Technical Report 2019-001
A Constructive Equivalence between Computation Tree Logic and Failure Trace Testing*

Stefan D. Bruda, Sunita Singh, A. F. M. Nokib Uddin, Zhiyu Zhang, and Rui Zuo
Department of Computer Science
Bishop's University
Sherbrooke, Quebec J1M 1Z7, Canada
stefan@bruda.ca, {singh|uddin|zzhang|zuo}@cs.ubishops.ca
14 January 2019

Abstract

The two major systems of formal verification are model checking and algebraic model-based testing. Model checking is based on some form of temporal logic such as linear temporal logic (LTL) or computation tree logic (CTL). One powerful and realistic logic being used is 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 its associated preorders. The most fine-grained preorder beside bisimulation (mostly of theoretical importance) is based on failure traces. We show that these two most powerful variants are equivalent; that is, we show that for any failure trace test there exists a CTL formula equivalent to it, and the other way around. All our proofs are constructive and algorithmic. Our result allows for parts of a large system to be specified logically while other parts are specified algebraically, thus combining the best of the two (logic and algebraic) worlds.

Keywords: model checking, model-based testing, stable failure, failure trace, failure trace preorder, temporal logic, computation tree logic, labelled transition system, Kripke structure

1 Introduction

Computing systems are already ubiquitous in our everyday life, from entertainment systems at home, to telephone networks and the Internet, and even to health care, transportation, and energy infrastructure. Ensuring the correct behaviour of software and hardware has been one of the goals of Computer Science since the dawn of computing. Since then computer use has skyrocketed and so has the need for assessing correctness.

Historically the oldest verification method, which is still widely used today, is empirical testing [22, 27]. This is a non-formal method which provides input to a system, observes the output, and verifies that the output is the one expected given the input. Such testing cannot check all the possible input combinations and so it can disprove correctness but can never prove it. Deductive verification [16, 18, 25] is chronologically the next verification method developed. It consists of providing proofs of program correctness manually, based on a set of axioms and inference rules. Program proofs provide authoritative evidence of correctness but are time consuming and require highly qualified experts.

Various techniques have been developed to automatically perform program verification with the same effect as deductive reasoning but in an automated manner. These efforts are grouped together in the general field of formal methods. The general technique is to verify a system automatically against some formal specification. Model-based testing and model checking are the two approaches to formal methods that became mainstream. Their roots can be traced to simulation and deductive reasoning, respectively.

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

These formal methods however are sound, complete, and to a large extent automatic. They have proven themselves through the years and are currently in wide use throughout the computing industry.

In model-based testing [4, 13, 29] the specification of a system is given algebraically, with the underlying semantics given in an operational manner as a labeled transition system (LTS for short), or sometimes as a finite automaton (a particular, finite kind of LTS). Such a specification is usually an abstract representation of the system's desired behaviour. The system under test is modeled using the same formalism (either finite or infinite LTS). The specification is then used to derive systematically and formally tests, which are then applied to the system under test. The way the tests are generated ensures soundness and completeness. In this paper we focus on arguably the most powerful method of model-based testing, namely failure trace testing [20]. Failure trace testing also introduces a smaller set (of sequential tests) that is sufficient to assess the failure trace relation.

By contrast, in model checking [9, 10, 24] the system specification is given in some form of temporal logic. The specification is thus a (logical) description of the desired properties of the system. The system under test is modeled as Kripke structures, another formalism similar to transition systems. The model checking algorithm then determines whether the initial states of the system under test satisfy the specification formulae, in which case the system is deemed correct. There are numerous temporal logic variants used in model checking, including CTL*, CTL and LTL. In this paper we focus on CTL.

There are advantages as well as disadvantages to each of these formal methods techniques. Model checking is a complete verification technique, which has been widely studied and also widely used in practice. The main disadvantage of this technique is that it is not compositional. It is also the case that model checking is based on the system under test being modeled using a finite state formalism, and so does not scale very well with the size of the system under test. By contrast, model-based testing is compositional by definition (given its algebraic nature), and so has better scalability. In practice however it is not necessarily complete given that some of the generated tests could take infinite time to run and so their success or failure cannot be readily ascertained. The logical nature of specification for model checking allows us to only specify the properties of interest, in contrast with the labeled transition systems or finite automata used in model-based testing which more or less require that the whole system be specified.

Some properties of a system may be naturally specified using temporal logic, while others may be specified using finite automata or labeled 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. However, such a mixed specification cannot be verified. Parts of it can be model checked and some other parts can be verified using model-based testing. However, no global algorithm for the verification of the whole system exists. Before even thinking of verifying such a specification we need to convert one specification to the form of the other.

We describe in this paper precisely such a conversion. We first propose two equivalence relations between labeled transition systems (the semantic model used in model-based testing) and Kripke structures (the semantic model used in model checking), and then we show that for each CTL formula there exists an equivalent failure trace test suite, and the other way around. In effect, we show that the two (algebraic and logic) formalisms are equivalent. All our proofs are constructive and algorithmic, so that implementing back and forth automated conversions is an immediate consequence of our result.

We believe that we are thus opening the domain of combined, algebraic and logic methods of formal system verification. The advantages of such a combined method stem from the above considerations but also from the lack of compositionality of model checking (which can thus be side-stepped by switching to algebraic specifications), from the lack of completeness of model-based testing (which can be side-stepped by switching to model checking), and from the potentially attractive feature of model-based testing of incremental application of a test suite insuring correctness to a certain degree (which the all-or-nothing model-checking lacks).

The reminder of this paper is organized as follows: We introduce basic concepts including model checking, temporal logic, model-based testing, and failure trace testing in the next section. Previous work is reviewed briefly in Section 3. Section 4 defines the concept of equivalence between LTS and Kripke structures, together with an algorithmic function for converting an LTS into its equivalent Kripke structure. Two such equivalence relations and conversion functions are offered (Section 4.1 and 4.2, respectively). Section 5 then presents our algorithmic conversions from failure trace tests to CTL formulae (Section 5.1 with an improvement in Section 5.2) and the other way around (Section 5.3). We discuss the significance and consequences of our work in Section 6. For the remainder of this paper results proved elsewhere are introduced as Propositions, while original results are stated as Theorems, Lemmata, or Corollaries.

2 Preliminaries

This section is dedicated to introducing the necessary background information on model checking, temporal logic, and failure trace testing. For technical reasons we also introduce TLOTOS, a process algebra used for describing algebraic specifications, tests, and systems under test. The reason for using this particular language is that earlier work on failure trace testing uses this language as well.

Given a set of symbols A we use as usual A^* to denote exactly all the strings of symbols from A. The empty string, and only the empty string is denoted by ε. We use ω to refer to $|\mathbb{N}|$, the cardinality of the set $\mathbb{N}$ of natural numbers. The power set of a set A is denoted as usual by 2^A.

2.1 Temporal Logic and Model Checking

A specification suitable for model checking is described by a temporal logic formula. The system under test is given as a Kripke structure. The goal of model checking is then to find the set of all states in the Kripke structure that satisfy the given logic formula. The system then satisfies the specification provided that all the designated initial states of the respective Kripke structure satisfy the logic formula.

Formally, a *Kripke structure* [10] K over a set AP of atomic propositions is a tuple $(S, S_0, \rightarrow, L)$, where S is a set of states, $S_0 \subseteq S$ is the set of initial states, $\rightarrow \subseteq S \times S$ is the transition relation, and $L : S \rightarrow 2^{\mathsf{AP}}$ is a function that assigns to each states exactly all the atomic propositions that are true in that state. As usual we write $s \rightarrow t$ instead of $(s,t) \in \rightarrow$. It is usually assumed [10] that $\rightarrow$ is total, meaning that for every state $s \in S$ there exists a state $t \in S$ such that $s \rightarrow t$. Such a requirement can however be easily established by creating a "sink" state that has no atomic proposition assigned to it, is the target of all the transitions from states with no other outgoing transitions, and has one outgoing "self-loop" transition back to itself.

A *path* π in a Kripke structure is a sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$ such that $s_i \rightarrow s_{i+1}$ for all $i \geq 0$. The path starts from state s_0. Any state may be the start of multiple paths. It follows that all the paths starting from a given state s_0 can be represented together as a computation tree with nodes labeled with states. Such a tree is rooted at s_0 and (s,t) is an edge in the tree if and only if $s \rightarrow t$. Some temporal logics reason about computation paths individually, while some other temporal logics reason about whole computation trees.

There are several temporal logics currently in use. We will focus in this paper on the CTL* family [10, 15] and more precisely on the CTL variant. CTL* is a general temporal logic which is usually restricted for practical considerations. One such a restriction is the linear-time temporal logic or LTL [10, 23], which is an example of temporal logic that represents properties of individual paths. Another restriction is the computation tree logic or CTL [8, 10], which represents properties of computation trees.

In CTL* the properties of individual paths are represented using five temporal operators: X (for a property that has to be true in the next state of the path), F (for a property that has to eventually become true along the path), G (for a property that has to hold in every state along the path), U (for a property that has to hold continuously along a path until another property becomes true and remains true for the rest of the path), and R (for a property that has to hold along a path until another property becomes true and releases the first property from its obligation). These path properties are then put together so that they become state properties using the quantifiers A (for a property that has to hold on all the outgoing paths) and E (for a property that needs to hold on at least one of the outgoing paths).

CTL is a subset of CTL*, with the additional restriction that the temporal constructs X, F, G, U, and R must be immediately preceded by one of the path quantifiers A or E. More precisely, the syntax of CTL formulae is defined as follows:

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

where $a \in \mathsf{AP}$, and f, f_1, f_2 are all state formulae.

CTL formulae are interpreted over states in Kripke structures. Specifically, the CTL semantics is given by the operator $\models$ such that $K, s \models f$ means that the formula f is true in the state s of the Kripke structure K. All the CTL formulae are state formulae, but their semantics is defined using the intermediate concept of path formulae. In this context the notation $K, \pi \models f$ means that the formula f is true along the path π in the Kripke structure K. The operator $\models$ is defined inductively as follows:

1. $K, s \models \top$ is true and $K, s \models \bot$ is false for any state s in any Kripke structure K.

2. $K, s \models a$, $a \in \mathsf{AP}$ if and only if $a \in L(s)$.

3. $K, s \models \neg f$ if and only if $\neg(K, s \models f)$ for any state formula f.

4. $K, s \models f \wedge g$ if and only if $K, s \models f$ and $K, s \models g$ for any state formulae f and g.

5. $K, s \models f \vee g$ if and only if $K, s \models f$ or $K, s \models g$ for any state formulae f and g.

6. $K, s \models \mathsf{E}\ f$ for some path formula f if and only if there exists a path $\pi = s \to s_1 \to s_2 \to \cdots \to s_i$, $i \in \mathbb{N} \cup \{\omega\}$ such that $K, \pi \models f$.

7. $K, s \models \mathsf{A}\ f$ for some path formula f if and only if $K, \pi \models f$ for all paths $\pi = s \to s_1 \to s_2 \to \cdots \to s_i$, $i \in \mathbb{N} \cup \{\omega\}$.

We use π^i to denote the i-th state of a path π, with the first state being π^0. The operator $\models$ for path formulae is then defined as follows:

1. $K, \pi \models \mathsf{X}\ f$ if and only if $k, \pi^1 \models f$ for any state formula f.

2. $K, \pi \models f\ \mathsf{U}\ g$ for any state formulae f and g if and only if there exists $j \geq 0$ such that $K, \pi^k \models g$ for all $k \geq j$, and $K, \pi^i \models f$ for all $i < j$.

3. $K, \pi \models f\ \mathsf{R}\ g$ for any state formulae f and g if and only if for all $j \geq 0$, if $K, \pi^i \not\models f$ for every $i < j$ then $K, \pi^j \models g$.

2.2 Labeled Transition Systems and Stable Failures

CTL semantics is defined over Kripke structures, where each state is labeled with atomic propositions. By contrast, the common model used for system specifications in model-based testing is the labeled transition system (LTS), where the labels (or actions) are associated with the transitions instead.

An LTS [19] is a tuple $M = (S, A, \rightarrow, s_0)$ where S is a countable, non empty set of states, $s_0 \in S$ is the initial state, and A is a countable set of actions. The actions in A are called visible (or observable), by contrast with the special, unobservable action $\tau \notin A$ (also called internal action). The relation $\rightarrow \subseteq S \times (A \cup \{\tau\}) \times S$ is the transition relation; we use $p \xrightarrow{a} q$ instead of $(p, a, q) \in \rightarrow$. A transition $p \xrightarrow{a} q$ means that state p becomes state q after performing the (visible or internal) action a.

The notation $p \xrightarrow{a}$ stands for $\exists p' : p \xrightarrow{a} p'$. The sets of states and transitions can also be considered global, in which case an LTS is completely defined by its initial state. We therefore blur whenever convenient the distinction between an LTS and a state, calling them both "processes". Given that $\rightarrow$ is a relation rather than a function, and also given the existence of the internal action, an LTS defines a nondeterministic process.

A *path* (or *run*) π starting from state p' is a sequence $p' = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots p_{k-1} \xrightarrow{a_k} p_k$ with $k \in \mathbb{N} \cup \{\omega\}$ such that $p_{i-1} \xrightarrow{a_i} p_i$ for all $0 < i \leq k$. We use $|\pi|$ to refer to k, the length of π. If $|\pi| \in \mathbb{N}$, then we say that π is finite. The trace of π is the sequence $\text{trace}(\pi) = (a_i)_{0<i\leq|\pi|, a_i \neq \tau} \in A^*$ of all the visible actions that occur in the run listed in their order of occurrence and including duplicates. Note in particular that internal actions do not appear in traces. The set of finite traces of a process p is defined as $\text{Fin}(p) = \{tr \in \text{traces}(p) : |tr| \in \mathbb{N}\}$. If we are not interested in the intermediate states of a run then we use the notation $p \stackrel{w}{\Longrightarrow} q$ to state that there exists a run π starting from state p and ending at state q such that $\text{trace}(\pi) = w$. We also use $p \stackrel{w}{\Longrightarrow}$ instead of $\exists p' : p \stackrel{w}{\Longrightarrow} p'$.

A process p that has no outgoing internal action cannot make any progress unless it performs a visible action. We say that such a process is *stable* [26]. We write $p\downarrow$ whenever we want to say that process p is stable. Formally, $p\downarrow = \neg(\exists p' \neq p : p \stackrel{\varepsilon}{\Longrightarrow} p')$. A stable process p responds predictably to any set of actions $X \subseteq A$, in the sense that its response depends exclusively on its outgoing transitions. Whenever there is no action $a \in X$ such that $p \xrightarrow{a}$ we say that p *refuses* the set X. Only stable processes are able to refuse actions; unstable processes refuse actions "by proxy": they refuse a set X whenever they can internally become a stable process that refuses X. Formally, p refuses X (written p ref X) if and only if $\forall a \in X : \neg(\exists p' : (p \stackrel{\varepsilon}{\Longrightarrow} p') \wedge p'\downarrow \wedge p' \xrightarrow{a})$.

To describe the behaviour of a process in terms of refusals we need to record each refusal together with the trace that causes that refusal. An observation of a refusal plus the trace that causes it is called a *stable failure* [26]. Formally, (w, X) is a stable failure of process p if and only if $\exists p^w : (p \stackrel{w}{\Longrightarrow} p^w) \wedge p^w\downarrow \wedge (p^w \text{ ref } X)$. The set of stable failures of p is then $\mathscr{SF}(p) = \{(w, X) : \exists p^w : (p \stackrel{w}{\Longrightarrow} p^w) \wedge p^w\downarrow \wedge (p^w \text{ ref } X)\}$.

Several preorder relations (that is, binary relations that are reflexive and transitive but not necessarily symmetric or antisymmetric) can be defined over processes based on their observable behaviour (including traces, refusals, stable failures, etc.) [5]. Such preorders can then be used in practice as implementation relations, which in turn create a process-oriented specification technique. The *stable failure preorder* is defined based on stable failures and is one of the finest such preorders (but not the absolute finest) [5].

Let p and q be two processes. The stable failure preorder $\sqsubseteq_{\text{SF}}$ is defined as $p \sqsubseteq_{\text{SF}} q$ if and only if $\text{Fin}(p) \subseteq \text{Fin}(q)$ and $\mathscr{SF}(p) \subseteq \mathscr{SF}(q)$. Given the preorder $\sqsubseteq_{\text{SF}}$ one can naturally define the stable failure equivalence $\simeq_{\mathscr{SF}}$: $p \simeq_{\mathscr{SF}} q$ if and only if $p \sqsubseteq_{\text{SF}} q$ and $q \sqsubseteq_{\text{SF}} p$.

2.3 Failure Trace Testing

In model-based testing [4] a test runs in parallel with the system under test and synchronizes with it over visible actions. A run of a test t and a process p represents a possible sequence of states and actions of t and p running synchronously. The outcome of such a run is either success ($\top$) or failure ($\bot$). The precise definition of synchronization, success, and failure depends on the particular type of tests being considered. We will present below such definitions for the particular framework of failure trace testing.

Given the nondeterministic nature of LTS there may be multiple runs for a given process p and a given test t and so a set of outcomes is necessary to give the results of all the possible runs. We denote by $\mathrm{Obs}(p,t)$ the set of exactly all the possible outcomes of all the runs of p and t. Given the existence of such a set of outcomes, two definitions of a process passing a test are possible. More precisely, a process p *may* pass a test t whenever some run is successful (formally, p may t if and only if $\top \in \mathrm{Obs}(p,t)$), while p *must* pass t whenever all runs are successful (formally, p must t if and only if $\{\top\} = \mathrm{Obs}(p,t)$).

In what follows we use the notation $\mathrm{init}(p) = \{a \in A : p \stackrel{a}{\Longrightarrow}\}$. A failure trace f [20] is a string of the form $f = A_0a_1A_1a_2A_2 \ldots a_nA_n$, $n \geq 0$, with $a_i \in A^*$ (sequences of actions) and $A_i \subseteq A$ (sets of refusals). Let p be a process such that $p \stackrel{\varepsilon}{\Longrightarrow} p_0 \stackrel{a_1}{\Longrightarrow} p_1 \stackrel{a_2}{\Longrightarrow} \cdots \stackrel{a_n}{\Longrightarrow} p_n$; $f = A_0a_1A_1a_2A_2 \ldots a_nA_n$ is then a failure trace of p whenever the following two conditions hold:

- If $\neg(p_i \stackrel{\tau}{\longrightarrow})$, then $A_i \subseteq (A \setminus \mathrm{init}(p_i))$; for a stable state the failure trace refuses any set of events that cannot be performed in that state (including the empty set).
- If $p_i \stackrel{\tau}{\longrightarrow}$ then $A_i = \emptyset$; whenever p_i is not a stable state it refuses an empty set of events by definition.

In other words, we obtain a failure trace of p by taking a trace of p and inserting refusal sets after stable states.

Systems and tests can be concisely described using the testing language TLOTOS [3, 20], which will also be used in this paper. A is the countable set of observable actions, ranged over by a. The set of processes or tests is ranged over by t, t_1 and t_2, while T ranges over the sets of tests or processes. The syntax of TLOTOS is then defined as follows:

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

The semantics of TLOTOS is then the following:

1. inaction (stop): no rules.
2. action prefix: $a;t_1 \stackrel{a}{\longrightarrow} t_1$ and $\mathbf{i};t_1 \stackrel{\tau}{\longrightarrow} t_1$
3. deadlock detection: $\theta;t_1 \stackrel{\theta}{\longrightarrow} t_1$.
4. successful termination: $\mathrm{pass} \stackrel{\gamma}{\longrightarrow} \mathrm{stop}$.
5. choice: with $g \in A \cup \{\gamma, \theta, \tau\}$,
$$\frac{t_1 \stackrel{g}{\longrightarrow} t_1'}{\begin{array}{c} t_1 \,\square\, t_2 \stackrel{g}{\longrightarrow} t_1' \\ t_2 \,\square\, t_1 \stackrel{g}{\longrightarrow} t_1' \end{array}}$$
6. generalized choice: with $g \in A \cup \{\gamma, \theta, \tau\}$,
$$\frac{t_1 \stackrel{g}{\longrightarrow} t_1'}{\Sigma(\{t_1\} \cup t) \stackrel{g}{\longrightarrow} t_1'}$$

Failure trace tests are defined in TLOTOS using the special actions γ which signals the successful completion of a test, and θ which is the deadlock detection label (the precise behaviour will be given later). Processes (or LTS) can also be described as TLOTOS processes, but such a description does not contain γ or θ. A test runs in parallel with the system under test according to the parallel composition operator $\|_\theta$. This operator also defines the semantics of θ as the lowest priority action:

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

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

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

Given that both processes and tests can be nondeterministic we have a set $\Pi(p\|_\theta t)$ of possible runs of a process and a test. The outcome of a particular run $\pi \in \Pi(p\|_\theta t)$ of a test t and a process under test p is success ($\top$) whenever the last symbol in trace(π) is γ, and failure ($\bot$) otherwise. One can then distinguish the possibility and the inevitability of success for a test as mentioned earlier: p may t if and only if $\top \in \text{Obs}(p,t)$, and p must t if and only if $\{\top\} = \text{Obs}(p,t)$.

The set $\mathscr{ST}$ of sequential tests is defined as follows [20]: pass $\in \mathscr{ST}$, if $t \in \mathscr{ST}$ then $a;t \in \mathscr{ST}$ for any $a \in A$, and if $t \in \mathscr{ST}$ then $\Sigma\{a;\text{stop} : a \in A'\} \,\square\, \theta;t \in \mathscr{ST}$ for any $A' \subseteq A$.

A bijection between failure traces and sequential tests exists [20]. For a sequential test t the failure trace ftr(t) is defined inductively as follows: $\text{ftr}(\text{pass}) = \emptyset$, $\text{ftr}(a;t') = a\ \text{ftr}(t')$, and $\text{ftr}(\Sigma\{a;\text{stop} : a \in A'\} \,\square\, \theta;t') = A'\ \text{ftr}(t')$. Conversely, let f be a failure trace. Then we can inductively define the sequential test st(f) as follows: $\text{st}(\emptyset) = \text{pass}$, $\text{st}(af) = a\ \text{st}(f)$, and $\text{st}(Af) = \Sigma\{a;\text{stop} : a \in A\} \,\square\, \theta;\text{st}(f)$. For all failure traces f we have that $\text{ftr}(\text{st}(f)) = f$, and for all tests t we have $\text{st}(\text{ftr}(t)) = t$. We then define the failure trace preorder $\sqsubseteq_{\text{FT}}$ as follows: $p \sqsubseteq_{\text{FT}} q$ if and only if $\text{ftr}(p) \subseteq \text{ftr}(q)$.

The above bijection effectively shows that the failure trace preorder (which is based on the behaviour of processes) can be readily converted into a testing-based preorder (based on the outcomes of tests applied to processes). Indeed there exists a successful run of p in parallel with the test t, if and only if f is a failure trace of both p and t. Furthermore, these two preorders are equivalent to the stable failure preorder introduced earlier:

Proposition 1. [20] *Let p be a process, t a sequential test, and f a failure trace. Then p* may *t if and only if $f \in \text{ftr}(p)$, where $f = \text{ftr}(t)$.*

Let p_1 and p_2 be processes. Then $p_1 \sqsubseteq_{\text{SF}} p_2$ if and only if $p_1 \sqsubseteq_{\text{FT}} p_2$ if and only if p_1 may *$t \implies p_2$* may *t for all failure trace tests t if and only if $\forall t' \in \mathscr{ST} : p_1$* may *$t' \implies p_2$* may *$t'$.*

Let t be a failure trace test. Then there exists $T(t) \subseteq \mathscr{ST}$ such that p may *t if and only if $\exists t' \in T(t)$: p* may *t'.*

We note in passing that unlike other preorders, $\sqsubseteq_{\text{SF}}$ (or equivalently $\sqsubseteq_{\text{FT}}$) can be in fact characterized in terms of may testing only; the must operator needs not be considered any further.

3 Previous Work

The investigation into connecting logical and algebraic frameworks of formal specification and verification has not been pursued in too much depth. To our knowledge the only substantial investigation on the matter is based on linear-time temporal logic (LTL) and its relation with Büchi automata [28]. Such an investigation started with timed Büchi automata [1] approaches to LTL model checking [10, 13, 17, 30, 31].

An explicit equivalence between LTL and the may and must testing framework of De Nicola and Hennessy [13] was developed as a unified semantic theory for heterogeneous system specifications featuring mixtures of labeled transition systems and LTL formulae [11]. This theory uses Büchi automata [28] rather than LTS as underlying semantic formalism. The Büchi must-preorder for a certain class of Büchi process was first established by means of trace inclusion. Then LTL formulae were converted into Büchi processes whose languages contain the traces that satisfy the formula.

The relation between may and must testing and temporal logic mentioned above [11] was also extended to the timed (or real-time) domain [6, 12]. Two refinement timed preorders similar to may and must testing were introduced, together with behavioural and language-based characterizations for these relations (to show that the new preorders are extensions of the traditional preorders). An algorithm for automated test generation out of formulae written in a timed variant of LTL called Timed Propositional Temporal Logic (TPTL) [2] was then introduced.

To our knowledge there were only two efforts on the equivalence between CTL and algebraic specifications [7, 14], one of which is our preliminary version of this paper. This earlier version [7] presents the equivalence between LTS and Kripke structures (Section 4 below) and also a tentative (but not complete) conversion from failure trace tests to CTL formulae. The other effort [14] is the basis of our second equivalence relation between LTS and Kripke structure (again see Section 4 below).

4 Two Constructive Equivalence Relations between LTS and Kripke Structures

We believe that the only meaningful basis for constructing a Kripke structure equivalent to a given LTS is by taking the outgoing actions of an LTS state as the propositions that hold on the equivalent Kripke structure state. This idea is amenable to at least two algorithmic conversion methods.

4.1 Constructing a Compact Kripke Structure Equivalent with a Given LTS

We first define an LTS satisfaction operator similar to the one on Kripke structures in a natural way (and according to the intuition presented above).

Definition 1. SATISFACTION FOR PROCESSES: *A process p satisfies $a \in A$, written by abuse of notation $p \models f$, 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 $\pi = p \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$.*

We use π^i to denote the i-th state of a path π (with the first state being state 0, or π^0). The definition of $\models$ for LTS paths is:

1. $\pi \models \mathsf{X}\, f$ iff $\pi^1 \models f$.
2. $\pi \models f \,\mathsf{U}\, g$ 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 $i < j$.
3. $\pi \models f \,\mathsf{R}\, g$ 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. Such an operator is similar to the original, but is defined over a set of states rather than a single state. By abuse of notation we denote this operator by $\models$ as well.

Definition 2. SATISFACTION OVER SETS OF STATES: *Consider a Kripke structure $K = (S, S_0, R, L)$ 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. $K, Q \models a$ *iff* $a \in L(s)$ *for some* $s \in Q$, $a \in \mathsf{AP}$.
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* $\pi = s \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_i$ *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* $\pi = s \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_i$.

With these definition we can introduce the following equivalence relation between Kripke structures and LTS.

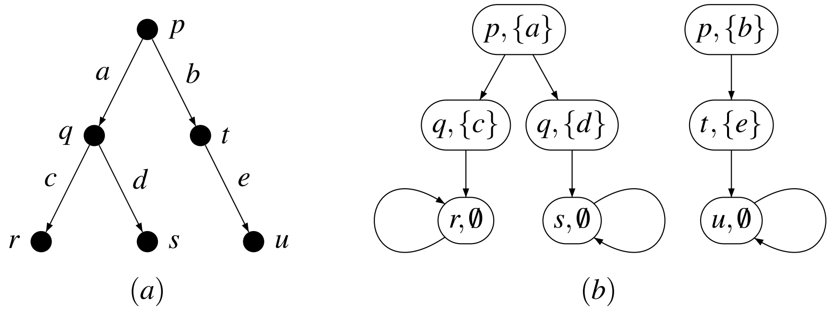
Definition 3. EQUIVALENCE BETWEEN KRIPKE STRUCTURES AND LTS: *Given a Kripke structure K and a set of states Q of K, the pair K, Q is equivalent to a process p, written $K, Q \simeq p$ (or $p \simeq K, Q$), if and only if for any CTL* formula f $K, Q \models f$ if and only if $p \models f$.*

It is easy to see that the relation $\simeq$ is indeed an equivalence relation. This equivalence has the useful property that given an LTS it is easy to construct its equivalent Kripke structure.

Theorem 2. *There exists an algorithmic function $\mathbb{K}$ which converts a labeled transition system p into a Kripke structure K and a set of states Q such that $p \simeq (K, Q)$.*

Specifically, for any labeled transition system $p = (S, A, \rightarrow, s_0)$, its equivalent Kripke structure $K = \mathbb{K}(p)$ is defined as $K = (S', Q, R', L')$ where:

1. $S' = \{\langle s, x \rangle : s \in S, x \subseteq \mathrm{init}(s)\}$.
2. $Q = \{\langle s_0, x \rangle \in S'\}$.
3. R' *contains exactly all the transitions* $(\langle s, N \rangle, \langle t, O \rangle)$ *such that* $\langle s, N \rangle, \langle t, O \rangle \in S'$, *and*
   (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).*



Each state of the Kripke structure (b) is labeled with the LTS state it came from, and the set of propositions that hold in that Kripke state.

Figure 1: A conversion of an LTS (a) to an equivalent Kripke structure (b).

4. $L' : S' \to 2^{\mathsf{AP}}$ *such that* $L'(s,x) = x$, *where* $\mathsf{AP} = A$.

Figure 1 illustrates graphically how we convert a labelled transition system to its equivalent Kripke structure. As illustrated in the figure, we combine each state in the labelled transition system with its actions provided as properties to form new states in the equivalent Kripke structure. The transition relation of the Kripke structure is formed by the new states and the corresponding transition relation in the original labelled transition system. The labeling function in the equivalent Kripke structure links the actions to their relevant states. An LTS states in split into multiple Kripke states whenever it can evolve differently by performing different actions (like state p in the figure).

Using such a conversion we can define the semantics of CTL* formulae with respect to a process rather than Kripke structure. One problem—that required the new satisfaction operator for sets of Kripke states as defined in Definition 2—is introduced by the fact that one state of a process can generate multiple initial Kripke states. We believe that the weaker satisfaction operator from Definition 2 is introduced without loss of generality and may even be worked around by such mechanisms as considering processes with one outgoing transition (a “start” action) followed by their normal behaviour.

Proof of Theorem 2. The proof relies on the properties of the syntax and semantics of CTL* formulae and is done by structural induction.

For the basis of the induction, we note that $\top$ is true for any process and for any state in any Kripke structure. $p \vDash \top$ iff $K, Q \vDash \top$ is therefore immediate. The same goes for $\bot$ (no process and no state in any Kripke structure satisfy $\bot$). $p \vDash a$ iff $\mathbb{K}(p) = K, Q \vDash a$ by the definition of $\mathbb{K}$; indeed, $a \in \text{init}(p)$ (so that $p \vDash a$) iff $a \in x$ for some state $\langle s, x\rangle \in Q$ that is, $a \in L'(\langle s, x\rangle)$.

On to the inductive step. $p \vDash f'$ iff $\mathbb{K}(p) \vDash f'$ for any formula f' by induction hypothesis, so we take $f' = \neg f$ and so $p \vDash \neg f$ iff $\mathbb{K}(p) \vDash \neg f$.

Suppose that $p \vDash f$ and [or] $p \vDash g$ (so that $p \vDash f \wedge g$ [$p \vDash f \vee g$]). This is equivalent by induction hypothesis to $\mathbb{K}(p) \vDash f$ and [or] $\mathbb{K}(p) \vDash g$, that is, $\mathbb{K}(p) \vDash f \wedge g$ [$\mathbb{K}(p) \vDash f \vee g$], as desired.

Let now π be a path $\pi = p \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ starting from a process p. According to the definition of $\mathbb{K}$, all the equivalent paths in the Kripke structure $\mathbb{K}(p)$ have the form $\pi' = \langle p, A_0\rangle \to \langle s_1, A_1\rangle \to \langle s_2, A_2\rangle \to \cdots \to \langle s_n, A_n\rangle$, such that $a_i \in A_i$ for all $0 \leq i < n$. Clearly, such a path π' exists. Moreover, given some path of form π', a path of form π also exists (because no path in the Kripke structure comes out of the blue; instead all of them come from paths in the original process). By abuse of notation we write $\mathbb{K}(\pi) = \pi'$, with the understanding that this incarnation of $\mathbb{K}$ is not necessarily a function (it could be a relation) but is complete (there exists a path $\mathbb{K}(\pi)$ for any path π). With this notion of equivalent paths we can now proceed to path formulae.

Consider the formula $\mathsf{X}f$ such that some path π satisfies it. Whenever $\pi \models \mathsf{X}f$, $\pi^1 \models f$ and therefore $\mathbb{K}(\pi)^1 \models f$ (by inductive assumption, for indeed f is a state, not a path formula) and therefore $\mathbb{K}(\pi) \models \mathsf{X}f$, as desired. Conversely, $\mathbb{K}(\pi) \models \mathsf{X}f$, that is, $\mathbb{K}(\pi)^1 \models f$ means that $\pi^1 \models f$ by inductive assumption, and so $\pi \models \mathsf{X}f$.

The proof for F, G, U, and R operators proceed similarly. Whenever $\pi \models \mathsf{F}\ f$, there is a state π^i such that $\pi^i \models f$. By induction hypothesis then $\mathbb{K}(\pi)^i \models f$ and so $\mathbb{K}(\pi) \models \mathsf{F}\ f$. The other way (from $\mathbb{K}(\pi)$ to π) is similar. The G operator requires that all the states along π satisfy f, which imply that all the states in any $\mathbb{K}(\pi)$ satisfy f, and thus $\mathbb{K}(\pi) \models \mathsf{G}\ f$ (and again things proceed similarly in the other direction). In all, the induction hypothesis established a bijection between the states in π and the states in (any) $\mathbb{K}(\pi)$. This bijection is used in the proof for U and R just as it was used in the above proof for F and G. Indeed, the states along the path π will satisfy f or g as appropriate for the respective operator, but this translates in the same set of states satisfying f and g in $\mathbb{K}(\pi)$, so the whole formula (using U or R holds in π iff it holds in $\mathbb{K}(\pi)$.

Finally, given a formula $\mathsf{E}\ f$, $p \models \mathsf{E}\ f$ implies that there exists a path π starting from p that satisfies f. By induction hypothesis there is then a path $\mathbb{K}(\pi)$ starting from $\mathbb{K}(p)$ that satisfies f (there is at least one such a path) and thus $\mathbb{K}(p) \models \mathsf{E}\ f$. The other way around is similar, and so is the proof for $\mathsf{A}\ f$ (all the paths π satisfy f so all the path $\mathbb{K}(\pi)$ satisfy f as well; there are no supplementary paths, since all the paths in $\mathbb{K}(p)$ come from the paths in p). □

4.2 Yet Another Constructive Equivalence between LTS and Kripke Structures

The function $\mathbb{K}$ developed earlier produces a very compact Kripke structure. However, a state in the original LTS can result in multiple equivalent state in the resulting Kripke structure, which in turn requires a modified notion of satisfaction (over sets of states, see Definition 2). This in turn implies a non-standard model checking algorithm. A different such a conversion algorithm [14] avoids this issue, at the expense of a considerably larger Kripke structure. We now explore a similar equivalence.

The just mentioned conversion algorithm [14] is based on introducing intermediate states in the resulting Kripke structure. These states are labelled with the special proposition Δ which is understood to mark a state that is ignored in the process of determining the truth value of a CTL formula; if Δ labels a state then it is the only label for that state. We therefore base our construction on the following definition of equivalence between processes and Kripke structures:

Definition 4. SATISFACTION FOR PROCESSES: *Given a Kripke structure K and a state s of K, the pair K,s is equivalent to a process p, written as $K,s \simeq p$ (or $p \simeq K,s$) iff for any CTL* formula f $K,s \models f$ iff $p \models f$. The operator $\models$ is defined for processes in Definition 1 and for Kripke structures as follows:*

1. $p \models \top$ *iff* $K,s \models \top$
2. $p \models a$ *iff* $K,s \models \Delta\ \mathsf{U}\ a$
3. $p \models \neg f$ *iff* $K,s \models \neg f$
4. $p \models f \wedge g$ *iff* $K,s \models f \wedge g$
5. $p \models f \vee g$ *iff* $K,s \models f \vee g$
6. $p \models \mathsf{E}f$ *iff* $K,s \models \mathsf{E}f$
7. $p \models f\ \mathsf{U}\ g$ *iff* $K,s \models (\Delta \vee f)\ \mathsf{U}\ g$

8. $p \models \mathsf{X}\, f$ *iff* $K, s \models \mathsf{X}\, (\Delta\ \mathsf{U}\ f)$

9. $p \models \mathsf{F} f$ *iff* $K, s \models \mathsf{F}\ f$

10. $p \models \mathsf{G}\ f$ *iff* $K, s \models \mathsf{G}\ (\Delta \vee f)$

11. $p \models f\ \mathsf{R}\ g$ *iff* $K, s \models f\ \mathsf{R}\ (\Delta \vee g)$

Note that the definition above is stated in terms of CTL* rather than CTL; however, CTL* is stronger and so equivalence under CTL* implies equivalence under CTL.

Most of the equivalence is immediate. However, some cases need to make sure that the states labelled Δ are ignored. This happens first in $K, s \models \Delta\ \mathsf{U}\ a$, which is equivalent to $p \models a$. Indeed, a needs to hold immediately, except that any preceding states labelled Δ must be ignored, hence a must be eventually true and when it becomes so it releases the chain of Δ labels. The formula for X is constructed using the same idea (except that the formula f releasing the possible chain of Δ happens starting from the next state).

Then expression $(\Delta \vee f)\ \mathsf{U}\ g$ means that f must remain true with possible interleaves of Δ until g becomes true. Similarly $f\ \mathsf{R}\ (\Delta \vee g)$ requires that g is true (with the usual interleaved Δ) until it is released by f becoming true.

Based on this equivalence we can define a new conversion of LTS into equivalent Kripke structures. This conversion is again based on a similar conversion [14] developed in a different context.

Theorem 3. *There exist at least two algorithmic functions for converting LTS into equivalent Kripke structures. The first is the function $\mathbb{K}$ described in Theorem 2.*

The new function $\mathbb{X}$ is defined as follows: with Δ a fresh symbol not in A: Given an LTS $p = (S, A, \rightarrow, s_0)$, the Kripke structure $\mathbb{X}(p) = (S', Q, R', L)$ is given by:

1. $\mathsf{AP} = A \uplus \Delta$;

2. $S' = S \cup \{(r, a, s) : a \in A \text{ and } r \xrightarrow{a} s\}$;

3. $Q = \{s_0\}$;

4. $R' = \{(r, s) : r \xrightarrow{\tau} s\} \cup \{(r, (r, a, s)) : r \xrightarrow{a} s\} \cup \{((r, a, s), s) : r \xrightarrow{a} s\}$;

5. *For* $r, s \in S$ *and* $a \in A : L(s) = \{\Delta\}$ *and* $L((r, a, s)) = \{a\}$.

Then $p \simeq \mathbb{X}(p)$.

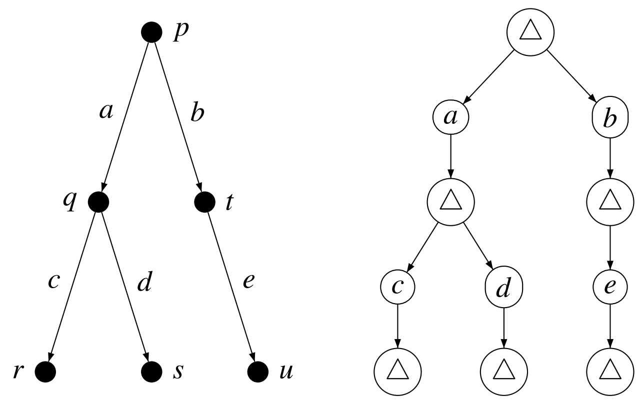
Proof. We prove the stronger equivalence over CTL* rather than CTL by structural induction. Since Δ is effectively handled by the satisfaction operator introduced in Definition 4 it will turn out that there is no need to mention it at all.

For the basis of the induction, we note that $\top$ is true for any process and for any state in any Kripke structure. $p \models \top$ iff $\mathbb{X}(p) \models \top$ is therefore immediate. The same goes for $\bot$ (no process and no state in any Kripke structure satisfy $\bot$). $p \models a$ iff $\mathbb{X}(p) \models a$; Indeed, $a \in A$(so that $p \models a$) iff $a \in L((r, a, s))$.

That $p \models \neg f$ iff $\mathbb{X}(p) \models \neg f$ is immediately given by the induction hypothesis that $p \models f$ iff $\mathbb{X}(p) \models f$.

Suppose that $p \models f$ and [or] $p \models g$ (so that $p \models f \wedge g$ [$p \models f \vee g$]). This is equivalent by induction hypothesis to $\mathbb{X}(p) \models f$ and [or] $\mathbb{X}(p) \models g$, that is, $\mathbb{X}(p) \models f \wedge g$ [$\mathbb{X}(p) \models f \vee g$], as desired.

Let now π' be a path $\pi' = p \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ starting from a process p. According to the definition of $\mathbb{X}$, all the equivalent paths in the Kripke structure $\mathbb{X}(p)$ have the form $\pi' = \Delta \rightarrow A_0 \rightarrow \Delta \rightarrow A_1 \rightarrow \Delta \rightarrow A_2 \rightarrow \cdots \Delta \rightarrow A_n$, such that $a_i \in A_i$ for all $0 \le i < n$. Clearly, such a path π' exists. According



(a) (b)

Figure 2: Conversion of an LTS (a) to its equivalent Kripke structure (b).

to the function $\mathbb{X}$, we know that Δ is a symbol that stands for states in the LTS and has no meaning in the Kripke structure. The satisfaction operator for Kripke structures (Definition 4) is specifically designed to ignore the Δ label and this insures that the part π' is equivalent to the path $\pi = A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n$ with $a_i \in A_i$ for all $0 \leq i < n$ and so we will use this form for the reminder of the proof.

Consider the formula $\mathsf{X}f$ such that some path π satisfies it. Whenever $\pi \models \mathsf{X}f$, $\pi^1 \models f$ and therefore $\mathbb{X}(\pi)^1 \models f$ (by inductive assumption, for indeed f is a state, not a path formula) and therefore $\mathbb{X}(\pi) \models \mathsf{X}f$, as desired. Conversely, $\mathbb{X}(\pi) \models \mathsf{X}f$, that is, $\mathbb{X}(\pi)^1 \models f$ means that $\pi^1 \models f$ by inductive assumption, and so $\pi \models \mathsf{X}f$.

The proof for F, G, U, and R operators proceed similarly. Whenever $\pi \models \mathsf{F}\ f$, there is a state π^i such that $\pi^i \models f$. By induction hypothesis then $\mathbb{X}(\pi)^i \models f$ and so $\mathbb{X}(\pi) \models \mathsf{F}\ f$. The other way (from $\mathbb{X}(\pi)$ to π) is similar. The G operator requires that all the states along π satisfy f, which implies that all the states in any $\mathbb{X}(\pi)$ satisfy f, and thus $\mathbb{X}(\pi) \models \mathsf{G}\ f$ (and again things proceed similarly in the other direction). In all, the induction hypothesis established a bijection between the states in π and the states in (any) $\mathbb{X}(\pi)$. This bijection is used in the proof for U and R just as it was used in the above proof for F and G. Indeed, the states along the path π will satisfy f or g as appropriate for the respective operator, but this translates in the same set of states satisfying f and g in $\mathbb{X}(\pi)$, so the whole formula (using U or R holds in π iff it holds in $\mathbb{X}(\pi)$).

Finally, given a formula $\mathsf{E}\ f$, $p \models \mathsf{E}\ f$ implies that there exists a path π starting from p that satisfies f. By induction hypothesis there is then a path $\mathbb{X}(\pi)$ starting from $\mathbb{X}(p)$ that satisfies f (there is at least one such a path) and thus $\mathbb{X}(p) \models \mathsf{E}\ f$. The other way around is similar, and so is the proof for $\mathsf{A}\ f$ (all the paths π satisfy f so all the path $\mathbb{X}(\pi)$ satisfy f as well; there are no supplementary paths, since all the paths in $\mathbb{X}(p)$ come from the paths in p). □

The process of the new version described in Theorem 3 is most easily described graphically; refer for this purpose to Figure 2. Specifically, the function $\mathbb{X}$ converts the LTS given in Figure 2(a) into the equivalent Kripke structure shown in Figure 2(b). In this new structure, instead of combining each state with its corresponding actions in the LTS (and thus possibly splitting the LTS state into multiple Kripke structure states), we use the new symbol Δ to stand for the original LTS states. Every Δ state of the Kripke structure is the LTS state, and all the other states in the Kripke structure are the actions in the LTS. This ensures that all states in the Kripke structure corresponding to actions that are outgoing from a single LTS state have all the same parent. This in turn eliminates the need for the weaker satisfaction operator over sets of states (Definition 2).

5 CTL Is Equivalent to Failure Trace Testing

We now proceed to show the equivalence between CTL formulae and failure trace tests. Let $\mathscr{P}$ be the set of all processes, $\mathscr{T}$ the set of all failure trace tests, and $\mathscr{F}$ the set of all CTL formulae. We have:

Theorem 4. 1. *For some $t \in \mathscr{T}$ and $f \in \mathscr{F}$, whenever p* may *t if and only if $\mathbb{K}(p) \vDash f$ for any $p \in \mathscr{P}$ we say that t and f are equivalent. Then, for every failure trace test there exists an equivalent CTL formula and the other way around. Furthermore a failure trace test can be algorithmically converted into its equivalent CTL formula and the other way around.*

2. *For some $t \in \mathscr{T}$ and $f \in \mathscr{F}$, whenever p* may *t if and only if $\mathbb{X}(p) \vDash f$ for any $p \in \mathscr{P}$ we say that t and f are equivalent. Then, for every failure trace test there exists an equivalent CTL formula and the other way around. Furthermore a failure trace test can be algorithmically converted into its equivalent CTL formula and the other way around.*

Proof. The proof of Item 1 follows from Lemma 5 (in Section 5.1 below) and Lemma 11 (in Section 5.3 below). The algorithmic nature of the conversion is shown implicitly in the proofs of these two results. The proof of Item 2 is fairly similar and is summarized in Lemma 6 (Section 5.1) and Lemma 12 (Section 5.3). □

The remainder of this section is dedicated to the proof of the lemmata mentioned above and so the actual proof of this result. Note incidentally that Lemmata 11 and 12 will be further improved in Theorem 7.

5.1 From Failure Trace Tests to CTL Formulae

Lemma 5. *There exists a function $\mathbb{F}_{\mathbb{K}} : \mathscr{T} \to \mathscr{F}$ such that p* may *t if and only if $\mathbb{K}(p) \vDash \mathbb{F}_{\mathbb{K}}(t)$ for any $p \in \mathscr{P}$.*

Proof. The proof is done by structural induction over tests. In the process we also construct (inductively) the function $\mathbb{F}_{\mathbb{K}}$.

We put $\mathbb{F}_{\mathbb{K}}(\text{pass}) = \top$. Any process passes pass and any Kripke structure satisfies $\top$, thus it is immediate that p may pass iff $\mathbb{K}(p) \vDash \top = \mathbb{F}_{\mathbb{K}}(\text{pass})$. Similarly, we put $\mathbb{F}_{\mathbb{K}}(\text{stop}) = \bot$. No process passes stop and no Kripke structure satisfies $\bot$.

On to the induction steps now. We put $\mathbb{F}_{\mathbb{K}}(\mathbf{i};t) = \mathbb{F}_{\mathbb{K}}(t)$: an internal action in a test is not seen by the process under test by definition. We then put $\mathbb{F}_{\mathbb{K}}(a;t) = a \wedge \mathsf{EX}\ \mathbb{F}_{\mathbb{K}}(t)$. We note that p may $(a;t)$ iff p may a and p' may t for some $p \xrightarrow{a} p'$. Now, p may a iff $\mathbb{K}(p) \vDash a$ by the construction of $\mathbb{K}$, and also p' may t iff $\mathbb{K}(p') \vDash \mathbb{F}_{\mathbb{K}}(t)$ by induction hypothesis. By Theorem 2, when we convert p to an equivalent Kripke structure $\mathbb{K}(p)$ we take as new states the original states together with their outgoing actions. So once we are (in $\mathbb{K}(p)$) in a state that satisfies a, all the next states of that state correspond to the states following p after executing a. Therefore, $\mathsf{X}(\mathbb{F}_{\mathbb{K}}(t))$ is satisfied in exactly those states in which t must succeed. Thus p may $a;t$ iff $\mathbb{K}(p) \vDash a \wedge \mathsf{EX}\mathbb{F}_{\mathbb{K}}(t)$. For illustration purposes note that in Figure 1 the initial state p becomes two initial states $(p,\{a\})$ and $(p,\{b\})$; the next state of the state satisfying the property a in the Kripke structure contains only q (and never t).

Note now that □ is just syntactical sugar, for indeed $t_1 \,\square\, t_2$ is perfectly equivalent with $\Sigma\{t_1,t_2\}$. We put[1] $\mathbb{F}_{\mathbb{K}}(\Sigma T) = \bigvee\{\mathbb{F}_{\mathbb{K}}(t) : t \in T\}$. p may ΣT iff p may t for at least one $t \in T$ iff $\mathbb{K}(p) \vDash \mathbb{F}_{\mathbb{K}}(t)$ for at least one $t \in T$ (by induction hypothesis) iff $\mathbb{K}(p) \vDash \bigvee\{\mathbb{F}_{\mathbb{K}}(t) : t \in T\}$.

[1] As usual $\bigvee\{t_1,\ldots,t_n\}$ is a shorthand for $t_1 \vee \cdots \vee t_n$.

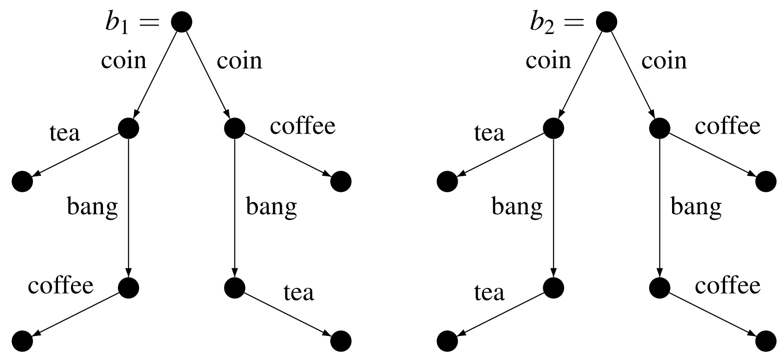


Figure 3: Two coffee machines.

We finally get to consider θ. Note first that whenever θ does not participate in a choice it behaves exactly like **i**, so we assume without loss of generality that θ appears only in choice constructs. We also assume without loss of generality that every choice contains at most one top-level θ, for indeed $\theta; t_1 \,\Box\, \theta; t_2$ is equivalent with $\theta; (t_1 \,\Box\, t_2)$. We put $\mathbb{F}_{\mathbb{K}}(t_1 \,\Box\, \theta; t) = ((\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t_1)) \vee (\neg(\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t))$.

According to the TLOTOS definition of $\|_\theta$ (see Section 2.3), if a common action is available for both p and t then the deadlock detection action θ will not play any role. In other words, whenever $p \stackrel{a}{\Longrightarrow}$ such that $a \in \mathrm{init}(t_1)$ we have p may $t_1 \,\Box\, \theta; t$ iff p may t_1. We further note that $p \stackrel{a}{\Longrightarrow}$ is equivalent to $\mathbb{K}(p) \vDash a$ and so given the inductive hypothesis (that p' may t_1 iff $\mathbb{K}(p') \vDash \mathbb{F}_{\mathbb{K}}(t_1)$ for any process p') we conclude that:

$$(p \text{ may } t_1 \,\Box\, \theta; t) \;\wedge\; (p \stackrel{a}{\Longrightarrow} \wedge a \in \mathrm{init}(t_1)) \quad \text{iff} \quad \mathbb{K}(p) \vDash (\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t_1) \tag{1}$$

Whenever it is not the case that $p \stackrel{a}{\Longrightarrow}$ and $a \in \mathrm{init}(t_1)$ (equivalent to $\mathbb{K}(p) \nvDash \bigvee \mathrm{init}(t_1)$), then the deadlock detection transition θ of $t_1 \,\Box\, \theta; t$ will fire and then the test will succeed iff t succeeds. Given once more the inductive hypothesis that p' may t iff $\mathbb{K}(p') \vDash \mathbb{F}_{\mathbb{K}}(t)$ for any process p' we have:

$$(p \text{ may } t_1 \,\Box\, \theta; t) \;\wedge\; \neg(p \stackrel{a}{\Longrightarrow} \wedge a \in \mathrm{init}(t_1)) \quad \text{iff} \quad \mathbb{K}(p) \vDash \neg(\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t) \tag{2}$$

The correctness of $\mathbb{F}_{\mathbb{K}}(t_1 \,\Box\, \theta; t)$ is then a direct consequence of Relations (1) and (2); indeed, one just has to take the disjunction of both sides of these relations to reach the desired equivalence:

$$p \text{ may } t_1 \,\Box\, \theta; t \quad \text{iff} \quad \mathbb{K}(p) \vDash (\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t_1) \;\vee\; \mathbb{K}(p) \vDash \neg(\bigvee \mathrm{init}(t_1)) \wedge \mathbb{F}_{\mathbb{K}}(t)$$

The induction is thus complete. ☐

Example 1. HOW TO TELL LOGICALLY THAT YOUR COFFEE MACHINE IS WORKING.: The coffee machines b_1 and b_2 below were famously introduced to illustrate the limitations of the may and must testing framework of De Nicola and Hennessy. Indeed, they have been found [20] to be equivalent under testing preorder [13] but not equivalent under stable failure preorder [20].

$$\begin{aligned} b_1 &= \mathrm{coin}; (\mathrm{tea} \,\Box\, \mathrm{bang}; \mathrm{coffee}) \quad \Box \quad \mathrm{coin}; (\mathrm{coffee} \,\Box\, \mathrm{bang}; \mathrm{tea}) \\ b_2 &= \mathrm{coin}; (\mathrm{tea} \,\Box\, \mathrm{bang}; \mathrm{tea}) \quad \Box \quad \mathrm{coin}; (\mathrm{coffee} \,\Box\, \mathrm{bang}; \mathrm{coffee}) \end{aligned}$$

These machines are also shown graphically (as LTS) in Figure 3.

The first machine accepts a coin and then dispenses either tea or coffee, at its discretion. Still, if one wants the other beverage, one just hits the machine. The second machine is rather stubborn, giving either tea or coffee at its discretion. By contrast with the first machines, the beverage offered will not be changed by hits. One failure trace test that differentiate these machines [20] is

$$t = \text{coin}; (\text{coffee}; \text{pass} \,\square\, \theta; \text{bang}; \text{coffee}; \text{pass})$$

The conversion of the failure trace test t into a CTL formula as per Lemma 5 yields $\mathbb{F}_{\mathbb{K}}(t) = \text{coin} \wedge \mathsf{EX}\ (\text{coffee} \wedge \mathsf{EX}\ \top \vee \neg(\text{coffee} \wedge \mathsf{EX}\ \top) \wedge \text{bang} \wedge \mathsf{EX}\ (\text{coffee} \wedge \mathsf{EX}\top))$. Eliminating the obviously true sub-formulae, we obtain the logically equivalent formula

$$\mathbb{F}_{\mathbb{K}}(t) = \text{coin} \wedge \mathsf{EX}\ (\text{coffee} \vee \neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee})$$

The meaning of this formula is clearly equivalent to the meaning of t, as it reads quite literally "a coin is expected, and in the next state either coffee is offered, or coffee is not offered but a bang is available and then the next state will offer coffee." This is rather dramatically identical to how one would describe the behaviour of t in plain English.

The formula $\mathbb{F}_{\mathbb{K}}(t)$ holds for both the initial states of $\mathbb{K}(b_1)$ (where coffee is offered from the outset or follows a hit on the machine) but holds in only one of the initial states of $\mathbb{K}(b_2)$ (the one that dispenses coffee).

Lemma 6. *There exists a function* $\mathbb{F}_{\mathbb{X}} : \mathscr{T} \to \mathscr{F}$ *such that* p may t *if and only if* $\mathbb{X}(p) \models \mathbb{F}_{\mathbb{X}}(t)$ *for any* $p \in \mathscr{P}$.

Proof. The proof for $\mathbb{F}_{\mathbb{K}}$ (Lemma 5) holds in almost all the cases. Indeed, the way the operator $\models$ is defined (Definition 4) ensures that all occurrences of Δ are "skipped over" as if they were not there in the first place. However the way LTS states are split by $\mathbb{K}$ facilitates the proof or Lemma 5, yet such a split no longer happens in $\mathbb{X}$. The original proof had $\mathbb{F}_{\mathbb{K}}(a;t) = a \wedge \mathsf{EX}\ \mathbb{F}_{\mathbb{K}}(t)$ but under $\mathbb{X}$ this construction will fail to work correctly on LTS such as $p = a \,\square\, b; p'$ such that p' may t. Indeed, $\mathbb{X}(p)$ features a node labeled Δ with two children; the first child is labeled a while the second child is not but has $\mathbb{X}(p')$ as an eventual descendant (through a possible chain of nodes labeled Δ). Clearly it is not the case that p may $a;t$, yet $\mathbb{X}(p) \models a \wedge \mathsf{EX}\ \mathbb{F}_{\mathbb{X}}(t)$, which shows that such a simple construction is not sufficient for $\mathbb{X}$.

To remedy this we set $\mathbb{F}_{\mathbb{X}}(a;t) = \mathsf{E}\ (a\ \wedge\ \mathsf{AX}\ \neg a)\ \mathsf{U}\ \mathbb{F}_{\mathbb{X}}(t) \wedge \mathsf{EX}\ \mathbb{F}_{\mathbb{X}}(t)$. The second term in the conjunction ensures that $\mathbb{F}_{\mathbb{X}}(t)$ will hold in some next state, while the first term specifies that a run of a will be followed by $\mathbb{F}_{\mathbb{X}}(t)$ (the $a\ \mathsf{U}\ \mathbb{F}_{\mathbb{X}}(t)$ component) and also that the run of a is exactly one state long (the $\mathsf{AX}\ \neg a$ part). Note in passing that the U operator is necessary in order to make sure that a and $\mathbb{F}_{\mathbb{X}}(t)$ are on the same path, for otherwise the example used above to show that the original $\mathbb{K}(p)$ does not work here will continue to be in effect.

The rest of the proof remains unchanged from the proof of Lemma 5. □

Example 2. HOW TO TELL LOGICALLY THAT YOUR COFFEE MACHINE IS WORKING, TAKE 2:
Consider again the coffee machines b_1 and b_2 from Example 1. As discussed earlier, one failure trace test that differentiate these machines is

$$t = \text{coin}; (\text{coffee}; \text{pass} \,\square\, \theta; \text{bang}; \text{coffee}; \text{pass})$$

The conversion of the failure trace test t in to a CTL formula (after eliminating all the trivially true sub-formulae) will be:

$$\begin{aligned}\mathbb{F}_{\mathbb{X}}(t) \quad = \quad & \mathsf{E}(\mathsf{coin} \wedge \mathsf{AX}\neg\mathsf{coin}\ \mathsf{U}\ \mathsf{coffee}) \wedge \mathsf{EX}(\mathsf{coffee}) \vee \\ & \mathsf{E}(\mathsf{coin} \wedge \mathsf{AX}\neg\mathsf{coin}\ \mathsf{U}\ \mathsf{bang}) \wedge \\ & \mathsf{EX}((\mathsf{bang} \wedge \mathsf{AX}\neg\mathsf{bang}\ \mathsf{U}\ \mathsf{coffee}) \wedge \mathsf{EX}(\mathsf{coffee}))\end{aligned}$$

This formula specifies that after a coin we can get a coffee immediately or after a bang we get a coffee immediately, The meaning of this formula is clearly equivalent to the meaning of t. As expected the formula $\mathbb{F}_{\mathbb{X}}(t)$ holds for b_1 but not for b_2.

5.2 Converting Failure Trace Tests into Compact CTL Formulae

Whenever all the runs of a test are finite then the conversion shown in Lemma 5 will produce a reasonable CTL formula. That formula is however not in its simplest form. In particular, the conversion algorithm follows the run of the test step by step, so whenever the test has one or more cycles (and thus features potentially infinite runs) the resulting formula has an infinite length. We now show that more compact formulae can be obtained and in particular finite formulae can be derived out of tests with potentially infinite runs. This extension works for both $\mathbb{F}_{\mathbb{K}}$ and $\mathbb{F}_{\mathbb{X}}$.

Theorem 7. *Let $\mathbb{F} \in \{\mathbb{F}_{\mathbb{K}}, \mathbb{F}_{\mathbb{X}}\}$. Then there exists an extension of $\mathbb{F}$ (denoted by abuse of notation $\mathbb{F}$ as well) such that $\mathbb{F} : \mathscr{T} \to \mathscr{F}$, p* may *t if and only if $\mathbb{K}(p) \models \mathbb{F}(t)$ for any $p \in \mathscr{P}$, and $\mathbb{F}(t)$ is finite for any test t provided that we are allowed to mark some entry action a so that we can refer to it as either a or* start(a) *in each loop of t. An entry action for a loop is defined as an action labeling an outgoing edge from a state that has an incoming edge from outside the loop.*

Proof. It is enough to show how to produce a finite formula starting from a general "loop" test. Such a conversion can be then applied to all the loops one by one, relying on the original conversion function from Lemma 5 or Lemma 6 (depending on whether $\mathbb{F}$ extends $\mathbb{F}_{\mathbb{K}}$ or $\mathbb{F}_{\mathbb{X}}$) for the rest of the test. Given the reliance on the mentioned lemmata we obtain overall an inductive construction. Therefore nested loops in particular will be converted inductively (that is, from the innermost loop to the outermost loop).

Thus to complete the proof it is enough to show how to obtain an equivalent, finite CTL formula for the following, general form of a loop test:

$$t = a_0; (t_0 \,\square\, a_1; (t_1 \,\square\, \cdots a_{n-1}; (t_{n-1} \,\square\, t) \cdots))$$

The loop itself consists of the actions $a_0, \ldots, a_{n-1}$. Each such an action a_i has the "exit" test t_i as an alternative. We make no assumption about the particular form of t_i, $0 \leq i < n$.

Given the intended use of our function, this proof will be done within the inductive assumptions of the proof of Lemmata 5 and 6. We will therefore consider that the formulae $\mathbb{F}(a_i)$ and $\mathbb{F}(t_i)$ exist and are finite, $0 \leq i < n$.

We have:

$$\mathbb{F}(t) = \left(\mathsf{E}\left(\bigvee_{i=0}^{n-1} C_i\right)\ \mathsf{U}\ \left(\bigvee_{i=0}^{n-1} E_i\right)\right) \vee \mathbb{F}(t_0)$$

where C_i represents the cycle in its various stages such that

$$C_i = \mathsf{EG}(\mathbb{F}(a_i) \wedge \mathsf{EX}(\mathbb{F}(a_{(i+1) \bmod n}) \wedge \mathsf{EX} \cdots \wedge \mathsf{EX}(\mathbb{F}(a_{(i+n-1) \bmod n})) \cdots))$$

and each E_i represents one possible exit from the cycle and so

$$E_i = \text{count}(a_i) \land \mathsf{EX}\ \mathbb{F}(t_i)$$

with

$$\text{count}(a_i) = \mathsf{EG}\ \text{start}(a_{j_0}) \land \mathsf{EX}\ (a_{j_1} \land \cdots \mathsf{EX}\ a_{j_{i-1}})$$

where the sequence $(j_0, j_1, \ldots, j_{i-1})$ is the subsequence of $(0, 1, \ldots, i-1)$ that contains exactly all the indices p such that $a_p \neq \tau$. It is worth noting that the second term in the disjunction ($\mathbb{F}(t_0)$) accounts for the possibility that while running t we exit immediately upon entering the cycle through the exit test t_0, in effect without traversing any portion of the loop.

Intuitively, each C_i corresponds to a_i as being available in the test loop, followed by all the rest of the loop in the correct order. It therefore models the decision of the test to perform a_i and remain in the loop. Whenever some a_i is available ("true") then the corresponding C_i is true and so the disjunction of the formulae C_i will keep being true as long as we stay in the loop. By contrast, each E_i corresponds to being in the right place for the test t_i to be available (and so $\text{count}(a_i)$ being true), combined with the exit from the loop using the test t_i. The formula E_i will become true whenever the test is in the right place and the test t_i succeeds. Such an event releases the loop formula from its obligations (following the semantics of the U operator), so such a path can be taken by the test and will be successful. Like the name implies, the function of $\text{count}(a_i)$ is like a counter for how many actions separate the test t_i from the marked action a_0. By counting actions we know what test is available to exit from the loop (depending on how many actions away we are from the start of the loop).

The formula above assumes that neither the actions in the cycle nor the top-level actions of the exit tests are θ. We introduce the deadlock detection action along the following cases, with k an arbitrary value, $0 \leq k < n$: θ may appear in the loop as a_k but not on top level of the alternate exit test $t_{k-1 \bmod n}$ (Case 1), on the top level of the test $t_{k-1 \bmod n}$ but not as alternate a_k (Case 2), or both as a_k and on the top level of the alternate $t_{k-1 \bmod n}$ (Case 3). Given that θ only affects the top level of the choice in which it participates, these cases are exhaustive.

1. If any $a_k = \theta$ and $\theta \notin \text{init}(t_{k-1 \bmod n})$ then we replace all occurrences of $\mathbb{F}(a_k)$ in $\mathbb{F}(t)$ with $\neg(\bigvee_{b \in \text{init}(t_{k-1 \bmod n})} \mathbb{F}(b))$ in conjunction with $\bigvee_{b \in \text{init}(t_k) \setminus \{\theta\}} \mathbb{F}(b)$ for the "exit" formulae and with $\mathbb{F}(a_{k+1 \bmod n})$ for the "cycle" formulae). Therefore $C_i = \mathsf{EG}\ (\mathbb{F}(a_i) \land \mathsf{EX}\ (\cdots \land \mathsf{EX}\ (\neg(\bigvee_{b \in \text{init}(t_{k-1 \bmod n})} \mathbb{F}(b)) \land \mathbb{F}(a_{k+1 \bmod n}) \land \mathsf{EX}\ \cdots \land \mathsf{EX}(\mathbb{F}(a_{(i+n-1) \bmod n}))\cdots)))$ and $E_k = \text{count}(a_{k-1 \bmod n}) \land \neg(\bigvee_{b \in \text{init}(t_{k-1 \bmod n})} \mathbb{F}(b)) \land \bigvee_{b \in \text{init}(t_k) \setminus \{\theta\}} \mathbb{F}(b) \land \mathsf{EX}(\mathbb{F}(t_k))$.

2. If $\theta \in \text{init}(t_{k-1 \bmod n})$ and $a_k \neq \theta$ then we change the exit formula $E_{k-1 \bmod n}$ so that it contains two components. If any action in $\text{init}(t_{k-1 \bmod n})$ is available then such an action can be taken, so a first component is $\text{count}(a_{k-1 \bmod n}) \land \mathsf{EX}\ (\bigvee_{b \in \text{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \land \mathbb{F}(t_{k-1 \bmod n})$. Note that any θ top-level branch in $t_{k-1 \bmod n}$ is invalidated (since some action $b \in \text{init}(t_{k-1 \bmod n})$ is available). The top-level θ branch of $t_{k-1 \bmod n}$ can be taken only if no action from $\text{init}(t_{k-1 \bmod n}) \cup \{a_k\}$ is available, so the second variant is $\mathbb{F}(a_{k-1 \bmod n}) \land \mathsf{EX}\ \neg\mathbb{F}(a_{(k)}) \land \neg(\bigvee_{b \in \text{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \land \mathbb{F}(t_{k-1 \bmod n}(\theta))$, where $t_{k-1 \bmod n} = t' \mathbin{\Box} \theta; t_{k-1 \bmod n}(\theta)$ for some test t' (recall that we can assume without loss of generality that there exists a single top-level θ branch in $t_{k-1 \bmod n}$).

   We take the disjunction of the above variants and so $E_{k-1 \bmod n} = \text{count}(a_{k-1 \bmod n}) \land \mathsf{EX}(\bigvee_{b \in \text{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \land \mathbb{F}(t_{k-1 \bmod n}) \lor \neg\mathbb{F}(a_{(k)}) \land \neg(\bigvee_{b \in \text{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \land \mathbb{F}(t_{k-1 \bmod n}(\theta))$.

3. If $a_k = \theta$ and $\theta \in \text{init}(t_{k-1 \bmod n})$, then we must modify the cycle as well as the exit test. Let $B = \text{init}(t_{k-1 \bmod n}) \setminus \{\theta\}$.

If an action from B is available the loop cannot continue, so we replace in C all occurrences of a_k with $\neg(\bigvee_{b\in B}\mathbb{F}(b)$ so that $C_i = \mathsf{EG}(\mathbb{F}(a_i)\wedge\mathsf{EX}(\cdots\wedge\mathsf{EX}(\mathbb{F}(\neg(\bigvee_{b\in\mathrm{init}(t_{k-1 \bmod n})\setminus\{\theta\}}\mathbb{F}(b)))\wedge\mathsf{EX}\cdots\wedge \mathsf{EX}(\mathbb{F}(a_{(i+n-1) \bmod n}))\cdots)))$.

Similarly, when actions from B are available the non-θ component of the exit test is applicable, while the θ branch can only be taken when no action from B is offered. Therefore we have $E_{k-1 \bmod n} = \mathrm{count}(a_{k-1 \bmod n})\wedge\mathsf{EX}\ \bigvee_{b\in\mathrm{init}(t_{k-1 \bmod n})\setminus\{\theta\}}\mathbb{F}(b)\wedge\mathbb{F}(t_{k-1 \bmod n})\vee \neg(\bigvee_{b\in\mathrm{init}(t_{k-1 \bmod n})\setminus\{\theta\}}\mathbb{F}(b))\wedge\mathbb{F}(t_{k-1 \bmod n}(\theta)))$. As before, $t_{k-1 \bmod n}(\theta)$ is the θ-branch of $t_{k-1 \bmod n}$ that is, $t_{k-1 \bmod n} = t'\ \square\ \theta; t_{k-1 \bmod n}(\theta)$ for some test t'.

Finally, recall that originally $E_k = \mathrm{count}(a_k)\wedge\mathsf{EX}\ \mathbb{F}(t_k)$. Now however $a_k = \theta$ and so we must apply to E_k the same process that we repeatedly performed earlier namely, adding $\neg(\bigvee_{b\in\mathrm{init}(t_{k-1 \bmod n})\setminus\{\theta\}}\mathbb{F}(b))$. In addition, θ does not consume any input by definition, so the EX construction disappears. In all we have $E_k = \mathrm{count}(a_{k-1})\wedge\neg(\bigvee_{b\in\mathrm{init}(t_{k-1 \bmod n})\setminus\{\theta\}}\mathbb{F}(b))\wedge\mathbb{F}(t_k)$ (when $a_k = \theta$, we do not need to count a_k).

We now prove that the construction described above is correct. We focus first on the initial, θ-less formula.

If the common actions are available for both p and t then $p \stackrel{a_i}{\Longrightarrow} p_1 \wedge p_1 \stackrel{a_{i+1}}{\Longrightarrow} \wedge\cdots\wedge p_{n-1} \stackrel{a_{i+n-1}}{\Longrightarrow} p$, which shows that process p performs some actions in the cycle. We further notice that these are equivalent to $\mathbb{K}(p)\models a_i$ and $\mathbb{K}(p_1)\models a_{i+1}\wedge\cdots\wedge\mathbb{K}(p_{n-1})\models a_{i+n-1}$, respectively. Therefore $p \stackrel{a_i}{\Longrightarrow} p_1 \wedge p_1 \stackrel{a_{i+1}}{\Longrightarrow} \wedge\cdots\wedge p_{n-1} \stackrel{a_{i+n-1}}{\Longrightarrow} p$ iff $\mathbb{K}(p)\models \mathsf{EG}(\mathbb{F}(a_i)\wedge\mathsf{EX}(\mathbb{F}(a_{(i+1) \bmod n})\wedge\mathsf{EX}\cdots\wedge\mathsf{EX}(\mathbb{F}(a_{(i+n-1) \bmod n}))\cdots))$. That is,

$$p \stackrel{a_i}{\Longrightarrow} p_1 \wedge p_1 \stackrel{a_{i+1}}{\Longrightarrow} \wedge\cdots\wedge p_{n-1} \stackrel{a_{i+n-1}}{\Longrightarrow} p \text{ iff } \mathbb{K}(p)\models C_i \tag{3}$$

We exit from the cycle as follows: When $p \stackrel{a_i}{\Longrightarrow} p' \stackrel{a_{j+1}}{\not\Longrightarrow}$ then the process p must take the test t_i after performing a_i and pass it. If however the action a_{i+1} is available in the cycle as well as in the test t_i, then it depends on the process p whether it will continue in the cycle or will take the test t_i. That means p may take t_i and pass the test or it may decide to continue in the cycle. Eventually however the process must take one of the exit tests. Given the nature of may-testing one successful path is enough for p to pass t.

Formally, we note that $p \stackrel{a_i}{\Longrightarrow} p' \wedge p'$may t_i is equivalent to $\mathbb{K}(p)\models \mathrm{count}(a_i)\wedge\mathsf{EX}(\mathbb{F}(t_i))$. Indeed, when p and t perform a_i the test t is i actions away from $\mathrm{start}(a_0)$ and so $\mathrm{count}(a_i)$ is true. Therefore given the inductive hypothesis (that p' may t_i iff $\mathbb{K}(p')\models\mathbb{F}(t_i)$ for any process p') we concluded that:

$$(p \stackrel{a_i}{\Longrightarrow} p' \wedge p' \text{ may } t_i) \text{ iff } \mathbb{K}(p)\models(\mathrm{count}(a_i)\wedge\mathsf{EX}(\mathbb{F}(t_i))) = E_i \tag{4}$$

Taking the disjunction of Relations (4) over all $0 \le i < n$ we have

$$(p \stackrel{a_i}{\Longrightarrow} p' \wedge p' \text{ may } \Sigma_{i=0}^{n-1} t_i) \text{ iff } \mathbb{K}(p)\models\bigvee_{i=0}^{n-1} E_i \tag{5}$$

The correctness of $\mathbb{F}(t)$ is then the direct consequences of Relations (3) and (5): We can stay in the cycle as long as one C_i remains true (Relation (3)), and we can exit at any time using the appropriate exit test (Relation (5)).

Next we consider the possible deadlock detection action as introduced in the three cases above. We have:

1. Let $a_k = \theta$, $\theta\notin\mathrm{init}(t_{k-1 \bmod n})$ and suppose that $p\|_\theta t$ runs along such that they reach the point $p'\|_\theta t' \stackrel{a_{k-1 \bmod n}}{\longrightarrow} p''\|_\theta t''$. Let $b\in\mathrm{init}(t_{k-1 \bmod n})$. If $p''\|_\theta t'' \stackrel{b}{\longrightarrow}$ then the run must exit the cycle according to the definition of $\|_\theta$. At the same time C_k is false because the disjunction

$\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n})} \mathbb{F}(b)$ is true and no other C_i is true, so C is false and therefore the only way for $\mathbb{F}(t)$ to be true is for E_k to be true. The two, testing and logic scenarios are clearly equivalent. On the other hand, if $p'' \|_\theta t'' \stackrel{b}{\not\longrightarrow}$ for any $b \in \mathrm{init}(t_{k-1 \bmod n})$, then the test must take the θ branch. At the same time C_k is true and so is C, whereas E_k is false (so the formula must "stay in the cycle"), again equivalent to the test scenario.

2. Now $\theta \in \mathrm{init}(t_{k-1 \bmod n})$ and $a_k \neq \theta$. The way the process and the test perform a_k and remain in the cycle is handled by the general case so we are only considering the exit test $t_{k-1 \bmod n}$. The only supplementary consequence of a_k being available is that any θ branch in $t_{k-1 \bmod n}$ is disallowed, which is still about the exit test rather than the cycle.

   There are two possible successful runs that involve the exit test $t_{k-1 \bmod n}$. First, $p \stackrel{a_{k-1 \bmod n}}{\Longrightarrow} p' \wedge \exists b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\} : p' \stackrel{b}{\Longrightarrow} \wedge\ p'$ may $t_{k-1 \bmod n}$. Second, $p \stackrel{a_{k-1 \bmod n}}{\Longrightarrow} p' \wedge \neg(\exists b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\} : p' \stackrel{b}{\Longrightarrow}) \wedge p' \stackrel{a_k}{\not\Longrightarrow} \wedge\ p'$ may $t_{k-1 \bmod n}(\theta)$. The first case corresponds to a common action b being available to both the process and the test (case in which the θ branch of $t_{k-1 \bmod n}$ is forbidden by the semantics of p' may $t_{k-1 \bmod n}$). The second case requires that the θ branch of the test is taken whenever no other action is available.

   Given the inductive hypothesis (that p' may t_i iff $\mathbb{K}(p') \models \mathbb{F}(t_i)$ for any process p') we have

$$\begin{aligned} &p \stackrel{a_{k-1 \bmod n}}{\Longrightarrow} p' \wedge \exists b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\} : p' \stackrel{b}{\Longrightarrow} \wedge p' \text{ may } t_{k-1 \bmod n} \text{ iff} \\ &\mathbb{K}(p) \models \mathrm{count}(a_{k-1 \bmod n}) \wedge \mathsf{EX} \textstyle\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b) \wedge \mathbb{F}(t_{k-1 \bmod n}) \end{aligned} \tag{6}$$

$$\begin{aligned} &p \stackrel{a_{k-1 \bmod n}}{\Longrightarrow} p' \wedge \neg(\exists b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\} : p' \stackrel{b}{\Longrightarrow}) \wedge p' \stackrel{a_k}{\not\Longrightarrow} \wedge \\ &p' \text{ may } t_{k-1 \bmod n}(\theta) \text{ iff } \mathbb{K}(p) \models \mathrm{count}(a_{k-1 \bmod n}) \wedge \mathsf{EX}\, \neg\mathbb{F}(a_{(k)}) \wedge \\ &\neg(\textstyle\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \wedge \mathbb{F}(t_{k-1 \bmod n}(\theta)) \end{aligned} \tag{7}$$

   The conjunction of Relations (6) and (7) establish this case. Indeed, the left hand sides of the two relations are the only two ways to have a successful run involving $t_{k-1 \bmod n}$ (as argued above).

3. Let now $a_k = \theta$ and $\theta \in \mathrm{init}(t_{k-1 \bmod n})$. Suppose that the process under test is inside the cycle and has reached a state p such that $p \stackrel{a_{k-1 \bmod n}}{\Longrightarrow} p'$, meaning that p' is ready to either continue within the cycle or pass $t_{k-1 \bmod n}$.

   Suppose first that $p' \stackrel{b}{\Longrightarrow}$ for some $b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}$. Then (a) p' cannot continue in the cycle, which is equivalent to C_k being false (since no C_i, $i \neq k$ can be true), and so (b) p' must pass $t_{k-1 \bmod n}$, which is equivalent to $\mathbb{K}(p') \models \bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b) \wedge \mathbb{F}(t_{k-1 \bmod n})$. That C_k is false happens because $\neg(\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b))$ is false. Note incidentally that the θ branch of $t_{k-1 \bmod n}$ is forbidden, but this is guaranteed by the semantics of p' passing $t_{k-1 \bmod n}$ (and therefore by the semantics of $\mathbb{K}(p') \models \mathbb{F}(t_{k-1 \bmod n})$ by inductive hypothesis).

   Suppose now that $p' \stackrel{b}{\not\Longrightarrow}$ for any $b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}$. Then the only possible continuations are (a) p' remaining in the cycle which is equivalent to C_k being true (ensured by $\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)$ being false), or (b) p' taking the θ branch of $t_{k-1 \bmod n}$, which is equivalent to $\neg(\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)) \wedge \mathbb{F}(t_k(\theta))$ by the fact that $\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)$ is false and the inductive hypothesis, or (c) p' taking the test t_k (which falls just after $a_k = \theta$ and so it is an alternative in the deadlock detection branch), which is equivalent to E_k being true, ensured by $\bigvee_{b \in \mathrm{init}(t_{k-1 \bmod n}) \setminus \{\theta\}} \mathbb{F}(b)$ being false and $\mathbb{F}(t_k)$ being true iff p' passes t_k by inductive hypothesis.

   Once more taking the disjunction of the two alternatives above establishes this case.

□

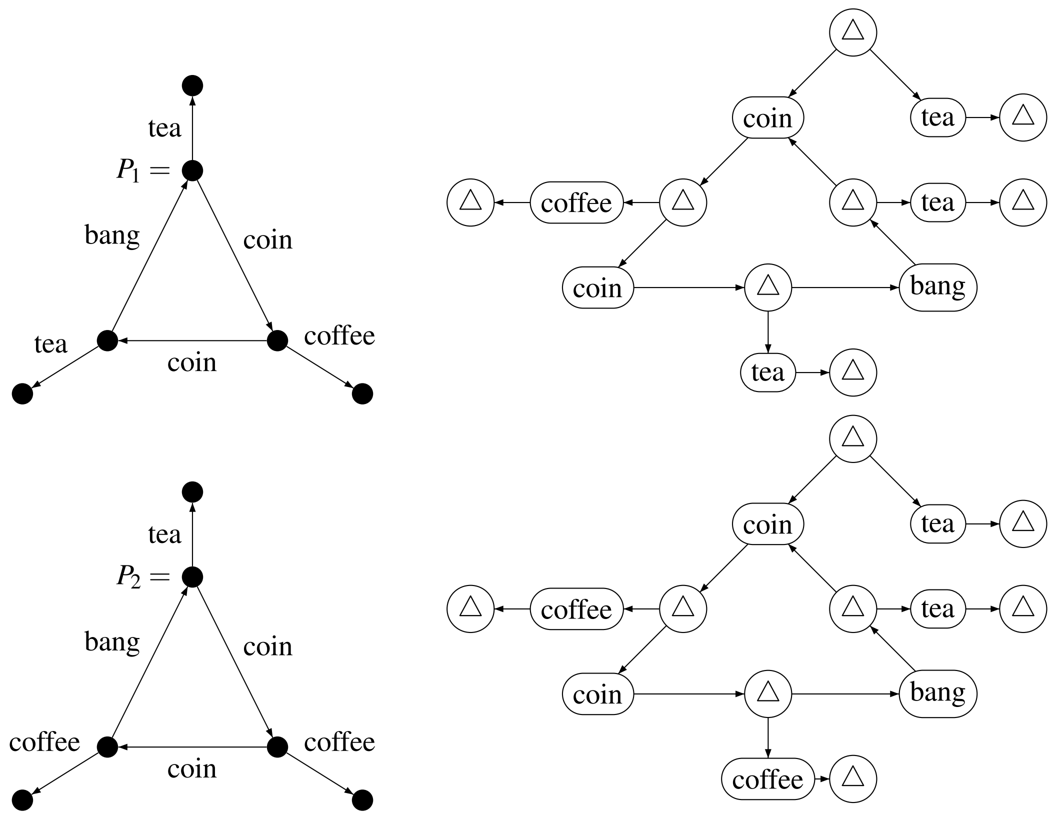


Figure 4: Yet another couple of vending machines P_1 and P_2 (left) and the equivalent Kripke structures $\mathbb{X}(P_1)$ and $\mathbb{X}(P_2)$ (right).

Example 3. COMPACT CTL FORMULA GENERATION:

In this section we illustrate the conversion of failure trace test into CTL formulae. For this purpose consider the following simple vending machines:

$$\begin{aligned} P_1 &= \text{coin}; (\text{coffee} \,\Box\, \text{coin};\ (\text{tea} \,\Box\, \text{bang}; (\text{tea} \,\Box\, P_1))) \\ P_2 &= \text{coin}; (\text{coffee} \,\Box\, \text{coin};\ (\text{coffee} \,\Box\, \text{bang}; (\text{tea} \,\Box\, P_2))) \end{aligned}$$

These two machines dispense a coffee after accepting a coin; the first machine dispenses a tea after second coin, whereas the second still dispenses coffee. After two coins and a hit by customer, the two machines will dispense tea. The vending machines as well as the equivalent Kripke structures are shown in Figure 4.

Consider now the following test:

$$t = \text{coin}; (\text{coffee}; \text{pass} \,\Box\, \text{coin}; (\text{tea}; \text{pass} \,\Box\, \text{bang}; (\text{tea}; \text{pass} \,\Box\, t)))$$

Using Theorem 7 (and thus implicitly Lemma 6) we can convert this test into the following

CTL formula (after eliminating all the obviously true sub-formulae):

$$\begin{aligned}\mathbb{F}_{\mathbb{X}}(t) \quad = \quad & (\mathsf{EG}\ (\text{coin}) \wedge \mathsf{EX}(\text{coin} \wedge \mathsf{EX}((\text{bang}) \wedge \mathsf{EX}(\text{coin}))) \vee \\ & \mathsf{EG}\ (\text{coin}) \wedge \mathsf{EX}(\text{bang} \wedge \mathsf{EX}((\text{coin}) \wedge \mathsf{EX}(\text{coin}))) \vee \\ & \mathsf{EG}\ (\text{bang}) \wedge \mathsf{EX}(\text{coin} \wedge \mathsf{EX}((\text{coin}) \wedge \mathsf{EX}(\text{bang}))) \\ & \mathsf{U}\ \mathsf{EG}\ \text{start}(\text{coin}) \wedge \mathsf{EX}\ (\text{coffee}) \vee \mathsf{EG}\ \text{start}(\text{coin}) \\ & \wedge\ \mathsf{EX}(\text{coin} \wedge \mathsf{EX}(\text{tea})) \vee \mathsf{EG}\ \text{start}(\text{coin}) \wedge \mathsf{EX}(\text{coin} \wedge \mathsf{EX}(\text{bang} \wedge \mathsf{EX}(\text{tea})))) \\ & \vee\ \mathsf{EG}(\text{tea})\end{aligned}$$

It is not difficult to see that the meaning of this formula is equivalent to the meaning of t. Indeed, the following is true for both the test t as well as the formula $\mathbb{F}_{\mathbb{X}}(t)$: tea is offered in the first step without any coin, or after a coin coffee is offered, or after two coins tea is offered, or after two coins and customer hits tea is offered; On the other hand after two coins and a bang, and if no coin is available next, then tea will be offered. Finally, after two coins and a bang if both coin and tea are available, then two options comes out: we can either keep in the cycle, or offer a tea. In all the process can remain in the cycle indefinitely or can exit from the cycle and pass the test. We can get coffee or tea at the first cycle or after few repetitions of the cycle.

The formula $\mathbb{F}_{\mathbb{X}}(t)$ holds for $\mathbb{X}(P_1)$ but it does not hold for $\mathbb{X}(P_2)$.

5.3 From CTL Formulae to Failure Trace Tests

We find it convenient to show first how to consider logical (but not temporal) combinations of tests.

Lemma 8. *For any test $t \in \mathscr{T}$ there exists a test $\bar{t} \in \mathscr{T}$ such that p* may *t if and only if $\neg(p \text{ may } \bar{t})$ for any $p \in \mathscr{P}$.*

Proof. We modify t to produce $\bar{t}$ as follows: We first force all the success states to become deadlock states by eliminating the outgoing action γ from t completely. If the action that leads to a state thus converted is θ then this action is removed as well (indeed, the "fail if nothing else works" phenomenon thus eliminated is implicit in testing). Finally, we add to all the states having their outgoing transitions θ or γ eliminated in the previous steps one outgoing transition labeled θ followed by one outgoing transition labeled γ.

The test $\bar{t}$ must fail every time the original test t succeeds. The first step (eliminating γ transitions) has exactly this effect.

In addition, we must ensure that $\bar{t}$ succeeds in all the circumstances in which t fails. The extra θ outgoing actions (followed by γ) ensure that whevever the end of the run reaches a state that was not successful in t (meaning that it had no outgoing γ transition) then this run is extended in $\bar{t}$ via the θ branch to a success state, as desired. □

Lemma 9. *For any two tests $t_1, t_2 \in \mathscr{T}$ there exists a test $t_1 \vee t_2 \in \mathscr{T}$ such that p* may *$(t_1 \vee t_2)$ if and only if $(p \text{ may } t_1) \vee (p \text{ may } t_2)$ for any $p \in \mathscr{P}$.*

Proof. We construct such a disjunction on tests by induction over the structure of tests.

For the base case it is immediate that pass $\vee\, t = t \,\vee$ pass $=$ pass and stop $\vee\, t = t \,\vee$ stop $= t$ for any structure of the test t.

For the induction step we consider without loss of generality that t_1 and t_2 have the following structure:

$$\begin{aligned} t_1 &= \Sigma\{b_1;t_1(b_1) : b_1 \in B_1\} \,\Box\, \theta;t_{N1} \\ t_2 &= \Sigma\{b_2;t_2(b_2) : b_2 \in B_2\} \,\Box\, \theta;t_{N2} \end{aligned}$$

Indeed, all the other possible structures of t_1 (and t_2) are covered by such a form since θ not appearing on the top level of t_1 (or t_2) is equivalent to $t_{N1} = \text{stop}$ (or $t_{N2} = \text{stop}$), while not having a choice on the top level of the test is equivalent to B_1 (or B_2) being an appropriate singleton.

We then construct $t_1 \vee t_2$ for the form of t_1 and t_2 mentioned above under the inductive assumption that the disjunction between any of the "inner" tests $t_1(b_1)$, t_{N1} $t_2(b_2)$, and t_{N2} is known. We have:

$$\begin{aligned} \Sigma\{b_1;t_1(b_1) : b_1 \in B_1\} \,\Box\, \theta;t_{N1} \quad \vee \quad & \Sigma\{b_2;t_2(b_2) : b_2 \in B_2\} \,\Box\, \theta;t_{N2} \\ = \quad & \Sigma\{b;(t_1(b) \vee t_2(b)) : b \in B_1 \cap B_2\} \quad \Box \\ & \Sigma\{b;(t_1(b) \vee [t_{N2}]_b) : b \in B_1 \setminus B_2\} \quad \Box \\ & \Sigma\{b;(t_2(b) \vee [t_{N1}]_b) : b \in B_2 \setminus B_1\} \quad \Box \\ & \theta;(t_{N1} \vee t_{N2}) \end{aligned} \tag{8}$$

where $[t]_b$ is the test t restricted to performing b as its first action and so is inductively constructed as follows:

1. If $t = \text{stop}$ then $[t]_b = \text{stop}$.
2. If $t = \text{pass}$ then $[t]_b = \text{pass}$.
3. If $t = b;t'$ then $[t]_b = t'$.
4. If $t = a;t'$ with $a \neq b$ then $[t]_b = \text{stop}$.
5. If $t = \mathbf{i};t'$ then $[t]_b = [t']_b$.
6. If $t = t' \,\Box\, t''$ such that neither t' nor t'' contain θ in their topmost choice then $[t]_b = [t']_b \,\Box\, [t'']_b$.
7. If $t = b;t' \,\Box\, \theta;t''$ then $[t]_b = t'$.
8. If $t = a;t' \,\Box\, \theta;t''$ with $a \neq b$ then $[t]_b = [t'']_b$.

If the test $t_1 \vee t_2$ is offered an action b that is common to the top choices of the two tests t_1 and t_2 ($b \in B_1 \cap B_2$) then the disjunction succeeds if and only if b is performed and then at least one of the tests $t_1(b)$ and $t_2(b)$ succeeds (meaning that $t_1(b) \vee t_2(b)$ succeeds inductively) afterward. The first term of the choice in Equation (8) represents this possibility.

If the test is offered an action b that appears in the top choice of t_1 but not in the top choice of t_2 ($b \in B_1 \setminus B_2$) then the disjunction succeeds if and only if $t_1(b)$ succeeds after b is performed, or t_{N2} performs b and then succeeds; the same goes for $b \in B_2 \setminus B_1$ (only in reverse). The second and the third terms of the choice in Equation (8) represent this possibility.

Finally whenever the test $t_1 \vee t_2$ is offered an action b that is in neither the top choices of the two component tests t_1 and t_2 (that is, $b \notin B_1 \cup B_2$), then the disjunction succeeds if and only if at least one of the tests t_{N1} or t_{N2} succeeds (or equivalently $t_{N1} \vee t_{N2}$ succeeds inductively). This is captured by the last term of the choice in Equation (8). Indeed, $b \notin B_1 \cup B_2$ implies that $b \notin (B_1 \cap B_2) \cup (B_1 \setminus B_2) \cup (B_2 \setminus B_1)$ and so such an action will trigger the deadlock detection (θ) choice.

There is no other way for the test $t_1 \vee t_2$ to succeed so the construction is complete. □

We believe that an actual example of how disjunction is constructed is instructive. The following is therefore an example to better illustrate disjunction over tests. A further example (incorporating temporal operators and also negation) will be provided later (see Example 5).

Example 4. A DISJUNCTION OF TESTS:
Consider the construction $t_1 \vee t_2$, where:

$$\begin{aligned} t_1 &= (\text{bang};\text{tea};\text{pass}) \,\Box \\ &\quad\; (\text{coin};(\text{coffee};\text{stop} \,\Box\, \theta;\text{pass})) \end{aligned}$$

$$\begin{aligned} t_2 &= (\text{coin};\text{pass}) \,\Box \\ &\quad\; (\text{nudge};\text{stop}) \,\Box \\ &\quad\; (\theta;(\text{bang};\text{water};\text{pass} \,\Box\, \text{turn};\text{stop})) \end{aligned}$$

Using the notation from Equation (8) we have $B_1 = \{\text{coin},\text{bang}\}$, $B_2 = \{\text{coin},\text{nudge}\}$ and so $B_1 \cap B_2 = \{\text{coin}\}$, $B_1 \setminus B_2 = \{\text{bang}\}$, and $B_2 \setminus B_1 = \{\text{nudge}\}$. We further note that $t_1(\text{coin}) = \text{coffee};\text{stop} \,\Box\, \theta;\text{pass}$, $t_1(\text{bang}) = \text{tea};\text{pass}$, $t_{N1} = \text{stop}$, $t_2(\text{coin}) = \text{pass}$, $t_2(\text{nudge}) = \text{stop}$, and $t_{N2} = \text{bang};\text{water};\text{pass} \,\Box\, \text{turn};\text{stop}$. Therefore we have:

$$\begin{aligned} t_1 \vee t_2 &= (\text{coin};(t_1(\text{coin}) \vee t_2(\text{coin}))) \,\Box \\ &\quad\; (\text{bang};(t_1(\text{bang}) \vee [t_{N2}]_{\text{bang}})) \,\Box \\ &\quad\; (\text{nudge};(t_2(\text{nudge}) \vee [t_{N1}]_{\text{nudge}})) \,\Box \\ &\quad\; \theta;(t_{N1} \vee t_{N2}) \\ &= (\text{coin};\text{pass}) \,\Box \\ &\quad\; (\text{bang};(t_1(\text{bang}) \vee [t_{N2}]_{\text{bang}})) \,\Box \\ &\quad\; (\text{nudge};\text{stop}) \,\Box \\ &\quad\; (\theta;(\text{bang};\text{water};\text{pass} \,\Box\, \text{turn};\text{stop})) \end{aligned}$$

Indeed, $t_1(\text{coin}) = \text{pass}$ so $t_1(\text{coin}) \vee t_2(\text{coin}) = \text{pass}$; $[t_{N1}]_{\text{nudge}} = \text{stop}$; and $t_{N1} = \text{stop}$ so $t_{N1} \vee t_{N2} = t_{N2}$.

We further have $[t_{N2}]_{\text{bang}} = \text{water};\text{pass}$, and therefore $t_1(\text{bang}) \vee [t_{N2}]_{\text{bang}} = (\text{tea};\text{pass}) \vee (\text{water};\text{pass})$. We proceed inductively as above, except that in this degenerate case $t_{N1} = t_{N2} = \text{stop}$ and $B_1 \cap B_2 = \emptyset$, so the result is a simple choice between the components: $t_1(\text{bang}) \vee [t_{N2}]_{\text{bang}} = (\text{tea};\text{pass}) \,\Box\, (\text{water};\text{pass})$. Overall we reach the following result:

$$\begin{aligned} t_1 \vee t_2 &= (\text{coin};\text{pass}) \,\Box \\ &\quad\; (\text{bang};((\text{tea};\text{pass}) \,\Box\, (\text{water};\text{pass}))) \,\Box \\ &\quad\; (\text{nudge};\text{stop}) \,\Box \\ &\quad\; (\theta;(\text{bang};\text{water};\text{pass} \,\Box\, \text{turn};\text{stop})) \end{aligned} \tag{9}$$

Intuitively, t_1 specifies that we can have tea if we hit the machine, and if we put a coin in we can get anything except coffee. Similarly t_2 specifies that we can put a coin in the machine, we cannot nudge it, and if none of the above happen then we can hit the machine (case in which we get water) but we cannot turn it upside down. On the other hand, the disjunction of these tests as shown in Equation (9) imposes the following specification:

1. If a coin is inserted then the test succeeds. Indeed, this case from t_2 supersedes the corresponding special case from t_1: the success of the test implies that the machine is allowed to do anything afterward, including not dispensing coffee.
2. We get either tea or water after hitting the machine. Getting tea comes from t_1 and getting water from t_2 (where the bang event comes from the θ branch).
3. If we nudge the machine then the test fails immediately; this comes directly from t_2.
4. In all the other cases the test behaves like the θ branch of t_2. This behaviour makes sense since there is no such a branch in t_1.

We can thus see how the disjunction construction from Lemma 9 makes intuitive sense.

This all being said, note that we do not claim that either of the tests t_1 and t_2 are useful in any way, and so we should not be held responsible for the behaviour of any machine built according to the disjunctive specification shown above.

Corollary 10. *For any tests $t_1, t_2 \in \mathscr{T}$ there exists a test $t_1 \wedge t_2 \in \mathscr{T}$ such that p may $(t_1 \wedge t_2)$ if and only if $(p \text{ may } t_1) \wedge (p \text{ may } t_2)$ for any $p \in \mathscr{P}$.*

Proof. Immediate from Lemmata 8 and 9 using the De Morgan rule $a \wedge b = \neg(\neg a \vee \neg b)$. □

We are now ready to show that any CTL formula can be converted into an equivalent failure trace test.

Lemma 11. *There exists a function $\mathbb{T}_{\mathbb{K}} : \mathscr{F} \to \mathscr{T}$ such that $\mathbb{K}(p) \models f$ if and only if p may $\mathbb{T}_{\mathbb{K}}(f)$ for any $p \in \mathscr{P}$.*

Proof. The proof is done by structural induction over CTL formulae. As before, the function $\mathbb{T}_{\mathbb{K}}$ will also be defined recursively in the process.

We have naturally $\mathbb{T}_{\mathbb{K}}(\top) = \text{pass}$ and $\mathbb{T}_{\mathbb{K}}(\bot) = \text{stop}$. Clearly any Kripke structure satisfies $\top$ and any process passes pass, so it is immediate that $\mathbb{K}(p) \models \top$ if and only if p may pass $= \mathbb{T}_{\mathbb{K}}(\top)$. Similarly $\mathbb{K}(p) \models \bot$ if and only if p may stop is immediate (neither is ever true).

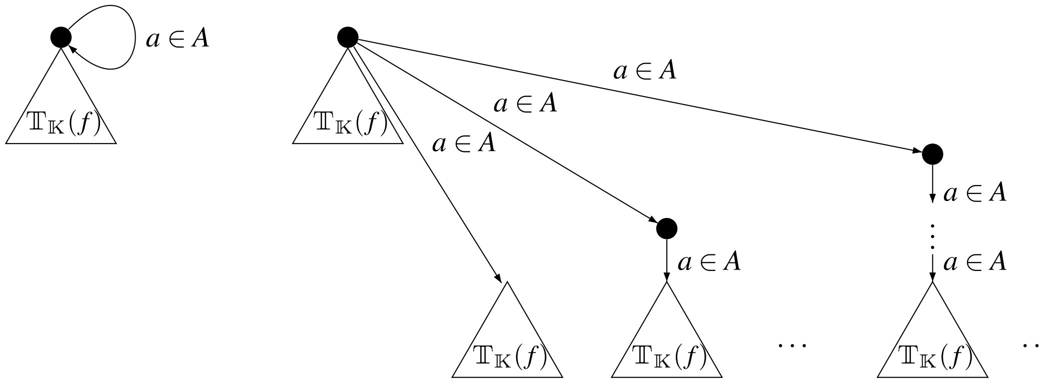
To complete the basis we have $\mathbb{T}_{\mathbb{K}}(a) = a;\text{pass}$, which is an immediate consequence of the definition of $\mathbb{K}$. Indeed, the construction defined in Theorem 2 ensures that for every outgoing action a of an LTS process p there will be an initial Kripke state in $\mathbb{K}(p)$ where a holds and so p may $a;\text{pass}$ if and only if $\mathbb{K}(p) \models a$.

The constructions for non-temporal logical operators have already been presented in Lemma 8, Lemma 9, and Corollary 10. We therefore have $\mathbb{T}_{\mathbb{K}}(\neg f) = \overline{\mathbb{T}_{\mathbb{K}}(f)}$ with $\overline{\mathbb{T}_{\mathbb{K}}(f)}$ as constructed in Lemma 8, while $\mathbb{T}_{\mathbb{K}}(f_1 \vee f_2) = \mathbb{T}_{\mathbb{K}}(f_1) \vee \mathbb{T}_{\mathbb{K}}(f_2)$ and $\mathbb{T}_{\mathbb{K}}(f_1 \wedge f_2) = \mathbb{T}_{\mathbb{K}}(f_1) \wedge \mathbb{T}_{\mathbb{K}}(f_2)$ with test disjunction and conjunction as in Lemma 9 and Corollary 10, respectively. That these constructions are correct follow directly from the respective lemmata and corollary.

We then move to the temporal operators.

We have $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f) = \Sigma\{a; \mathbb{T}_{\mathbb{K}}(f) : a \in A\}$. When applied to some process p the resulting test performs any action $a \in A$ and then (in the next state p' such that $p \xrightarrow{a} p'$) gives control to $\mathbb{T}_{\mathbb{K}}(f)$. $\mathbb{T}_{\mathbb{K}}(f)$ will always test the next state (since there is no θ in the top choice), and there is no restriction as to what particular next state p' is expected (since any action of p is accepted by the test).

Concretely, suppose that p may $\Sigma\{a; \mathbb{T}_{\mathbb{K}}(f) : a \in A\}$. Then there exists some action $a \in A$ such that p performs a, becomes p', and p' may $\mathbb{T}_{\mathbb{K}}(f)$ (by definition of may testing). It follows by inductive assumption that $\mathbb{K}(p') \models f$ and so $\mathbb{K}(p) \models \mathsf{X}\ f$ (since $\mathbb{K}(p')$ is one successor of $\mathbb{K}(p)$). Conversely, by the definition of $\mathbb{K}$ all successors of $\mathbb{K}(p)$ have the form $\mathbb{K}(p')$ such that $p \xrightarrow{a} p'$ for some $a \in$



(a) (b)

Figure 5: Test equivalent to the CTL formula $\mathsf{EF}\ f$ (a) and its unfolded version (b).

A. Suppose then that $\mathbb{K}(p) \models \mathsf{X}\ f$. Then there exists a successor $\mathbb{K}(p')$ of $\mathbb{K}(p)$ such that $\mathbb{K}(p') \models f$, which is equivalent to p' may $\mathbb{T}_{\mathbb{K}}(f)$ (by inductive assumption), which in turn implies that $(p = a; p')$ may $\Sigma\{a; \mathbb{T}_{\mathbb{K}}(f) : a \in A\}$ (by the definition of may testing), as desired.

We then have $\mathbb{T}_{\mathbb{K}}(\mathsf{EF}\ f) = t'$ such that $t' = \mathbb{T}_{\mathbb{K}}(f) \ \Box\ (\Sigma(a; t' : a \in A))$. The test t' is shown graphically in Figure 5(a). It specifies that at any given time the system under test has a choice to either pass $\mathbb{T}_{\mathbb{K}}(f)$ or perform some (any) action and then pass t' anew. Repeating this description recursively we conclude that to be successful the system under test can pass $\mathbb{T}_{\mathbb{K}}(f)$, or perform an action and then pass $\mathbb{T}_{\mathbb{K}}(f)$, or perform two actions and then pass $\mathbb{T}_{\mathbb{K}}(f)$, and so on. The overall effect (which is shown in Figure 5(b)) is that exactly all the processes p that perform an arbitrary sequence of actions and then pass $\mathbb{T}_{\mathbb{K}}(f)$ at the end of this sequence will pass $t' = \mathbb{T}_{\mathbb{K}}(\mathsf{EF}\ f)$. Given the inductive assumption that $\mathbb{T}_{\mathbb{K}}(f)$ is equivalent to f this is equivalent to $\mathbb{K}(p)$ being the start of an arbitrary path to some state that satisfies f, which is precisely the definition of $\mathbb{K}(p) \models \mathsf{EF}\ f$, as desired.

Following a similar line of thought we have $\mathbb{T}_{\mathbb{K}}(\mathsf{EG}\ f) = \mathbb{T}_{\mathbb{K}}(f) \wedge (\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f') \ \Box\ \theta; \text{pass})$, with $f' = f \wedge \mathsf{EX}\ f'$. Suppose that p may $\mathbb{T}_{\mathbb{K}}(\mathsf{EG}\ f)$. This implies that p may $\mathbb{T}_{\mathbb{K}}(f)$. This also implies that p may $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f')$ but only unless $p = \text{stop}$; indeed, the θ choice appears in conjunction with a multiple choice that offers all the possible alternatives (see the conversion for EX above) and so can only be taken if no other action is available.

By inductive assumption p may $\mathbb{T}_{\mathbb{K}}(f)$ if and only if $\mathbb{K}(p) \models f$. By the conversion of EX (see above) p may $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f')$ if and only if $\mathbb{K}(p') \models f'$ where $p \xrightarrow{a} p'$ for some $a \in A$ and so $\mathbb{K}(p')$ is the successor of $\mathbb{K}(p)$ on some path. We thus have p may $\mathbb{T}_{\mathbb{K}}(\mathsf{EG}\ f)$ if and only if $\mathbb{K}(p) \models f$ and $\mathbb{K}(p') \models f'$ for some successor p' of p. Repeating the reasoning above recursively (starting from p', etc.) we conclude that p may $\mathbb{T}_{\mathbb{K}}(\mathsf{EG}\ f)$ if and only if $\mathbb{K}(p_1) \models f$ for all the states $\mathbb{K}(p_1)$ on some path that starts from $\mathbb{K}(p)$, which is clearly equivalent to $\mathbb{K}(p) \models \mathsf{EG}\ f'$. The recursive reasoning terminates at the end of the path, when the LTS state p becomes stop, and the process is therefore released from its obligation to have states in which f holds (since no states exist any longer). In this case $\mathbb{K}(p)$ is a "sink" state with no successor (according to Item 3c in Theorem 2) and so the corresponding Kripke path is also at an end (and so there are no more states for f to hold in).

Finally we have $\mathbb{T}_{\mathbb{K}}(\mathsf{E}\ f_1\ \mathsf{U}\ f_2) = (\mathbb{T}_{\mathbb{K}}(f_1) \wedge (\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f') \ \Box\ \theta; \text{pass})) \ \Box\ \mathbf{i}; (\mathbb{T}_{\mathbb{K}}(f_2) \wedge (\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f'') \ \Box\ \theta; \text{pass}))$, with $f' = f_1 \wedge \mathsf{EX}\ f'$ and $f'' = f_2 \wedge \mathsf{EX}\ f''$. Following the same reasoning as above (in the EG case) the fact that a process p follows the test $\mathbb{T}_{\mathbb{K}}(f_1) \wedge \mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f')$ without deadlocking until some arbitrary state p' is reached is equivalent to $\mathbb{K}(p)$ featuring a path of arbitrary length ending in state $\mathbb{K}(p')$ whose states $\mathbb{K}(p_1)$ will all satisfy f_1. At the arbitrary (and nondeterministically chosen) point p' the test $\mathbb{T}_{\mathbb{K}}(\mathsf{E}\ f_1\ \mathsf{U}\ f_2)$ will exercise its choice and so p' has to pass $\mathbb{T}_{\mathbb{K}}(f_2) \wedge \mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ f'')$ in order for p to pass $\mathbb{T}_{\mathbb{K}}(\mathsf{E}\ f_1\ \mathsf{U}\ f_2)$. We follow once more the same reasoning as in the EG case and we thus conclude that this is equivalent to $\mathbb{K}(p_1) \vDash f$ for all the states $\mathbb{K}(p_1)$ on some path that starts from $\mathbb{K}(p')$. Putting the two phases together we have that p may $\mathbb{T}_{\mathbb{K}}(\mathsf{E}\ f_1\ \mathsf{U}\ f_2)$ if and only if $\mathbb{K}(p)$ features a path along which f_1 holds up to some state and f_2 holds from that state on, which is equivalent to $\mathbb{K}(p) \vDash \mathsf{E}\ f_1\ \mathsf{U}\ f_2$, as desired. The $\theta;\mathrm{pass}$ choices play the same role as in the EG case (namely, they account for the end of a path since they can only be taken when no other action is available).

Thus we complete the proof and the conversion between CTL formulae and sequential tests. Indeed, note that EX, EF, EG, and EU is a complete, minimal set of temporal operators for CTL [10], so all the remaining CTL constructs can be rewritten using only the constructs discussed above. ☐

Example 5. HOW TO TEST THAT YOUR COFFEE MACHINE IS WORKING:
We turn our attention again to the coffee machines b_1 and b_2 below, as presented earlier in Example 1, and also graphically as LTS in Figure 3:

$$\begin{aligned} b_1 &= \mathrm{coin};(\mathrm{tea} \,\Box\, \mathrm{bang};\mathrm{coffee}) \quad \Box \quad \mathrm{coin};(\mathrm{coffee} \,\Box\, \mathrm{bang};\mathrm{tea}) \\ b_2 &= \mathrm{coin};(\mathrm{tea} \,\Box\, \mathrm{bang};\mathrm{tea}) \quad \Box \quad \mathrm{coin};(\mathrm{coffee} \,\Box\, \mathrm{bang};\mathrm{coffee}) \end{aligned}$$

Also recall that the following CTL formula was found to differentiate between the two machines:

$$\phi = \mathrm{coin} \wedge \mathsf{EX}\ (\mathrm{coffee} \vee \neg\mathrm{coffee} \wedge \mathrm{bang} \wedge \mathsf{EX}\ \mathrm{coffee})$$

Indeed, the formula holds for both the initial states of $\mathbb{K}(b_1)$ (where coffee is offered from the outset or follows a hit on the machine) but holds in only one of the initial states of $\mathbb{K}(b_2)$ (the one that dispenses coffee).

Let $C = \{\mathrm{coin},\mathrm{tea},\mathrm{bang},\mathrm{coffee}\}$ be the set of all actions. We will henceforth convert silently all the tests $\theta;\mathrm{pass}$ into pass as long as these tests do not participate in a choice. We will also perform simplifications of the intermediate tests as we go along in order to simplify (and so clarify) the presentation.

The process of converting the formula ϕ to a test suite $\mathbb{T}_{\mathbb{K}}(\phi)$ then goes as follows:

1. We convert first $\mathrm{bang} \wedge \mathsf{EX}\ \mathrm{coffee} = \neg(\neg\mathrm{bang} \vee \neg\mathsf{EX}\ \mathrm{coffee})$.

   We have $\mathbb{T}_{\mathbb{K}}(\neg\mathrm{bang}) = \mathrm{bang};\mathrm{stop} \,\Box\, \theta;\mathrm{pass}$. On the other hand $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \mathrm{coffee}) = \Sigma\{a;\mathrm{coffee};\mathrm{pass} : a \in C\}$ and so $\mathbb{T}_{\mathbb{K}}(\neg\mathsf{EX}\ \mathrm{coffee}) = \Sigma\{a;\mathrm{coffee};\mathrm{stop} \,\Box\, \theta;\mathrm{pass} : a \in C\} \,\Box\, \theta;\mathrm{pass} = \Sigma\{a;\mathrm{coffee};\mathrm{stop} \,\Box\, \theta;\mathrm{pass} : a \in C\}$ (we ignore the topmost θ branch since the rest of the choice covers all the possible actions).

   Now we compute the disjunction $\mathbb{T}_{\mathbb{K}}(\neg\mathrm{bang}) \vee \mathbb{T}_{\mathbb{K}}(\neg\mathsf{EX}\ \mathrm{coffee})$. With the notations used in the proof of Lemma 9 we have $B_1 = \{\mathrm{bang}\}$, $B_2 = C$, $t_1(\mathrm{bang}) = \mathrm{stop}$, $t_{N1} = \mathrm{pass}$, $t_2(b) = \mathrm{coffee};\mathrm{stop} \,\Box\, \theta;\mathrm{pass}$ for all $b \in B_2$, and $t_{N2} = \mathrm{pass}$. We therefore have $\mathbb{T}_{\mathbb{K}}(\neg\mathrm{coffee}) \vee \mathbb{T}_{\mathbb{K}}(\neg\mathsf{EX}\ \mathrm{coffee}) = \mathrm{bang};(\mathrm{coffee};\mathrm{stop} \,\Box\, \theta;\mathrm{pass}) \,\Box\, \Sigma\{a;\mathrm{pass} : a \in C \setminus \{\mathrm{bang}\}\} \,\Box\, \theta;\mathrm{pass}$. Therefore:

$$\begin{aligned} \mathbb{T}_{\mathbb{K}}(\neg(\mathrm{bang} \wedge \mathsf{EX}\ \mathrm{coffee})) &= \mathrm{bang};(\mathrm{coffee};\mathrm{stop} \,\Box\, \theta;\mathrm{pass}) \\ &\quad \Box\, \theta;\mathrm{pass} \end{aligned} \tag{10}$$

   For brevity we integrated the $C \setminus \{\mathrm{bang}\}$ into the θ branch. Negating this test yields:

$$\mathbb{T}_{\mathbb{K}}(\mathrm{bang} \wedge \mathsf{EX}\ \mathrm{coffee}) = \mathrm{bang};\mathrm{coffee};\mathrm{pass}$$

Note in passing that that the negated version as shown in Equation (10) will suffice. The last formula is only provided for completeness and also as a checkpoint in the conversion process. Indeed, the equivalence between $\text{bang} \wedge \mathsf{EX}\ \text{coffee}$ and $\text{bang};\text{coffee};\text{pass}$ can be readily ascertained intuitively.

2. We move to $\neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee} = \neg(\text{coffee} \vee \neg(\text{bang} \wedge \mathsf{EX}\ \text{coffee}))$. By Equation (10) we have $\mathbb{T}_{\mathbb{K}}(\text{coffee} \vee \neg(\text{bang} \wedge \mathsf{EX}\ \text{coffee})) = \text{coffee};\text{pass} \vee (\text{bang};(\text{coffee};\text{stop} \ \Box\ \theta;\text{pass}) \ \Box\ \theta;\text{pass})$. This time $B_1 = \{\text{coffee}\}$, $B_2 = \{\text{bang}\}$, $t_1(b) = \text{pass}$, $t_2(b) = \text{coffee};\text{stop} \ \Box\ \theta;\text{pass}$, $t_{N1} = \text{stop}$, and $t_{N2} = \text{pass}$. Therefore $\mathbb{T}_{\mathbb{K}}(\text{coffee} \vee \neg(\text{bang} \wedge \mathsf{EX}\ \text{coffee})) = \text{coffee};\text{pass} \ \Box\ \text{bang};(\text{coffee};\text{stop} \ \Box\ \theta;\text{pass}) \ \Box\ \theta;\text{pass}$ (note that $B_1 \cap B_2 = \emptyset$). Negating this test yields:

$$\mathbb{T}_{\mathbb{K}}(\neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee}) = \text{coffee};\text{stop} \ \Box\ \text{bang};\text{coffee};\text{pass} \tag{11}$$

This is yet another checkpoint in the conversion, as the equivalence above can be once more easily ascertained.

3. The conversion of $\text{coffee} \vee \neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee}$ combines in a disjunction the test $\text{coffee};\text{pass}$ and the test from Equation (11). None of these tests feature a θ branch in their top choice and so the combination is a simple choice between the two:

$$\mathbb{T}_{\mathbb{K}}(\text{coffee} \vee \neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee}) = \text{coffee};\text{pass} \ \Box\ \text{bang};\text{coffee};\text{pass} \tag{12}$$

In what follows we use for brevity $\phi' = \text{coffee} \vee \neg\text{coffee} \wedge \text{bang} \wedge \mathsf{EX}\ \text{coffee}$.

4. We have $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi') = \Sigma\{a;\mathbb{T}_{\mathbb{K}}(\phi') : a \in C\}$ and therefore

$$\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi') = \Sigma\{a;(\text{coffee};\text{pass} \ \Box\ \text{bang};\text{coffee};\text{pass}) : a \in C\} \tag{13}$$

We will actually need in what follows the negation of this formula, which is the following:

$$\mathbb{T}_{\mathbb{K}}(\neg\mathsf{EX}\ \phi') = \Sigma\{a;(\text{coffee};\text{stop} \ \Box\ \text{bang};(\text{coffee};\text{stop} \ \Box\ \theta;\text{pass}) \ \Box\ \theta;\text{pass}) : a \in C\} \ \Box\ \theta;\text{pass} \tag{14}$$

At this point our test becomes complex enough so that we can use it to illustrate in more detail the negation algorithm (Lemma 8). We thus take this opportunity to explain in detail the conversion between $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')$ from Equation (13) and $\overline{\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')} = \mathbb{T}_{\mathbb{K}}(\neg\mathsf{EX}\ \phi')$ shown in Equation (14).

(a) The test $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')$ is shown as an LTS in Figure 6(a). For convenience all the states are labeled t_i, $1 \leq i \leq 7$ so that we can easily refer to them.

(b) We then eliminate all the success (γ) transitions. The states t_3 and t_6 are thus converted from pass to stop.

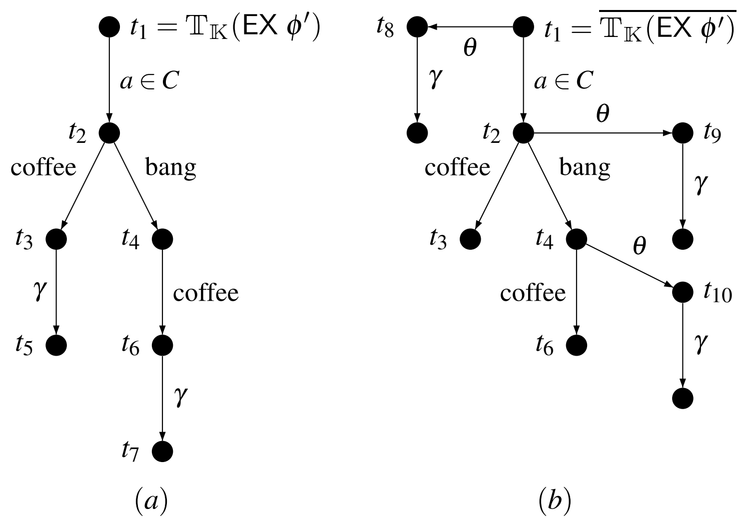


Figure 6: Conversion of the test $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')$ (a) into its negation $\overline{\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')}$ (b).

(c) All the states that were not converted in the previous step (that is, states t_1, t_2, and t_4) gain a $\theta;\text{pass}$ branch. The result is the negation of the original test and is shown in Figure 6(b).

(d) If needed, the conversion the other way around would proceed as follows: The success transitions are eliminated (this affects t_8, t_9, and t_{10}). The preceding θ transitions are also eliminated (which eliminates t_8, t_9, t_{10} and affects the states t_1, t_2, and t_4). The states unaffected by this process are t_3 and t_6 so they both gain a $\theta;\text{pass}$ branch; however, such a branch is the only outgoing one for both t_3 and t_6 so it is equivalent to a simple pass in both cases. The result is precisely the test $\mathbb{T}_{\mathbb{K}}(\mathsf{EX}\ \phi')$ as shown in Figure 6(a).

5. We finally reach the top formula ϕ. Indeed, $\phi = \text{coin} \wedge \mathsf{EX}\ \phi' = \neg(\neg\text{coin} \vee \neg\mathsf{EX}\ \phi')$. We thus need to combine in a disjunction the test $\text{coin};\text{stop} \ \Box\ \theta;\text{pass}$ with the test shown in Equation (14). We have:

$$\begin{aligned}\mathbb{T}_{\mathbb{K}}(\neg\text{coffee} \vee \neg\mathsf{EX}\ \phi') \quad = \quad & \text{coin};(\text{coffee};\text{stop} \\ & \qquad \Box\ \text{bang};(\text{coffee};\text{stop}\ \Box\ \theta;\text{pass}) \\ & \qquad \Box\ \theta;\text{pass}) \\ & \Box\ \Sigma\{b;\text{pass} : b \in C \setminus \{\text{coin}\}\} \\ & \Box\ \theta;\text{pass}\end{aligned}$$

Indeed, $B_1 = \{\text{coin}\}$, $B_2 = C$, $t_1(b) = \text{stop}$, $t_2(b) = \text{coffee};\text{stop}\ \Box\ \text{bang};(\text{coffee};\text{stop}\ \Box\ \theta;\text{pass})\ \Box\ \theta;\text{pass}$, and $t_{N1} = t_{N2} = \text{pass}$.

To reduce the size of the expression we combine the $C \setminus \{\text{coin}\}$ and θ choices and so we obtain:

$$\begin{aligned}\mathbb{T}_{\mathbb{K}}(\neg\text{coffee} \vee \neg\mathsf{EX}\ \phi') \quad = \quad & \text{coin};(\text{coffee};\text{stop} \\ & \qquad \Box\ \text{bang};(\text{coffee};\text{stop}\ \Box\ \theta;\text{pass}) \\ & \qquad \Box\ \theta;\text{pass}) \\ & \Box\ \theta;\text{pass}\end{aligned}$$

Negating the above expression results in the test equivalent to the original formula:

$$\mathbb{T}_{\mathbb{K}}(\phi) \quad = \quad \text{coin};(\text{coffee};\text{pass} \;\Box\; \text{bang};\text{coffee};\text{pass})$$

Recall now that the test we started from in Example 1 was slightly different, namely:

$$t = \text{coin};(\text{coffee};\text{pass} \;\Box\; \theta;\text{bang};\text{coffee};\text{pass})$$

We argue however that these two tests are in this case equivalent. Indeed, both tests succeed whenever coin is followed by coffee. Suppose now that coin does happen but the next action is not coffee. Then t will follow on the deadlock detection branch, which will only succeed if the next action is bang. On the other hand $\mathbb{T}_{\mathbb{K}}(\phi)$ does not have a deadlock detection branch in the choice following coin; however, the only alternative to coffee in $\mathbb{T}_{\mathbb{K}}(\phi)$ is bang, which is precisely the same alternative as for t (as shown above). We thus conclude that t and $\mathbb{T}_{\mathbb{K}}(\phi)$ are indeed equivalent.

Lemma 12. *There exist a function $\mathbb{T}_{\mathbb{X}} : \mathscr{F} \to \mathscr{T}$ such that $\mathbb{X}(p) \models f$ iff p* may *$\mathbb{T}_{\mathbb{X}}(f)$ for any $p \in \mathscr{P}$.*

Proof. The proof established earlier for $\mathbb{T}_{\mathbb{K}}$ (Lemma 11) will also work for $\mathbb{T}_{\mathbb{X}}$. Indeed, the way the operator $\models$ is defined (Definition 4) ensures that all occurrences of Δ are "skipped over" as if they were not there in the first place. However, the paths without the Δ labels are identical to the paths examined in Lemma 11. □

6 Conclusions

Our work creates a constructive equivalence between CTL and failure trace testing. For this purpose we start by offering an equivalence relation between a process or a labeled transition system and a Kripke structure (Section 4.1), then the development of an algorithmic function $\mathbb{K}$ that converts any labeled transition system into an equivalent Kripke structure (Theorem 2).

As already mentioned, the function $\mathbb{K}$ creates a Kripke structure that may have multiple initial states, and so a weaker satisfaction operator (over sets of states rather than states) was needed. It was further noted that this issue only happens for those LTS that start with a choice of multiple visible actions. It follows that in order to eliminate the need for a new satisfaction operator (over sets of states) one can in principle simply create an extra LTS state which becomes the initial state and performs an artificial "start" action to give control to the original initial state. This claim needs however to be verified. Alternatively, one can investigate another equivalence relation (and subsequent conversion function) without this disadvantage. We offer precisely such a relation (Section 4.1) and the corresponding algorithmic conversion function $\mathbb{X}$ (Theorem 3).

There are advantages and disadvantages to both these approaches. The conversion function $\mathbb{K}$ uses an algorithmic conversion between LTS and Kripke structures that results in very compact Kripke structures but introduces the need to modify the model checking algorithm (by requiring a modified notion of satisfaction for CTL formulae). The function $\mathbb{K}$ on the other hand results in considerably larger Kripke structures but does not require any modification of the model checking algorithm.

Once the conversion functions are in place we develop algorithmic functions for the conversion of failure trace tests to equivalent CTL formulae and the other way around, thus showing that CTL and failure trace testing are equivalent (Theorem 4). Furthermore this equivalence holds for both notions of equivalence between LTS and Kripke structure and so our thesis is that failure trace testing and CTL are equivalent under any reasonable equivalence relation between LTS and Kripke structures.

Finally, we note that the straightforward, inductive conversion of failure trace tests into CTL formulae produces in certain cases infinite formulae. We address this issue by showing how failure trace tests containing cycles (which produce the unacceptable infinite formulae) can be converted into compact (and certainly finite) CTL formulae (Theorem 7).

We believe that our results (providing a combined, logical and algebraic method of system verification) have unquestionable advantages. To emphasize this point consider the scenario of a network communication protocol between two end points and through some communication medium being formally specified. The two end points are likely to be algorithmic (or even finite state machines) and so the natural way of specifying them is algebraic. The communication medium on the other hand has a far more loose specification. Indeed, it is likely that not even the actual properties are fully known at specification time, since they can vary widely when the protocol is actually deployed (between say, the properties of a 6-foot direct Ethernet link and the properties of a nondeterministic and congested Internet route between Afghanistan and Zimbabwe). The properties of the communication medium are therefore more suitable to logic specification. Such a scenario is also applicable to systems with components at different levels of maturity (some being fully implemented already while others being at the prototype stage of even not being implemented at all and so less suitable for being specified algebraically). Our work enables precisely this kind of mixed specification. In fact no matter how the system is specified we enable the application of either model checking of model-based testing (or even both) on it, depending on suitability or even personal taste.

The results of this paper are important first steps towards the ambitious goal of a unified (logic and algebraic) approach to conformance testing. We believe in particular that this paper opens several direction of future research.

The issue of tests taking an infinite time to complete is an ever-present issue in model-based testing. Our conversion of CTL formulae is no exception, as the tests resulting from the conversion of expressions that use EF, EG, and EU fall all into this category. Furthermore Rice's theorem [21] (which states that any non-trivial and extensional property of programs is undecidable) guarantees that tests that take an infinite time to complete will continue to exist no matter how much we refine our conversion algorithms. We therefore believe that it is very useful to investigate methods and algorithms for partial (or incremental) application of tests. Such methods will offer increasingly stronger guarantees of correctness as the test progresses, and total correctness at the limit (when the test completes).

It would also be interesting to extend this work to other temporal logics (such as CTL*) and whatever testing framework turns out to be equivalent to it (in the same sense as used in our work).

References

[1] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[2] P. Bellini, R. Mattolini, and P. Nesi, *Temporal logics for real-time system specification*, ACM Computing Surveys, 32 (2000), pp. 12–42.

[3] E. Brinksma, G. Scollo, and C. Steenbergen, *LOTOS specifications, their implementations and their tests*, in IFIP 6.1 Proceedings, 1987, pp. 349–360.

[4] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, eds., *Model-Based Testing of Reactive Systems: Advanced Lectures*, vol. 3472 of Lecture Notes in Computer Science, Springer, 2005.

[5] S. D. Bruda, *Preorder relations*, in Broy et al. [4], pp. 117–149.

[6] S. D. Bruda and C. Dai, *A testing theory for real-time systems*, International Journal of Computers, 4 (2010), pp. 97–106.

[7] S. D. Bruda and Z. Zhang, *Refinement 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, Nov. 2009.

[8] E. M. Clarke and E. A. Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic*, in Works in Logic of Programs, 1982, pp. 52–71.

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla, *Automatic verification of finite state concurrent systems using temporal logic specification*, ACM Transactions on Programming Languages and Systems, 8 (1986), pp. 244–263.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.

[11] R. Cleaveland and G. Lüttgen, *Model checking is refinement—Relating Büchi testing and linear-time temporal logic*, Tech. Rep. 2000-14, ICASE, Langley Research Center, Hampton, VA, Mar. 2000.

[12] C. Dai and S. D. Bruda, *A testing framework for real-time specifications*, in Proceedings of the 9th IASTED International Conference on Software Engineering and Applications (SEA 08), Orlando, Florida, Nov. 2008.

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

[14] R. De Nicola and F. Vaandrager, *Action versus state based logics for transition systems*, Lecture Notes in Computer Science, 469 (1990), pp. 407–419.

[15] ——, *Three logics for branching bisimulation*, Journal of the ACM, 42 (1995), pp. 438–487.

[16] R. W. Floyd, *Assigning meanings to programs*, in Mathematical Aspects of Computer Science, J. T. Schwartz, ed., vol. 19 of Proceedings of Symposia in Applied Mathematics, American Mathematical Society, 1967, pp. 19–32.

[17] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, *Simple on-the-fly automatic verification of linear temporal logic*, in Proceedings of the IFIP symposium on Protocol Specification, Testing and Verification (PSTV 95), Warsaw, Poland, 1995, pp. 3–18.

[18] C. A. R. Hoare, *An axiomatic basis for computer programming*, Communications of the ACM, 12 (1969), pp. 576–580 and 583.

[19] J.-P. Katoen, *Labelled transition systems*, in Broy et al. [4], pp. 615–616.

[20] R. Langerak, *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, 1989, pp. 87–98.

[21] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 2nd ed., 1998.

[22] K. Pawlikowski, *Steady-state simulation of queueing processes: survey of problems and solutions*, ACM Computing Surveys, 22 (1990), pp. 123–170.

[23] A. Pnueli, *A temporal logic of concurrent programs*, Theoretical Computer Science, 13 (1981), pp. 45–60.

[24] J. P. Queille and J. Sifakis, *Fairness and related properties in transition systems — a temporal logic to deal with fairness*, Acta Informatica, 19 (1983), pp. 195–220.

[25] H. Saïdi, *The invariant checker: Automated deductive verification of reactive systems*, in Proceedings of Computer Aided Verification (CAV 97), vol. 1254 of Lecture Notes In Computer Science, Springer, 1997, pp. 436–439.

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

[27] T. J. Schriber, J. Banks, A. F. Seila, I. Ståhl, A. M. Law, and R. G. Born, *Simulation textbooks - old and new, panel*, in Winter Simulation Conference, 2003, pp. 1952–1963.

[28] W. Thomas, *Automata on infinite objects*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. B, North Holland, 1990, pp. 133–191.

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

[30] M. Vardi and P. Wolper, *An automata-theoretic approach to automatic program verification*, in Proceedings of the First Annual Symposium on Logic in Computer Science (LICS 86), 1986, pp. 332–344.

[31] M. Y. Vardi and P. Wolper, *Reasoning about Infinite Computations*, Academic Press, 1994.