$$

$$\frac{t \xrightarrow{\gamma} \mathrm{stop}}{p\|_\theta t \xrightarrow{\gamma} \mathrm{stop}} \qquad \frac{p \xrightarrow{a} p' \quad t \xrightarrow{a} t'}{p\|_\theta t \xrightarrow{a} p'\|_\theta t'}\ a \in A$$

$$\frac{t \xrightarrow{\theta} t'}{p\|_\theta t \xrightarrow{\theta} p\|_\theta t'}\ \neg\exists\, x \in A \cup \{\tau,\gamma\} : p\|_\theta t \xrightarrow{x}$$

The outcome of $\pi \in \Pi(p\|_\theta t)$ is success ($\top$) whenever the last symbol in trace(π) is γ, and failure ($\bot$) otherwise. The set of outcomes of all the runs in $\Pi(p\|_\theta t)$ is denoted by $\mathbb{O}(p,t)$. Then p may t iff $\top \in \mathbb{O}(p,t)$, and p must t iff $\{\top\} = \mathbb{O}(p,t)$. $\sqsubseteq_{\mathrm{SF}}$ can be characterized in terms of may testing only:

Proposition 1. (Langerak, 1989). *Let p_1 and p_2 be processes. Then $p_1 \sqsubseteq_{\mathrm{SF}} p_2$ iff p_1 may $t \implies p_2$ may t for all failure trace tests t.*

3 PREVIOUS WORK

We summarize our previous results intimately related to this paper (Bruda and Zhang, 2009). We define first an LTS satisfaction operator similar to the one on Kripke structures in a natural way.

Definition 1. *(Bruda and Zhang, 2009). A process p satisfies $a \in A$, written by abuse of notation $p \models a$, iff $p \xrightarrow{a}$. That p satisfies some (general) CTL* state formula is defined inductively as follows: Let f and g be some state formulae unless stated otherwise; then,*

1. *$p \models \top$ is true and $p \models \bot$ is false for any process p.*
2. *$p \models \neg f$ iff $\neg(p \models f)$.*
3. *$p \models f \wedge g$ iff $p \models f$ and $p \models g$.*
4. *$p \models f \vee g$ iff $p \models f$ or $p \models g$.*
5. *$p \models \mathsf{E}\ f$ for some path formula f iff there is a path $\pi = p \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ such that $\pi \models f$.*
6. *$p \models \mathsf{A}\ f$ for some path formula f iff $p \models f$ for all paths $\pi = p \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$.*
7. *$\pi \models \mathsf{X}\ f$ iff $\pi^1 \models f$.*
8. *$\pi \models f\ \mathsf{U}\ g$ iff there exists $j \geq 0$ such that $\pi^j \models g$ and $\pi^k \models g$ for all $k \geq j$, and $\pi^i \models f$ for all $i < j$.*
9. *$\pi \models f\ \mathsf{R}\ g$ iff for all $j \geq 0$, if $\pi^i \not\models f$ for every $i < j$ then $\pi^j \models g$.*

We also need to define a weaker satisfaction operator for CTL*.

Definition 2. (Bruda and Zhang, 2009). *Consider a Kripke structure $K = (S, S_0, \rightarrow, L)$ over AP. For some set $Q \subseteq S$ and some CTL* state formula f we define $K, Q \models f$ as follows, with f and g state formulae unless stated otherwise:*

1. *$K,Q \models \top$ is true and $K,Q \models \bot$ is false for any set Q in any Kripke structure K.*
2. *$K,Q \models a$, $a \in \mathsf{AP}$ iff $a \in L(s)$ for some $s \in Q$.*
3. *$K,Q \models \neg f$ iff $\neg(K,Q \models f)$.*
4. *$K,Q \models f \wedge g$ iff $K,Q \models f$ and $K,Q \models g$.*
5. *$K,Q \models f \vee g$ iff $K,Q \models f$ or $K,Q \models g$.*
6. *$K,Q \models \mathsf{E}\ f$ for some path formula f iff for some $s \in Q$ there exists a path $\pi = s \rightarrow s_1 \rightarrow \cdots$ such that $K,\pi \models f$.*
7. *$K,Q \models \mathsf{A}\ f$ for some path formula f iff for some $s \in Q$ it holds that $K,\pi \models f$ for all paths $\pi = s \rightarrow s_1 \rightarrow \cdots$.*

We introduce the following equivalence relation:

Definition 3. (Bruda and Zhang, 2009). *Given a Kripke structure K and a set of states Q, K,Q is equivalent to a process p, written $K,Q \simeq p$ (or $p \simeq K,Q$), iff for any CTL* formula f, $K,Q \models f$ iff $p \models f$.*

Proposition 2. (Bruda and Zhang, 2009). *There exists an algorithmic function ξ which converts an LTS p into a Kripke structure K and a set of states Q such that $p \simeq (K,Q)$.*

Specifically, for any LTS $p = (S, A, \rightarrow, s_0)$, its equivalent Kripke structure K is defined as $K = (S', Q, R', L')$ where $S' = \{\langle s,x\rangle : s \in S, x \subseteq \mathrm{init}(s)\}$, $Q = \{\langle s_0,x\rangle \in S'\}$, $L' : S' \rightarrow 2^A$ such that $L'(s,x) = x$, and R' contains exactly all the transitions $(\langle s,N\rangle, \langle t,O\rangle) \in S' \times S'$ such that: (a) for any $n \in N$, $s \stackrel{n}{\Longrightarrow} t$, (b) for some $q \in S$ and for any $o \in O$, $t \stackrel{o}{\Longrightarrow} q$, and (c) if $N = \emptyset$ then $O = \emptyset$ and $t = s$ (these loops ensure that the relation R' is complete).

Using such a conversion we can define the semantics of CTL* formulae with respect to a process rather than Kripke structure.

Let now $\mathcal{P}$ be the set of all processes, $\mathcal{T}$ the set of all the failure trace tests, and $\mathcal{F}$ the set of all CTL formulae. We have:

Proposition 3. (Bruda and Zhang, 2009). *There exists a function $\psi : \mathcal{T} \rightarrow \mathcal{F}$ such that for any process p, p may t iff $\xi(p) \models \psi(t)$.*

4 CTL IS EQUIVALENT TO FAILURE TRACE TESTING

We go now in the opposite direction from Proposition 3 and show that CTL formulae can be converted into failure trace tests. Recall that $\mathcal{P}$ is the set of processes, $\mathcal{T}$ the set of failure trace tests, and $\mathcal{F}$ the set of CTL formulae. By abuse of notation we write p may T for some $T \subseteq \mathcal{T}$ iff p may t for all $t \in T$.

Theorem 4. *There exists a function $\omega : \mathcal{F} \to 2^{\mathcal{T}}$ such that for any process p, $\xi(p) \models f$ iff p may $\omega(f)$.*

Proof. The proof is done by structural induction over CTL formulae.

Let $\omega(\top) = \{\text{pass}\}$. Any Kripke structure satisfies $\top$ and any process passes pass, so $\xi(p) \models \top$ iff p may $\{\text{pass}\} = \omega(\top)$. Similarly $\xi(p) \models \bot$ iff p may $\{\text{stop}\}$ (namely, never!), so we put $\omega(\bot) = \{\text{stop}\}$. We then put $\omega(a) = \{a;\text{pass}\}$, noting that $\xi(p) \models a$ iff p may $a;$pass by the definition of ξ.

For exactly all the tests $t \in \omega(f)$ we add t' to $\omega(\neg f)$, where t' is generated out of t using the following algorithm: We force all the successful states to become deadlock states (i.e., we eliminate γ from t); whenever the test would have reached a successful state, it now reaches a deadlock state and fails. Then we introduce an action θ followed by an action γ (success) to all the states except the ones that were success states originally; this way, t' can succeed in any state other than the original success states. This conversion is very similar to the construction of a finite automaton that accepts the complement of a given language (Lewis and Papadimitriou, 1998), and its correctness can be established using a similar argument.

We put $\omega(f_1 \wedge f_2) = \omega(f_1) \cup \omega(f_2)$. Indeed, $\xi(p) \models f_1 \wedge f_2$ iff $\xi(p) \models f_1$ and $\xi(p) \models f_2$ iff p may $\omega(f_1)$ and p may $\omega(f_2)$ (by induction hypothesis) iff p may $\omega(f_1) \cup \omega(f_2)$. Whenever $\xi(p) \models f_1 \wedge f_2$ the process p should pass all of the tests in $\omega(f_1)$ and $\omega(f_2)$ (and the other way around).

We have $\omega(f_1 \vee f_2) = \omega(\neg(\neg f_1 \wedge \neg f_2))$ by De Morgan rules ($\neg(f_1 \vee f_2) = \neg f_1 \wedge \neg f_2$) and the previous definitions of $\omega(\neg f)$ and $\omega(f_1 \wedge f_2)$.

Now we put $\omega(\mathsf{EX}\ f) = \{\Sigma\{a;t : a \in A\} : t \in \omega(f)\}$. As shown in Figure 1, the test suite combines a choice of action from all the available actions with the tests from $\omega(f)$. p may $\omega(\mathsf{EX}\ f)$ iff p passes each all test cases above, equivalent to p being able to perform an action (any action!) and then pass the tests in $\omega(f)$. By the inductive assumption that $\omega(f)$ is equivalent to f, the above is equivalent to $\xi(p) \models \mathsf{EX}\ f$ (namely, perform any action then satisfy f).

We then have $\omega(\mathsf{AX}\ f) = \{a;t \,\Box\, \theta;\text{pass} : a \in A, t \in \omega(f)\}$. As shown in Figure 2, the test suite is generated by combining an action and a test from $\omega(f)$. When the action is not provided, a deadlock detection transition takes place and leads to a pass state (so that particular test does not play any role). The test suite is generated by providing all the possible actions; the system under test however does not necessarily have to perform all the possible actions before going to the point where the tests are from $\omega(f)$. When the system under test runs in parallel with the test case, we get a deadlock whenever the respective action is not encountered in the system; this leads to a pass state and then we continue to check the results of the other runs with the other test cases. p may $\omega(\mathsf{AX}\ f)$ iff p passes each of the test cases above, i.e., whenever p can perform an action then after performing it (in the next state) it passes all the tests in $\omega(f)$ (which by inductive assumption is equivalent to the formula f).

We put $\omega(\mathsf{EF}\ f) = \{t' = \mathbf{i};t \,\Box\, \mathbf{i};(\Sigma\{a;t' : a \in A\}) : t \in \omega(f)\}$. A (slightly simplified) way of depicting each test from $\omega(\mathsf{EF}\ f)$ graphically is shown in Figure 3(a). The test suite is generated by combining a choice of actions and the tests in $\omega(f)$. Then, we combine a choice of action followed by another choice of action with the tests in $\omega(f)$, and so on until the last layer of the the Kripke structure (viewed as a tree). The resulting test suite is highly nondeterministic. p satisfies $\mathsf{EF}\ f$ iff p passes $\omega(f)$, or p performs one action and in the next state passes $\omega(f)$, or p can perform two actions and then passes $\omega(f)$ and so on. Clearly this corresponds to the formula $\mathsf{EF}\ f$ given the induction hypothesis that $\omega(f)$ is equivalent to f.

The conversion of the remaining CTL operators is partially based on "unfolding" these operators using the operators considered above, nothing that we have already established the test sets for these operators.

Let $\omega(\mathsf{AF}\ f) = \{\mathbf{i};t \,\Box\, \mathbf{i};t' : t \in \omega(f), t' \in \omega(\mathsf{AX}\ f')\}$, with $f' = f \vee \mathsf{AX}\ f'\}$; f' is a recursive definition. Unfolding f' yields f or $\mathsf{AX}\ (f \vee \mathsf{AX}\ f')$ or $\mathsf{AX}\ (f \vee \mathsf{AX}\ (f \vee \mathsf{AX}\ f'))$, and so on. The process should satisfy any of the unfolded formulae in order to satisfy $\mathsf{AF}\ f$. That is, the root state of a Kripke structure must satisfy f (the root being common to every path, this satisfies the original formula); otherwise, every next state either satisfies f (so the respective path is delivered from all its obligations) or has a set of next states that all satisfy (recursively) f', meaning that they satisfy the same requirement. In terms of LTS, a process either passes $\omega(f)$ or performs some action to the second layer states; all of these (second layer) states now either satisfy $\omega(f)$ themselves or have a set of next states that all satisfy (recursively) $\omega(f')$. These are clearly equivalent.

Similarly, we put $\omega(\mathsf{EG}\ f) = \omega(f) \cup \omega(\mathsf{EX}\ f')$, with $f' = f \wedge \mathsf{EX}\ f'$. When we unfold f' we get f and

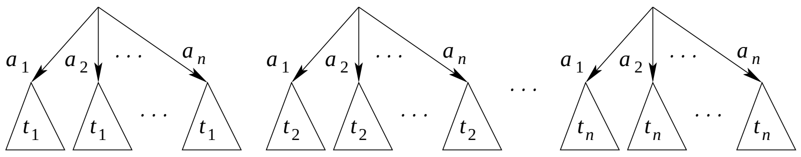

Figure 1: Test suite for the CTL formula $\mathsf{EX}\ f$.

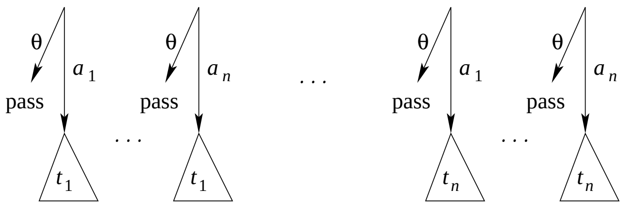

Figure 2: Test suite for the CTL formula $\mathsf{AX}\ f$.

$\mathsf{EX}\ (f \wedge \mathsf{EX}\ f')$ and $\mathsf{EX}\ (f \wedge \mathsf{EX}\ (f \wedge \mathsf{EX}\ f'))$ and so on. The process should satisfy all of the unfolded formulae in order for the the process satisfy the original formula $\mathsf{EG}\ f$. In a Kripke structure, the states in the first layer need to satisfy f and then some next states (in the second layer) need to satisfy the formula f and also need to have some next states (in the third layer) that satisfy (recursively) f'. In terms of LTS, the process need to pass $\omega(f)$ and then perform some action to the second layer states that then pass $\omega(f)$ and also (recursively) $\omega(f')$. Again, these are equivalent.

Once more similarly we put $\omega(\mathsf{AG}\ f) = \omega(f) \cup \omega(\mathsf{AX}\ f')$, $f' = f \wedge \mathsf{AX}\ f'$. Unfolding f' yields f and $\mathsf{AX}\ (f \wedge \mathsf{AX}\ f')$ and $\mathsf{AX}\ (f \wedge \mathsf{AX}\ (f \wedge \mathsf{AX}\ f'))$, etc. Figure $3(b)$ illustrates that the test suite is generated by combining a choice of action and the tests in $\omega(f)$, then a choice of two actions followed by the tests in $\omega(f)$, etc. p satisfies $\mathsf{AG}\ f$ iff p passes the tests in $\omega(f)$, i.e., p can perform an action and then in the next states pass the tests in $\omega(f)$, and p can perform two actions and then pass the tests from $\omega(f)$, etc.

Finally, $\omega(\mathsf{E}\ f_1\ \mathsf{U}\ f_2) = \{t_1 \Box \mathbf{i}; t_2 : t_1 \in \omega(f_1) \cup \omega(\mathsf{EX}\ f'), t_2 \in \omega(f_2) \cup \omega(\mathsf{EX}\ f'')\}$, with $f' = f_1 \wedge \mathsf{EX}\ f'$ and $f'' = f_2 \wedge \mathsf{EX}\ f''$. This is similar to the $\mathsf{EG}\ f$ case, but now every path is allowed at some point to switch from states satisfying f' to states satisfying f''. Unfolding f' and f'' yield f_1 and $\mathsf{EX}\ (f_1 \wedge \mathsf{EX}\ f')$ and $\mathsf{EX}\ (f_1 \wedge \mathsf{EX}\ (f_1 \wedge \mathsf{EX}\ f'))$ and so on, until we change via an internal action to f_2 and $\mathsf{EX}\ (f_2 \wedge \mathsf{EX}\ f'')$ and $\mathsf{EX}\ (f_2 \wedge \mathsf{EX}\ (f_2 \wedge \mathsf{EX}\ f''))$, etc. In a Kripke structure, the states in the first layer need to satisfy the formula f_1 and then some successive states in the second layer need to satisfy the formula f_1, etc.. At some point however, some states need to satisfy the formula f_2 and from then on some successive states need to satisfy f_2 along the whole path. In terms of LTS, the process need to pass $\omega(f_1)$ and then perform some action to the second layer of states where some state needs to pass $\omega(f_1)$ again, and so on until some point where some state passes $\omega(f_2)$; from then on, some state from every layer needs to pass $\omega(f_2)$.

All the remaining CTL constructs can be rewritten using only the constructs discussed above. Indeed, $\mathsf{A}\ f_1 \mathsf{R}\ f_2 \equiv \neg\mathsf{E}\ (\neg f_1\ \mathsf{U}\ \neg f_2)$, $\mathsf{E}\ f_1\ \mathsf{R}\ f_2 \equiv \neg\mathsf{A}\ (\neg f_1\ \mathsf{U}\ \neg f_2)$. and $\mathsf{A}\ f_1\ \mathsf{U}\ f_2 \equiv \neg\mathsf{E}\ (\neg f_2\ \mathsf{U}\ (\neg f_1 \wedge \neg f_2)) \wedge \neg\mathsf{EG}\ (\neg f_2)$. The proof is thus complete. □

5 CONCLUSIONS

We defined previously a function ψ that converts any failure trace test into an equivalent CTL formula. Now we defined ω, the function that convert CTL formulae into equivalent failure trace test suites.

It is worth mentioning that the function ξ creates a Kripke structure that may have multiple initial states, and so we were forced to use a weaker satisfaction operator. We note however that such an issue manifests itself only when the LTS being converted exhibits *initial nondeterminism* (Bruda and Zhang, 2009), which can be eliminated by creating a new start state that performs a "start" action and then gives control to the original LTS. Our results are thus without loss of generality: in the absence of initial nondeterminism the proofs of Theorem 4 (and also Propositions 2 and 3) revert to the normal satisfaction operator.

This work opens the way toward a combined, logical and algebraic approach to conformance testing. This is extremely important for large systems with components at different level of maturity. The canonic example is a communication protocol: the end points are algorithms that are likely to be amenable to algebraic specification, while the communication medium is something we don't know much about. It could be a direct link, a local network or something else. However, its properties are ex-

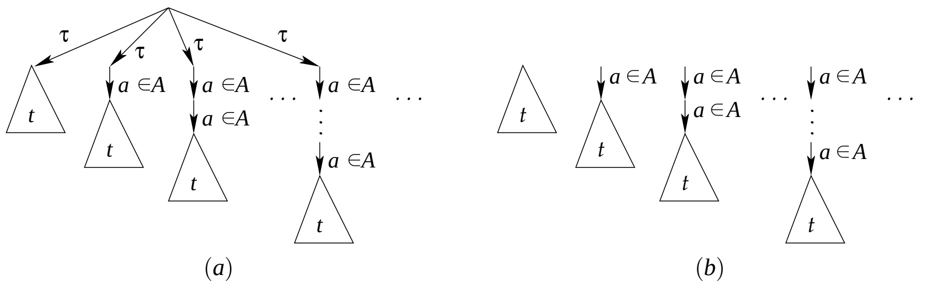

Figure 3: Test suite for the CTL formulae $\mathsf{EF}\ f\ (a)$ and $\mathsf{AG}\ f\ (b)$. The suite is depicted partially, namely only for one $t \in \omega(f)$.

pressible using temporal logic formulae. In general, our conversions allow the use of the fastest, most suitable, or even most preferred method of conformance testing, irrespective to the form of the specification.

Our results are important first steps toward a unified (logic and algebraic) approach to conformance testing. We believe in particular that this paper opens several direction of future research. The main challenge in the method we introduced is dealing with the infinite-state test cases. Indeed, the test cases produced from CTL temporal logic formulae are infinite. This is fine theoretically, but from a practical perspective it is worthy of future work to eliminate infinite states or to obtain usable algorithms than simulate runs of infinite-state test suites with the system under test and obtain useful results in finite time. The tests developed here can be combined with partial application so that another interesting research direction is to find partial application algorithms that yield total correctness at the limit and have some correctness insurance milestones along the way.

On the conversion the other way around (from tests to CTL formulae) we note that the obtained formulae $\psi(t)$ may not be in their simplest (or more concise) form possible in several respects. More significantly, the formulae may even have an infinite length (this happens for infinite tests), since they don't really exploit the operators G, F, U, and R. We have strong reasons to believe that bringing them to more manageable proportions is possible, and this is one subject of our research.

ACKNOWLEDGEMENTS

This research was supported by the Natural Sciences and Engineering Research Council of Canada. Part of this research was also supported by Bishop's University.

REFERENCES

Brinksma, E., Scollo, G., and Steenbergen, C. (1987). LOTOS specifications, their implementations and their tests. In *IFIP 6.1 Proceedings*, pages 349–360.

Bruda, S. D. and Zhang, Z. (2009). Refinment is model checking: From failure trace tests to computation tree logic. In *Proceedings of the 13th IASTED International Conference on Software Engineering and Applications (SEA 09)*, Cambridge, MA.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press.

Cleaveland, R. and Lüttgen, G. (2000). Model checking is refinment—Relating Büchi testing and linear-time temporal logic. Technical Report 2000-14, ICASE, Langley Research Center, Hampton, VA.

De Nicola, R. and Hennessy, M. C. B. (1984). Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133.

Langerak, R. (1989). A testing theory for LOTOS using deadlock detection. In *Proceedings of the IFIP WG6.1 Ninth International Symposium on Protocol Specification, Testing and Verification IX*, pages 87–98.

Lewis, H. R. and Papadimitriou, C. H. (1998). *Elements of the Theory of Computation*. Prentice-Hall, 2nd edition.

Schneider, S. (2000). *Concurrent and Real-time Systems: The CSP Approach*. John Wiley & Sons.

Tretmans, J. (1996). Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79.