More Testable Properties*

Yliès Falcone[1], Jean-Claude Fernandez[2], Thierry Jéron[1], Hervé Marchand[1], and Laurent Mounier[2]
Firstname.Lastname@inria.fr - Firstname.Lastname@imag.fr

[1] INRIA, Rennes - Bretagne Atlantique, France
[2] VERIMAG, Université Grenoble I, France

Abstract. In this paper, we explore the set of testable properties within the *Safety-Progress* classification where testability means to establish by testing that a relation, between the tested system and the property under scrutiny, holds. We characterize testable properties wrt. several relations of interest. For each relation, we give a sufficient condition for a property to be testable. Then, we study and delineate, for each Safety-Progress class, the subset of testable properties and their corresponding test oracle producing verdicts for the possible test executions. Finally, we address automatic test generation for the proposed framework.

1 Introduction

Due to its ability to scale up well and its practical aspect, testing remains one of the most effective and widely used validation technique for software systems. However, due to recent needs in the software industry (for instance in terms of security), it is important to reconsider the classes of requirements this technique allows to validate or invalidate. The aim of a testing stage may be either to find defects or to witness expected behaviors on an implementation under test (IUT). From a practical point of view, a test campaign consists in producing a test suite (*test generation*) from some initial system description, and executing it on the system implementation (*test execution*). The test suite consists in a set of test cases, where each test case is a set of interaction sequences to be executed by an external tester (performed on the points of control and observation, PCOs). Any execution of a test case should lead to a *test verdict*, indicating if the system succeeded or not on this particular test (or if the test was not conclusive).

One way to improve the practical feasibility of a test campaign is to use a property to drive the test execution. In this case, the property is used to generate the so-called test purposes [2, 3] so as to select the most relevant test case behaviors. A property may also represent the desired behavior of the system. In this setting, the property may be a formalization of a security policy describing prohibited behaviors and expectations from the users, as considered in [4, 5]. Several approaches (*e.g.*, [6]) combine classical testing techniques and property verification so as to improve the test activity. Most of these approaches used safety and co-safety properties. A natural question is the existence of other kinds of properties that can be "tested", *i.e.,* to define a precise notion of *testability*.

* An extended version of this paper with complete proofs can be found in [1].

In [7, 8], Nahm, Grabowski, and Hogrefe addressed this issue by discussing the set of temporal properties that can be tested on an implementation. A property is said to be *testable* if it is possible to determine if a given relation (*e.g.*, inclusion) holds between the sequences described by a property and the set of execution sequences that can be produced by interacting with the IUT, after the execution of a finite sequence on the IUT. In their work, testability of properties is studied wrt. the *Safety-Progress* classification ([9] and Section 3) for infinitary properties. The announced classes of testable properties are the safety and guarantee[3] classes. Then, it is not too surprising that most of the previously depicted approaches used safety and co-safety properties during testing.

Context. In this paper, we shall use the same notion of testability. We consider a generic approach, where an underlying property is compared to the possibly infinite execution sequences of the IUT by a tester. This property expresses finite and infinite[4] observable behaviors (which may be desired or not). Usually, IUT's execution sequences are expressed in a different alphabet than the one used to describe the property and have thus to be interpreted. However, testability and the test oracle problem (*i.e.,* the problem of deciding verdicts) can be studied while abstracting this alphabet discrepancy. A second characteristic is that we do not require the existence of an executable specification to generate the test cases. This allows to encompass several conformance testing approaches by viewing the specification as a special property.

Motivations and contributions. The main motivation of this paper is to leverage the use of an extended version of the *Safety-Progress* classification of properties dedicated to runtime techniques. We give a precise characterization of testable properties and provide a formal basis for several previous testing activities. We extend the results of [7] by showing that lots of interesting properties (neither safety nor guarantee) are also testable. Moreover, this framework allows to simply obtain test oracles producing verdicts according to the test execution.

Paper organization. The remainder of this paper is organized as follows. In Section 2, some preliminary concepts and notations are introduced. A quick overview of the *Safety-Progress* classification of properties for runtime validation techniques is given in Section 3. Section 4 introduces the notion of testability considered in this paper. In Section 5, testable properties are characterized. Automatic test generation is addressed in Section 6. Next, in Section 7, we overview the related work and propose a discussion on the results provided by this paper. Finally, Section 8 gives some concluding remarks and raised perspectives.

2 Preliminaries

Given an alphabet of actions Σ, a sequence σ on Σ is a total function $\sigma : I \to \Sigma$ where I is either the interval $[0, n]$ for some $n \in \mathbb{N}$, or $\mathbb{N}$ itself. The empty

[3] In the *Safety-Progress* classification the guarantee class is the co-safety class in the *Safety-Liveness* classification.

[4] The tester observes a finite sequence of the IUT and should state a verdict about all potential continuations of this execution sequence (finite and infinite ones).

sequence is denoted by ϵ. We denote by Σ^* the set of finite sequences over Σ and by Σ^ω the set of infinite sequences over Σ. $\Sigma^* \cup \Sigma^\omega$ is noted Σ^∞. The length (number of elements) of a finite sequence σ is noted $|\sigma|$ and the $(i+1)$-th element of σ is denoted by σ_i. For $\sigma \in \Sigma^*, \sigma' \in \Sigma^\infty$, $\sigma \cdot \sigma'$ is the concatenation of σ and σ'. The sequence $\sigma \in \Sigma^*$ is a *strict prefix* of $\sigma' \in \Sigma^\infty$ (equivalently σ' is a *strict continuation* of σ), noted $\sigma \prec \sigma'$, when $\forall i \in [0, |\sigma| - 1] : \sigma_i = \sigma'_i$ and $|\sigma| < |\sigma'|$. When $\sigma' \in \Sigma^*$, we note $\sigma \preceq \sigma' \stackrel{\text{def}}{=} \sigma \prec \sigma' \vee \sigma = \sigma'$. For $\sigma \in \Sigma^\infty$ and $n \in \mathbb{N}$, $\sigma_{\dots n}$ is the sub-sequence containing the $n+1$ first elements of σ. The set of prefixes of $\sigma \in \Sigma^\infty$ is $\mathit{pref}(\sigma) \stackrel{\text{def}}{=} \{\sigma' \in \Sigma^* \mid \sigma' \preceq \sigma\}$. For a finite sequence $\sigma \in \Sigma^*$, the set of finite continuations is $\mathit{cont}^*(\sigma) \stackrel{\text{def}}{=} \{\sigma' \in \Sigma^* \mid \exists \sigma'' \in \Sigma^* : \sigma' = \sigma \cdot \sigma''\}$.

The IUT is a program $\mathcal{P}$ abstracted as a generator of execution sequences. We are interested in a restricted set of operations that influence the truth value of tested properties and are made on PCOs. We abstract these operations by an alphabet Σ. We denote by $\mathcal{P}_\Sigma$ a program with alphabet Σ. The set of execution sequences of $\mathcal{P}_\Sigma$ is denoted by $\mathit{Exec}(\mathcal{P}_\Sigma) \subseteq \Sigma^\infty$. This set is *prefix-closed*, that is $\forall \sigma \in \mathit{Exec}(\mathcal{P}_\Sigma) : \mathit{pref}(\sigma) \subseteq \mathit{Exec}(\mathcal{P}_\Sigma)$. We will use $\mathit{Exec}_{\mathrm{f}}(\mathcal{P}_\Sigma)$ (resp. $\mathit{Exec}_\omega(\mathcal{P}_\Sigma)$) to refer to the finite (resp. infinite) execution sequences of $\mathcal{P}_\Sigma$, that is $\mathit{Exec}_{\mathrm{f}}(\mathcal{P}_\Sigma) \stackrel{\text{def}}{=} \mathit{Exec}(\mathcal{P}_\Sigma) \cap \Sigma^*$ and $\mathit{Exec}_\omega(\mathcal{P}_\Sigma) \stackrel{\text{def}}{=} \mathit{Exec}(\mathcal{P}_\Sigma) \cap \Sigma^\omega$.

Properties as sets of execution sequences. A *finitary property* (resp. an *infinitary property*, a *property*) is a subset of execution sequences of Σ^* (resp. Σ^ω, Σ^∞). Given a finite (resp. infinite) execution sequence σ and a property ϕ (resp. φ), we say that σ *satisfies* ϕ (resp. φ) when $\sigma \in \phi$, noted $\phi(\sigma)$ (resp. $\sigma \in \varphi$, noted $\varphi(\sigma)$). A consequence of this definition is that properties we will consider are restricted to *linear time* execution sequences, excluding specific properties defined on powersets of execution sequences and branching properties.

Runtime properties [10]. Runtime properties should characterize satisfaction for both kinds of sequences (finite and infinite) in a uniform way. To do so, we define *r-properties* as pairs $\Pi = (\phi, \varphi) \subseteq \Sigma^* \times \Sigma^\omega$. We say that $\sigma \in \mathit{Exec}(\mathcal{P}_\Sigma)$ satisfies (ϕ, φ) (noted $\Pi(\sigma)$) when $\sigma \in \Sigma^* \wedge \phi(\sigma) \vee \sigma \in \Sigma^\omega \wedge \varphi(\sigma)$. The definition of the negation of an *r-property* follows from definition of the negation for finitary and infinitary properties. Boolean combinations of *r-properties* are defined in a natural way. For $* \in \{\vee, \wedge\}$, $(\phi_1, \varphi_1) * (\phi_2, \varphi_2) \stackrel{\text{def}}{=} (\phi_1 * \phi_2, \varphi_1 * \varphi_2)$.

An *r-property* $\Pi \subseteq \Sigma^* \times \Sigma^\omega$ is said to be negatively (resp. positively) determined [11] by $\sigma \in \Sigma^*$ if $\neg\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty : \neg\Pi(\sigma \cdot \mu)$ (resp. $\Pi(\sigma) \wedge \forall \mu \in \Sigma^\infty : \Pi(\sigma \cdot \mu)$), denoted $\ominus$–*determined*(σ, Π) (resp. $\oplus$–*determined*(σ, Π)).

3 A Safety-Progress classification for runtime techniques

The *Safety-Progress* (SP) classification of properties [12, 9] introduced a hierarchy between regular (linear time) properties[5] defined as sets of *infinite* execution sequences. In [10], we extended the classification to deal also with finite-length execution sequences by revisiting it using runtime properties (*r-properties*). The *Safety-Progress* classification is an alternative to the classical *Safety-Liveness* [13,

[5] In the remainder of this paper, the term property will stand for regular property.

14] dichotomy. Unlike this later, the *Safety-Progress* classification is a hierarchy and not a partition, and provides a finer-grain classification of properties in a uniform way according to 4 views [15]: a language-theoretic view (seeing properties as sets of sequences), a logical view (seeing properties as LTL formulas), a topological view (seeing properties as open or closed sets), and an automata view (seeing properties as accepted words of Streett automata [16]).

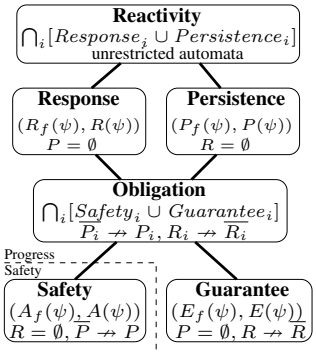


Fig. 1: SP classification

A graphical representation of the *Safety-Progress* classification of properties is depicted in Fig. 1. Further details and results can be found in [17]. Here, we consider only the language and the automata views.

The language-theoretic view of r-properties. The language-theoretic view of the SP classification is based on the construction of infinitary properties and finitary properties from finitary ones. It relies on the use of four operators A, E, R, P (building infinitary properties) and four operators A_f, E_f, R_f, P_f (building finitary properties) applied to finitary properties. Formal definitions can be found in [17]. In the following ψ is a finitary property.

$A(\psi)$ consists of all infinite words σ s.t. *all* prefixes of σ belong to ψ. $E(\psi)$ consists of all infinite words σ s.t. *some* prefixes of σ belong to ψ. $R(\psi)$ consists of all infinite words σ s.t. *infinitely many* prefixes of σ belong to ψ. $P(\psi)$ consists of all infinite words σ s.t. *all but finitely many* prefixes of σ belong to ψ.

$A_f(\psi)$ consists of all finite words σ s.t. *all* prefixes of σ belong to ψ. One can observe that $A_f(\psi)$ is the largest prefix-closed subset of ψ. $E_f(\psi)$ consists of all finite words σ s.t. *some* prefixes of σ belong to ψ. One can observe that $E_f(\psi) = \psi \cdot \Sigma^*$. $R_f(\psi)$ consists of all finite words σ s.t. $\psi(\sigma)$ and there exists an infinite number of continuations σ' of σ also belonging to ψ. $P_f(\psi)$ consists of all finite words σ belonging to ψ s.t. there exists a continuation σ' of σ s.t. σ' persistently has continuations staying in ψ (*i.e.,* σ'' s.t. $\sigma' \cdot \sigma''$ belongs to ψ).

The automata view of r-properties [10]. We define a variant of deterministic and complete Streett automata (introduced in [16] and used in [15]). We add to original Streett automata an acceptance condition for finite sequences in such a way that these automata uniformly recognize *r-properties*.

Definition 1 (Streett automaton). *A deterministic Streett automaton $\mathcal{A}$ is a tuple $(Q^{\mathcal{A}}, q^{\mathcal{A}}_{\text{init}}, \Sigma, \longrightarrow_{\mathcal{A}}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$. The set $Q^{\mathcal{A}}$ is the set of states, $q^{\mathcal{A}}_{\text{init}} \in Q^{\mathcal{A}}$ is the initial state. $\longrightarrow_{\mathcal{A}}$: $Q^{\mathcal{A}} \times \Sigma \to Q^{\mathcal{A}}$ is the (complete) transition function. $\{(R_1, P_1), \ldots, (R_m, P_m)\}$ is the set of accepting pairs, for all $i \leq n$, $R_i \subseteq Q^{\mathcal{A}}$ and $P_i \subseteq Q^{\mathcal{A}}$ are the sets of recurrent and persistent states.*

We refer to an automaton with m accepting pairs as an m-automaton. A *plain*-automaton is a 1-automaton, and we refer to R_1 and P_1 as R and P. Moreover, for $\sigma = \sigma_0 \cdots \sigma_{n-1} \in \Sigma^*$ and $q, q' \in Q^{\mathcal{A}}$, we note $q \xrightarrow{\sigma} q'$ when $\exists q_1, \ldots, q_{n-2} \in Q^{\mathcal{A}} : q \xrightarrow{\sigma_0} q_1 \wedge \ldots \wedge q_{n-2} \xrightarrow{\sigma_{n-2}} q'$. For $q \in Q^{\mathcal{A}}$, $Reach_{\mathcal{A}}(q) = \{q' \in Q^{\mathcal{A}} \mid \exists \sigma \in \Sigma^* \setminus \{\epsilon\} : q \xrightarrow{\sigma}_{\mathcal{A}} q'\} \cup \{q\}$ is the set of reachable states from q. For

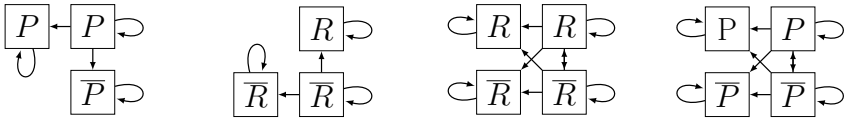


Fig. 2: Schematic illustrations of the shapes of Streett automata for basic classes

$\sigma \in \Sigma^\infty$, the *run* of σ on $\mathcal{A}$ is the sequence of states involved by the execution of σ on $\mathcal{A}$. It is formally defined as $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdots$ where $\forall i : (q_i \in Q^{\mathcal{A}} \cap Reach_{\mathcal{A}}(q^{\mathcal{A}}_{\text{init}}) \wedge q_i \xrightarrow{\sigma_i}_{\mathcal{A}} q_{i+1}) \wedge q_0 = q^{\mathcal{A}}_{\text{init}}$. For an execution sequence $\sigma \in \Sigma^\omega$ on a Streett automaton $\mathcal{A}$, we define $vinf(\sigma, \mathcal{A})$ as the set of states appearing infinitely often in $run(\sigma, \mathcal{A})$.

Definition 2 (Acceptance conditions). *For $\sigma \in \Sigma^\omega$, $\mathcal{A}$ accepts σ if $\forall i \in [1, m] : vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$. For $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $\mathcal{A}$ accepts σ if $(\exists q_0, \ldots, q_{n-1} \in Q^{\mathcal{A}} : run(\sigma, \mathcal{A}) = q_0 \cdots q_{n-1} \wedge q_0 = q^{\mathcal{A}}_{\text{init}}$ and $\forall i \in [1, m] : q_{n-1} \in P_i \cup R_i)$. $\mathcal{A}$ defines an r-property $(\phi, \varphi) \in 2^{\Sigma^* \times \Sigma^\omega}$ iff the set of finite (resp. infinite) sequences accepted by $\mathcal{A}$ is equal to ϕ (resp. φ).*

The hierarchy of r-properties. The hierarchical organization of *r-properties* can be seen in the language view using the operators and in the automata view using syntactic restrictions on Streett automata (illustrated in Fig. 2 for basic classes).

Definition 3 (Safety-Progress classes). *An r-property Π defined by $(Q^{\mathcal{A}_\Pi}, q^{\mathcal{A}_\Pi}_{\text{init}}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$, Π is said to be*

- *A* safety *r-property if $\Pi = (A_f(\psi), A(\psi))$ for some $\psi \subseteq \Sigma^*$ or equivalently $\mathcal{A}_\Pi$ is a plain-automaton s.t. $R = \emptyset$ and there is no transition from $\overline{P}$ to P.*
- *A* guarantee *r-property if $\Pi = (E_f(\psi), E(\psi))$ for some $\psi \subseteq \Sigma^*$ or equivalently $\mathcal{A}_\Pi$ is a plain-automaton s.t. $P = \emptyset$ and there is no transition from R to $\overline{R}$.*
- *An m-*obligation *r-property if $\Pi = \bigcap_{i=1}^m (S_i(\psi_i) \cup G_i(\psi'_i))$ or $\Pi = \bigcup_{i=1}^m (S_i(\psi_i) \cap G_i(\psi'_i))$ where $S(\psi_i)$ (resp. $G(\psi'_i)$) are safety (resp. guarantee) r-properties defined over the ψ_i and the ψ'_i; or equivalently $\mathcal{A}_\Pi$ is an m-automaton s.t. for $i \in [1, m]$ there is no transition from $\overline{P_i}$ to P_i and from R_i to $\overline{R_i}$.*
- *A* response *r-property if $\Pi = (R_f(\psi), R(\psi))$ for some $\psi \subseteq \Sigma^*$ or equivalently $\mathcal{A}_\Pi$ is a plain-automaton s.t. $P = \emptyset$.*
- *A* persistence *r-property if $\Pi = (P_f(\psi), P(\psi))$ for some $\psi \subseteq \Sigma^*$ or equivalently $\mathcal{A}_\Pi$ is a plain-automaton s.t. $R = \emptyset$.*
- *A* reactivity *r-property if Π is obtained by finite boolean combinations of response and persistence r-properties or equivalently $\mathcal{A}_\Pi$ is unrestricted.*

An r-property of a given class is pure *when not belonging to any other sub-class.*

Example 1 (r-properties). Let us consider $\Sigma_1 = \{a, b, c\}$ and $\psi_1 = a^* \cdot \big(b^* + c \cdot (c + a)^* \cdot b^+\big)$ defined by the deterministic finite-state automaton (DFA) in Fig. 3a with accepting states $1, 2$. The Streett automaton in Fig. 3b defines $(A_f(\psi_1), A(\psi_1))$. Let $\Sigma_2 = \{a, b\}$, and the finitary property $\psi_2 = (a \cdot b)^+$ recognized by the DFA depicted in Fig. 4a. The Streett automaton in Fig. 4b (resp. Fig. 4c) represents the guarantee (resp. response) *r-property* built upon ψ_2.

4 Some notions of testability

From its *finite* interaction with the underlying IUT, the tester produces a sequence of events in Σ^*. We study the conditions for a tester, using the produced sequence of events, to determine whether a given relation holds between the set of *all* (finite and infinite) execution sequences that can be produced by the IUT ($Exec(\mathcal{P}_\Sigma)$), and the set of sequences described by the *r-property* Π. Roughly speaking, the challenge addressed by a tester is thus to determine a verdict between Π and $Exec(\mathcal{P}_\Sigma)$, from a finite sequence extracted from $Exec_{\mathrm{f}}(\mathcal{P}_\Sigma)$[6].

Let us recall that the *r-property* is a pair made of two sets: a set of finite sequences and a set of infinite sequences. In the sequel, we shall compare this pair to the set of execution sequences of the IUT which is a set constituted of finite and infinite sequences. As noticed in [7], one may consider several possible relations between the execution sequences produced by the program and those described by the property. Those relations are recalled here in the context of *r-properties*. In [1], further relations are studied.

Definition 4 (Relations between IUT sequences and an *r-property* [7]). *The possible relations of interest between $Exec(\mathcal{P}_\Sigma)$ and Π are:*

- $Exec_{\mathrm{f}}(\mathcal{P}_\Sigma) \subseteq \Pi \cap \Sigma^*$ *and* $Exec_\omega(\mathcal{P}_\Sigma) \subseteq \Pi \cap \Sigma^\omega$ *(noted* $Exec(\mathcal{P}_\Sigma) \subseteq \Pi$*).*
- $Exec_{\mathrm{f}}(\mathcal{P}_\Sigma) \cap (\Pi \cap \Sigma^*) \neq \emptyset$ *and* $Exec_\omega(\mathcal{P}_\Sigma) \cap (\Pi \cap \Sigma^\omega) \neq \emptyset$ *(noted* $Exec(\mathcal{P}_\Sigma) \cap \Pi \neq \emptyset$*).*

The test verdict is thus determined according to the conclusions that one can obtain for the considered relation. In essence, a tester can and must only determine a verdict from a *finite interaction* $\sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma)$. In Section 5, we will study the conditions to state weaker verdicts on a single execution sequence.

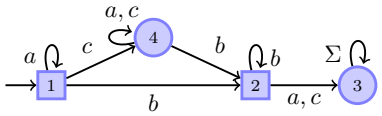
Definition 5 (Verdicts [7]). *Given a relation $\mathcal{R}$ between $Exec(\mathcal{P}_\Sigma)$ and Π and a test execution σ, the tester produces verdicts as follows:*

- *pass if σ allows to determine that $\mathcal{R}$ holds;*
- *fail if σ allows to determine that $\mathcal{R}$ does not hold;*
- *unknown otherwise.*

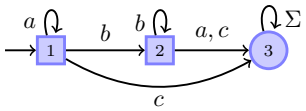
We note $verdict(\sigma, \mathcal{R}(Exec(\mathcal{P}_\Sigma), \Pi))$ the verdict that the observation of σ allows to determine. Let us remark the two following practical problems:

- In general, the IUT may be a program exhibiting infinite-length execution sequences. Obviously these sequences cannot be evaluated by a tester wrt. Π.

[6] Or from a finite set of finite sequences, as a straightforward extension.

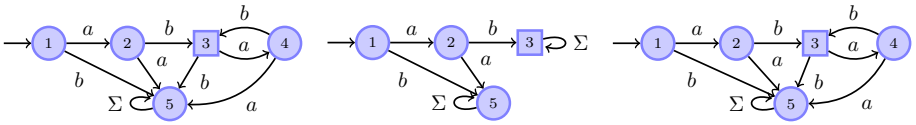


(a) $\psi_1 = a^* \cdot \left(b^* + c \cdot (c + a)^* \cdot b^+\right)$



(b) Streett safety, $P = \{1, 2\}$

Fig. 3: DFA for ψ_1 and Streett for $(A_f(\psi_1), A(\psi_1))$



(a) $\psi_2 = (a \cdot b)^+$ (b) Streett guarantee (c) Streett response

Fig. 4: DFA for ψ_2 and Streett for $(E_f(\psi_2), E(\psi_2))$, $(R_f(\psi_2), R(\psi_2))$, $R = \{3\}$

– Moreover, finite execution sequences contained in the *r-property* cannot be processed easily. For instance, if the test execution exhibits a sequence $\sigma \notin \Pi$, deciding to stop the test is a critical issue. Actually, nothing allows to claim that a continuation of the test execution would not exhibit a new sequence belonging to the *r-property*, *i.e.*, $\sigma' \in \Sigma^\infty$ s.t. $\sigma \cdot \sigma' \in \Pi$.

Thus, the test should be stopped only when there is no doubt regarding the verdict to be established. Following [7], we propose a notion of testability, that takes into account the aforementioned practical limitations, and that is set in the context of the Safety-Progress classification. We suppose the existence of a tester that can interpret the execution sequences with the IUT $\mathcal{P}_\Sigma$ on $Exec_{\mathrm{f}}(\mathcal{P}_\Sigma)$.

Definition 6 (Testability). *An r-property Π is said to be* testable *on $\mathcal{P}_\Sigma$ wrt. the relation $\mathcal{R}$ if there exists an execution sequence $\sigma \in \Sigma^*$ s.t.:*

$$\sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma) \Rightarrow verdict(\sigma, \mathcal{R}(Exec(\mathcal{P}_\Sigma), \Pi)) \in \{pass, fail\}$$

Intuitively, this condition compels the existence of a sequence which, if played on the IUT, allows to determine for sure, whether the relation holds or not. Let us note that this definition entails to synthesize a test oracle which allows to determine $\mathcal{R}\big(Exec(\mathcal{P}_\Sigma), \Pi\big)$ from the observation of a sequence $\sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma)$.

A test oracle is a finite state machine (FSM) parametrized by a test relation as shown in Definition 4. It reads incrementally an interaction sequence $\sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma)$ and produces verdicts in $\{pass, fail, unknown\}$.

Definition 7 (Test Oracle). *A* test oracle *$\mathcal{O}$ for an IUT $\mathcal{P}_\Sigma$, a relation $\mathcal{R}$ and an r-property Π is a 4-tuple $(Q^{\mathcal{O}}, q^{\mathcal{O}}_{\mathrm{init}}, \longrightarrow_{\mathcal{O}}, \Gamma^{\mathcal{O}})$. The finite set $Q^{\mathcal{O}}$ denotes the control states and $q^{\mathcal{O}}_{\mathrm{init}} \in Q^{\mathcal{O}}$ is the initial state. The complete function $\longrightarrow_{\mathcal{O}}$: $Q^{\mathcal{O}} \times \Sigma \to Q^{\mathcal{O}}$ is the transition function. The output function $\Gamma^{\mathcal{O}} : Q^{\mathcal{O}} \to \{pass, fail, unknown\}$ produces verdicts with the following constraints:*

– *all states emitting a pass or a fail verdict are final (sink states),*
– $\exists \sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma) : q^{\mathcal{O}}_{\mathrm{init}} \xrightarrow{\sigma}_{\mathcal{O}} q \wedge \Gamma(q) = pass \Rightarrow \mathcal{R}(Exec(\mathcal{P}_\Sigma), \Pi)$,
– $\exists \sigma \in Exec_{\mathrm{f}}(\mathcal{P}_\Sigma) : q^{\mathcal{O}}_{\mathrm{init}} \xrightarrow{\sigma}_{\mathcal{O}} q \wedge \Gamma(q) = fail \Rightarrow \neg\mathcal{R}(Exec(\mathcal{P}_\Sigma), \Pi)$.

5 Testable properties without executable specification

The framework of *r-properties* (Section 3) allows to determine the testability of the different classes of properties using positive and negative determinacy. Moreover, this framework provides a computable oracle, which is a sufficient condition for testing. Furthermore, we will be able to characterize which test sequences allow to establish sought verdicts. Then, we will determine which verdict has to be produced in accordance with the played test sequence.

In this paper, we focus on the relation $Exec(\mathcal{P}_\Sigma) \subseteq \Pi$. Characterizations for the relation $Exec(\mathcal{P}_\Sigma) \cap \Pi \neq \emptyset$ (by duality) and others relations are in [1].

Obtainable verdicts and sufficient conditions. For this relation, the unique verdicts that may be produced are *fail* and *unknown*. We explicit this below.

A *pass* verdict means that all execution sequences of $\mathcal{P}_\Sigma$ belong to Π. The unique case where it is possible to establish a *pass* verdict is in the trivial case where $\Pi = (\Sigma^*, \Sigma^\omega)$, *i.e.,* the *r-property* Π is always verified. Obviously, every implementation with alphabet Σ satisfies this relation. In other cases, it is impossible to obtain such a verdict (whatever is the property class under consideration), since the whole set $\mathcal{P}_\Sigma$ is usually unknown from the tester. In Section 5, we will study the conditions under which it is possible to state *weak pass* verdicts, when reasoning on a *single* execution sequence of the IUT.

A *fail* verdict means that there exists some sequences of the program which are not in Π. In order to produce this verdict, a sufficient condition is to exhibit an execution sequence of $\mathcal{P}_\Sigma$ s.t. Π is *negatively determined* by this sequence:

$$\exists \sigma \in \mathit{Exec}_{\mathrm{f}}(\mathcal{P}_\Sigma) : \ominus\text{–}\mathit{determined}(\sigma, \Pi) \;\Rightarrow\; \mathit{verdict}(\sigma, \mathit{Exec}(\mathcal{P}_\Sigma) \subseteq \Pi) = \mathit{fail}$$

Testability of this relation in the Safety-Progress classification. For each SP class, we state the conditions under which the properties of this class are testable.

Theorem 1 (Testability of $\mathit{Exec}(\mathcal{P}_\Sigma) \subseteq \Pi$). *For $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\mathrm{init}}^{\mathcal{A}_\Pi}, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ recognizing an r-property Π, according to the class of Π, the testability conditions expressed both in the language-theoretic and automata views are given in Table 1.*

Verdicts to deliver. We now state the verdicts that should be produced by a tester for the possibly infinite sequences of the IUT. Each testability condition in the language view is in the form $f(\{\psi_i\}_i) \neq \emptyset$ where the $\psi_i \subseteq \Sigma^*$ ($i \in [1, n]$) are used to build the *r-property* and f is a composition of set operations on ψ_i. When $\sigma \in \mathit{Exec}_{\mathrm{f}}(\mathcal{P}_\Sigma) \cap f(\{\psi_i\}_i)$, the test oracle should deliver *fail* since the underlying *r-property* is negatively determined. Conversely, when $\sigma \in \mathit{Exec}_{\mathrm{f}}(\mathcal{P}_\Sigma) \setminus f(\{\psi_i\}_i)$, the test oracle can deliver *unknown*. In practice, those verdicts are determined by a computable function, reading an interaction sequence, *i.e.,* a test oracle. In our framework, the test oracle is obtained from a Streett automaton[7]:

Property 1 (Test oracle for the relation $\mathit{Exec}(\mathcal{P}_\Sigma) \subseteq \Pi$). Given $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q_{\mathrm{init}}^{\mathcal{A}_\Pi}, \Sigma, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ defining Π, the test oracle $(Q^{\mathcal{O}}, q_{\mathrm{init}}^{\mathcal{O}}, \longrightarrow_{\mathcal{O}}, \Gamma^{\mathcal{O}})$ for the relation $\mathit{Exec}(\mathcal{P}_\Sigma) \subseteq \Pi$ is defined as follows. $Q^{\mathcal{O}}$ is the smallest subset of $Q^{\mathcal{A}_\Pi}$, reachable from $q_{\mathrm{init}}^{\mathcal{O}}$ by $\longrightarrow_{\mathcal{O}}$ (defined below) with $q_{\mathrm{init}}^{\mathcal{O}} = q_{\mathrm{init}}^{\mathcal{A}_\Pi}$.

- $\Gamma^{\mathcal{O}}$ is defined as follows:
  - If Π is a pure safety, guarantee, obligation, or response property $\Gamma^{\mathcal{O}}(q) =$ *fail* if $q \in \bigcup_{i=1}^{k} (\overline{P_i} \cap \{q \in \overline{R_i} \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \subseteq \overline{R_i}\}$ and *unknown* otherwise,
  - If Π is a pure persistence property $\Gamma^{\mathcal{O}}(q) = \mathit{fail}$ if $q \in \{q \in \overline{P} \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \subseteq \overline{P}\}$ and *unknown* otherwise;
- $\longrightarrow_{\mathcal{O}}$ is defined as the smallest relation verifying:
  - $q \xrightarrow{e}_{\mathcal{O}} q$ if $\exists e \in \Sigma, \exists q' \in Q^{\mathcal{O}} : q \xrightarrow{e}_{\mathcal{A}_\Pi} q'$ and $\Gamma^{\mathcal{O}}(q) = \mathit{fail}$,
  - $\longrightarrow_{\mathcal{O}} = \longrightarrow_{\mathcal{A}_\Pi}$ otherwise.

$Exec(\mathcal{P}_{\mathbf{\Sigma}}) \subseteq \mathbf{\Pi}$	Testability Condition (language view)	Testability Condition (automata view)
Safety $(A_f(\psi), A(\psi))\ \|R = \emptyset, \overline{P} \nrightarrow P$	$\overline{\psi} \neq \emptyset$	$\overline{P} \neq \emptyset$
Guarantee $(E_f(\psi), E(\psi))\|P = \emptyset, R \nrightarrow \overline{R}$	$\{\sigma \in \overline{\psi} \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \overline{\psi}\} \neq \emptyset$	$\{q \in \overline{R} \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \overline{R}\} \neq \emptyset$
Obligation $\bigcap_{i=1}^{k}(S_i(\psi_i) \cup G_i(\psi_i'))$ $\bigcup_{i=1}^{k}(S_i(\psi_i) \cap G_i(\psi_i'))$ $\overline{P_i} \nrightarrow P_i, R_i \nrightarrow \overline{R_i}$	$\bigcup_{i=1}^{k}(\overline{\psi_i} \cap \{\sigma \in \overline{\psi_i'} \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \overline{\psi_i'}\}) \neq \emptyset$ $\bigcap_{i=1}^{k}(\overline{\psi_i} \cup \{\sigma \in \overline{\psi_i'} \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \overline{\psi_i}\}) \neq \emptyset$	$\bigcup_{i=1}^{k}(\overline{P_i} \cap \{q \in \overline{R_i} \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \overline{R_i}\}) \neq \emptyset$
Response $(R_f(\psi), R(\psi))\ \|P = \emptyset$	$\{\sigma \in \overline{\psi} \mid cont^*(\sigma) \subseteq \overline{\psi}\} \neq \emptyset$	$\{q \in \overline{R} \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \overline{R}\} \neq \emptyset$
Persistence $(P_f(\psi), P(\psi))\ \|R = \emptyset$	$\{\sigma \in \overline{\psi} \mid cont^*(\sigma) \subseteq \overline{\psi}\} \neq \emptyset$	$\{q \in \overline{P} \mid Reach_{\mathcal{A}_\Pi}(q) \subseteq \overline{P}\} \neq \emptyset$

Table 1: Summary of testability results wrt. the relation $Exec(\mathcal{P}_\Sigma) \subseteq \Pi$

The proof of this property follows from Theorem 1 and Definition 7.

Example 2 (Testability of some r-properties wrt. $Exec(\mathcal{P}_\Sigma) \subseteq \Pi$). We present the testability of three r-properties introduced in Example 1. The safety r-property Π_1 is testable wrt. the relation $Exec(\mathcal{P}_{\Sigma_1}) \subseteq \Pi_1$. Indeed in the language view, there are sequences belonging to $\overline{\psi_1}$ (the corresponding DFA has a non accepting state). In the automata view, we have sink $\in \overline{P}$ (reachable from the initial state). The guarantee r-property Π_2 is testable wrt. the relation $Exec(\mathcal{P}_{\Sigma_2}) \subseteq \Pi_2$. Indeed, there are sequences belonging to $\overline{\psi_2}$ s.t. all prefixes of theses sequences and all its continuations are also in $\overline{\psi_2}$. In the automata view, there is a (reachable) state in $\overline{R}$ from which all reachable states are in $\overline{R}$. The response r-property Π_3 is testable wrt. the relation $Exec(\mathcal{P}_{\Sigma_2}) \subseteq \Pi_3$. Indeed, there are sequences belonging to $\overline{\psi_2}$ s.t. all continuations of these sequences belong to $\overline{\psi_2}$. In the automata view, there is a (reachable) state in $\overline{R}$ from which all reachable states are in $\overline{R}$.

Thus, we have clarified and extended some results of [7]. First, we have shown that the safety *r-property* $(\Sigma^*, \Sigma^\omega)$ always lead to a *pass* verdict and is vacuously testable. Moreover, we exhibited some *r-properties* of other classes which are testable, *i.e.,* some obligation, response, and persistence *r-properties*.

Refining verdicts. Similarly to the introduction of weak truth values in runtime verification [18, 10, 17], it is possible to introduce *weak* verdicts in testing. In this respect, stopping the test and producing a weak verdict consists in stating that the test interaction sequence produced so far belongs (or not) to the property. The idea of satisfaction "if the program stops here" in runtime verification [18, 10] corresponds to the idea of "the test has shown enough on the implementation" in testing. In this case, testing would be similar to a kind of "active runtime verification": one is interested in the satisfaction of one execution of the program which is steered externally by a tester. Basically, it amounts to not seeing testing as a destructive activity, but as a way to enhance confidence in the implementation compliance wrt. a property.

Under some conditions, it is possible to determine *weak verdicts* for some classes of properties in the following sense: the verdict is expressed on *one single execution sequence* σ, and it does not afford any conclusion on the set $Exec(\mathcal{P}_\Sigma)$.

[7] The test oracle can be also obtained from the *r-properties* described in others views (language, logic). Indeed, in [17] we describe how to express an *r-property* in the automata view from its expression in the language or the logic view.

We have seen that, for $Exec(\mathcal{P}_\Sigma) \subseteq \Pi$, the only verdicts that can be produced were *fail* and *unknown*. Clearly, *fail* verdicts can still be produced. Furthermore, *unknown* verdicts can be refined into weak *pass* verdicts when the sequence σ *positively determines* the *r-property*. In this case, the test can be stopped since whatever is the future behavior of the IUT, it will exhibit behaviors that will satisfy the *r-property*. In this case, it seems reasonable to produce a weak pass verdict and consider new test executions in order to gain in confidence.

We revisit, for each *Safety-Progress* class, the situations when weak *pass* verdicts can be produced for this relation.

For safety *r-properties*. Let Π be a safety *r-property*, then there exists $\psi \subseteq \Sigma^*$ s.t. Π can be expressed $(A_f(\psi), A(\psi))$. When the produced sequence belongs to $\{\sigma \in \psi \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \psi\}$, the tester can produce a weak *pass* verdict.

For guarantee *r-properties*. Let Π be a guarantee *r-property*, then there exists $\psi \subseteq \Sigma^*$ s.t. Π can be expressed $(E_f(\psi), E(\psi))$. It is possible to produce a weak *pass* verdict if the set ψ is not empty: guarantee *r-properties* are always positively determined when they are satisfied.

For obligation *r-properties*. Let Π be an m-obligation *r-property*.

- If for $m \in \mathbb{N}^*$, Π is expressed $\bigcap_{i=1}^{m}(S_i(\psi_i) \cup G_i(\psi'_i))$ where $S_i(\psi_i)$ (resp. $G_i(\psi'_i)$) is a safety (resp. guarantee) *r-property* built upon ψ_i (resp. ψ'_i), $i \in [1, m]$. The tester can produce a weak *pass* verdict when the interaction sequence belongs to $\bigcap_{i=1}^{m} \psi'_i$.
- If for $m \in \mathbb{N}^*$, Π is expressed $\bigcup_{i=1}^{m}(S_i(\psi_i) \cap G_i(\psi'_i))$ where $S_i(\psi_i)$ (resp. $G_i(\psi'_i)$) is a safety (resp. guarantee) *r-property* built upon ψ_i (resp. ψ'_i), $i \in [1, m]$. The tester can produce a weak *pass* verdict when the interaction sequence produced by the program belongs to $\bigcup_{i=1}^{m}(\{\sigma \in \psi_i \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \psi_i\} \cap \psi'_i)$.

For response and persistence *r-properties*. The reasoning is similar to the one used for safety *r-properties*. Let Π be a response (resp. persistence) *r-property*, then there exists $\psi \subseteq \Sigma^*$ s.t. Π can be expressed $(R_f(\psi), R(\psi))$ (resp. $(P_f(\psi), P(\psi))$). When the interaction sequence belongs to $\{\sigma \in \psi \mid pref(\sigma) \cup cont^*(\sigma) \subseteq \psi\}$, the tester can produce a weak *pass* verdict.

6 Automatic test generation

In this section, we address test generation for the testing framework introduced in this paper. Here, test generation is based on *r-properties*, and the purpose of the test campaign is to detect verdicts for a relation between an *r-property* and an IUT. Before entering into the details of test generation, we first discuss informally some practical constraints that have to be taken into account for test generation. After that, we are able to compute the canonical tester, discuss test selection, and show how quiescence can be taken into account in our framework.

Which sequences should be played? The sequences of interest to play on the IUT are naturally those leading to a *fail* or a *weak pass* verdict and these can be used to generate test cases. In the language view (resp. automata view), these sequences are those belonging to the exhibited sets (resp. leading to the exhibited set of states) in testability conditions. For instance, for a safety *r-property* $\Pi_S = (A_f(\psi), A(\psi))$ built upon ψ, and defined by a safety automaton $\mathcal{A}_{\Pi_S}$, one should play sequences in $\overline{\psi}$ or equivalently those leading to $\overline{P}$ in $\mathcal{A}_{\Pi_S}$.

When to stop the test? When the tested program produces an execution sequence $\sigma \in \Sigma^*$, a raised question is when to safely stop the test. Obviously, a first answer is when a *fail* or *weak pass* verdict has been issued since this verdict is definitive. Although in other cases, when the test interactions produced some test sequences leading so far to *unknown* evaluations, the question prevails. It remains to the tester appraisal to decide when the test should be stopped (see Section 6.2).

Vocabularies and test architecture. In order to address test generation, we will need to distinguish inputs and outputs and the vocabularies of the IUT and the *r-property*. The alphabet Σ of the property is now partitioned into $\Sigma_?$ (input actions) and $\Sigma_!$ (output actions). The alphabet of the IUT becomes Σ^{IUT} and is partitioned into $\Sigma^{IUT}_?$ (input actions) and $\Sigma^{IUT}_!$ (output actions) with $\Sigma_? = \Sigma^{IUT}_?$ and $\Sigma_! = \Sigma^{IUT}_!$. As usual, we also suppose that the behavior of the IUT can be modeled by an IOLTS $\mathcal{I} = (Q^{\mathcal{I}}, q^{\mathcal{I}}_{\text{init}}, \Sigma^{IUT}, \longrightarrow_{\mathcal{I}})$.

6.1 Computation of the canonical tester.

We adapt the classical construction of the canonical tester for our framework. The canonical tester that we build for a relation $\mathcal{R}$ between an IUT $\mathcal{P}_\Sigma$ and a *r-property* Π is purposed to detect all verdicts for the relation between the *r-property* and all possible interactions that can be produced with $\mathcal{P}_\Sigma$.

We define canonical testers from Streett automata. To do so, we will use a set of subsets of Streett automaton states that we introduced in [10] for runtime verification. For a Streett automaton $\mathcal{A}_\Pi$, the sets $G^{\mathcal{A}_\Pi}, G_c^{\mathcal{A}_\Pi}, B_c^{\mathcal{A}_\Pi}, B^{\mathcal{A}_\Pi}$ form a partition of $Q^{\mathcal{A}_\Pi}$ and designate respectively the good (resp. currently good, currently bad, bad) states:

- $G^{\mathcal{A}_\Pi} = \{q \in Q^{\mathcal{A}_\Pi} \cap \bigcap_{i=1}^m (R_i \cup P_i) \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \subseteq \bigcap_{i=1}^m (R_i \cup P_i)\}$
- $G_c^{\mathcal{A}_\Pi} = \{q \in Q^{\mathcal{A}_\Pi} \cap \bigcap_{i=1}^m (R_i \cup P_i) \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \not\subseteq \bigcap_{i=1}^m (R_i \cup P_i)\}$
- $B_c^{\mathcal{A}_\Pi} = \{q \in Q^{\mathcal{A}_\Pi} \cap \bigcup_{i=1}^m (\overline{R_i} \cap \overline{P_i}) \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \not\subseteq \bigcup_{i=1}^m (\overline{R_i} \cap \overline{P_i})\}$
- $B^{\mathcal{A}_\Pi} = \{q \in Q^{\mathcal{A}_\Pi} \cap \bigcup_{i=1}^m (\overline{R_i} \cap \overline{P_i}) \mid \mathit{Reach}_{\mathcal{A}_\Pi}(q) \subseteq \bigcup_{i=1}^m (\overline{R_i} \cap \overline{P_i})\}$

It is possible to show [10] that if a sequence σ reaches a state in $B^{\mathcal{A}_\Pi}$ (resp. $G^{\mathcal{A}_\Pi}$), then the underlying property Π is negatively (resp. positively) determined by σ.

The canonical tester is defined as follows.

Definition 8 (Canonical Tester). *From a Streett automaton* $\mathcal{A}_\Pi = (Q^{\mathcal{A}_\Pi}, q^{\mathcal{A}_\Pi}_{\text{init}}, \longrightarrow_{\mathcal{A}_\Pi}, \{(R_1, P_1), \ldots, (R_m, P_m)\})$ *defining a* testable *r-property* Π*, the canonical tester is the IOLTS* $T = (Q^T, q^T_{\text{init}}, \Sigma, \longrightarrow_T)$ *defined as follows:*

- $Q^T = B_c^{\mathcal{A}_\Pi} \cup G_c^{\mathcal{A}_\Pi} \cup \{\mathit{Fail}\} \cup \{\mathit{WeakPass}\}$ *with* $q^T_{\text{init}} = q^{\mathcal{A}_\Pi}_{\text{init}}$*;*
- $\longrightarrow_T$ *is defined as follows:*
  - $\forall e \in \Sigma : \mathit{Fail} \xrightarrow{e}_T \mathit{Fail} \wedge \mathit{WeakPass} \xrightarrow{e}_T \mathit{WeakPass}$,
  - $q \xrightarrow{e}_T \mathit{Fail}$ *if* $q \xrightarrow{e}_{\mathcal{A}_\Pi} q' \wedge q' \in B^{\mathcal{A}_\Pi}$*, for any* $e \in \Sigma$,
  - $q \xrightarrow{e}_T \mathit{WeakPass}$ *if* $q \xrightarrow{e}_{\mathcal{A}_\Pi} q' \wedge q' \in G^{\mathcal{A}_\Pi}$*, for any* $e \in \Sigma$,
  - $q \xrightarrow{e}_T q'$ *if* $q \xrightarrow{e}_{\mathcal{A}_\Pi} q' \wedge q, q' \in G_c^{\mathcal{A}_\Pi} \cup B_c^{\mathcal{A}_\Pi}$*, for any* $e \in \Sigma$.

A Streett automaton is transformed as follows. Transitions leading to a bad (resp. good) state are redirected to *Fail* (resp. *WeakPass*). Those latest states are terminal: the test can be stopped and the verdict produced.

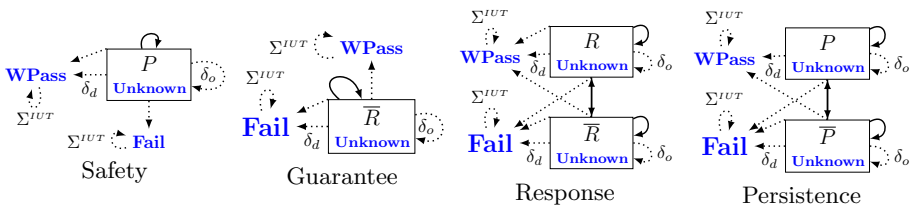


Fig. 5: Schematic illustrations of the canonical tester for basic classes

6.2 Test selection

For a given *r-property*, the set of potential sequences to be played is infinite. In practice, one may use the underlying Streett automaton to constrain the states that should be visited during a test. Furthermore, as usual, one needs to select a test case that is *controllable* [3]. It can be done on the canonical tester by first disabling input actions that do not permit to reach sought verdicts. Second, for a state in which several input actions are possible, one needs to generate different test cases with one input per state. More details can be found in [1].

Test selection plays also a role to state *weak pass* verdicts. Indeed, when dealing with sequences satisfying a *r-property so far* and not positively determining it, test selection should plan the moment for stopping the test. It can be, for instance, when the test lasted more than a given expected duration or when the number of interactions with the IUT is greater or equal than an expected number. However, one should not forget that there might exist a continuation, that can be produced by letting the test execution continue, not satisfying the *r-property* or even negatively determining it. Here, it thus remains to the tester expertise to state the halting criterion (possibly using quiescence, see Section 6.3).

6.3 Introducing quiescence

Quiescence [19, 3] was introduced in conformance testing in order to represent IUT's inactivity. In practice, several kinds of quiescence may happen (see [3] for instance). Here we distinguish two kinds of quiescence. Outputlocks (denoted δ_o) represent the situations where the IUT is waiting for an input and produces no outputs. Deadlocks (denoted δ_d) represent the situations where the IUT cannot interact anymore, *e.g.*, its execution is terminated or it is deadlocked. Thus, we introduce those two events in the output alphabet of the IUT. We have now the following additional alphabets: $\Sigma^{IUT}_{!,\delta} = \Sigma^{IUT}_{!} \cup \{\delta_o, \delta_d\}$, $\Sigma^{IUT}_{\delta} = \Sigma^{IUT}_{!,\delta} \cup \Sigma^{IUT}_{?}$.

We also have to distinguish the set of traces of the IUT from the set of potential interactions with the IUT. This latest is based on the observable behavior of the IUT and potential choices of the tester. The set of executions of the IUT is now $Exec(\mathcal{P}_{\Sigma^{IUT}}) \subseteq (\Sigma^{IUT}_{\delta})^{\infty}$. The set of interactions of the tester with the IUT is $Inter(\Sigma^{IUT}) \subseteq (\Sigma^{IUT} + \delta_o)^* \cdot (\delta_d + \epsilon)$, *i.e.,* the tester can observe IUT's outputlocks and finishes by the observation of a deadlock or program termination. When considering quiescence, characterizing testable properties now consists in comparing the set of interactions to the set of sequences described by the *r-property*. The intuitive ideas are the following:

– the tester can observe self-terminated executions of the IUT with δ_d,
– the tester can decide to terminate the program when observing an outputlock.

$Exec(\mathcal{P}_\Sigma) \subseteq \Pi$	Possible Verdicts	Testability Condition
Safety $(A_f(\psi), A(\psi))$	*fail*, *unknown*	$\overline{\psi} \neq \emptyset$
Guarantee $(E_f(\psi), E(\psi))$	*fail*, *unknown*	$\{\sigma \in \overline{\psi} \mid pref(\sigma) \subseteq \overline{\psi}\}\} \neq \emptyset$
Obligation $\bigcap_{i=1}^{k}(S_i(\psi_i) \cup G_i(\psi_i'))$ $\bigcup_{i=1}^{k}(S_i(\psi_i) \cap G_i(\psi_i'))$	*fail*, *unknown*	$\bigcup_{i=1}^{k}\left(\overline{\psi_i} \cap \{\sigma \in \overline{\psi_i'} \mid pref(\sigma) \subseteq \overline{\psi_i'}\right) \neq \emptyset$ $\bigcap_{i=1}^{k}\left(\overline{\psi_i} \cup \{\sigma \in \overline{\psi_i'} \mid pref(\sigma) \subseteq \overline{\psi_i'}\right\} \neq \emptyset$
Response $(R_f(\psi), R(\psi))$	*fail*, *unknown*	$\overline{\psi} \neq \emptyset$
Persistence $(P_f(\psi), P(\psi))$	*fail*, *unknown*	$\overline{\psi} \neq \emptyset$

Table 2: Testability wrt. $Inter(\mathcal{P}_{\Sigma^{IUT}}) \subseteq \Pi$ with quiescence

The notion of negative determinacy is now modified in the context of quiescence as follows. We say that the *r-property* Π is negatively determined upon quiescence by the sequence $\sigma \in Inter(\mathcal{P}_{\Sigma^{IUT}})$ (denoted $\ominus\text{–}determined\text{–}q(\sigma, \Pi)$) if $\ominus\text{–}determined(\sigma_{\downarrow_{\Sigma^{IUT}}}, \Pi) \vee (|\sigma| > 1 \wedge last(\sigma) \in \{\delta_d, \delta_o\} \wedge \neg\Pi((\sigma_{\ldots|\sigma|-2})_{\downarrow_{\Sigma^{IUT}}})$, where $\sigma_{\downarrow_{\Sigma^{IUT}}}$ is the projection of σ on Σ^{IUT}.

For the proposed approach, the usefulness of quiescence lies in the fact that the current test sequence does not have any continuation. Consequently, testability conditions may be weakened. Indeed, when one has determined that the current interaction with the IUT is over, it is not necessary that the *r-property* should be evaluated in the same way. In some sense, it amounts to consider that the evaluation produced by the last event before observing quiescence "terminates" the execution sequence. Thus, if the *r-property* is not satisfied by the last observed sequence, then the *r-property* is negatively determined by it.

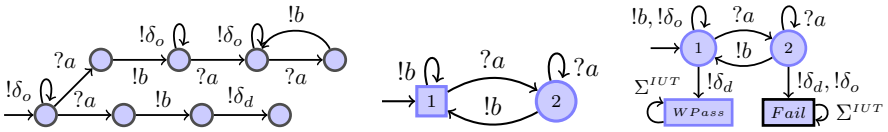
Revisiting previous results. With quiescence, the purpose of the tester is now to "drive" the IUT in a state in which the underlying *r-property* is not satisfied, and then observe quiescence. Informally, the testability condition relies now on the existence of a sequence s.t. the *r-property* is not satisfied. Testability results, upon the observation of quiescence and in order to produce *fail* verdicts when the tested *r-property* is not satisfied, are updated using the notion of negative determinacy with quiescence as shown in Table 2.

The canonical tester construction is also updated by adding the following rules for $\longrightarrow_T$: $\forall q \in B_c^{\mathcal{A}_\Pi} : q \xrightarrow{\delta_o, \delta_d}_T Fail$, $\forall q \in G_c^{\mathcal{A}_\Pi} : q \xrightarrow{\delta_o}_T q \wedge q \xrightarrow{\delta_d}_T WeakPass$. Illustrations of the construction of the canonical tester for basic classes with quiescence is given in Fig. 5, where the original (resp. modified) transitions from the Streett automaton are in plain (resp. dotted) lines.

Example 3 (Testability with quiescence). We illustrate the usefulness of quiescence. Consider the IUT depicted in Fig. 6a with observable actions $\Sigma_?^{IUT} = \{?a\}$ *and* $\Sigma_!^{IUT} = \{!b\}$*. This IUT waits for an* $?a$*, produces a* $!b$*, and then non deterministically terminates or waits for an* $?a$*, and repeats the behavior consisting in receiving an* $?a$ *and producing a* $!b$*. The executions and possible interactions with the tester are ("?" and "!" are not represented and* $x^{\&}$ *stands for* $x + \epsilon$*):*

$$Exec(\mathcal{P}_{\Sigma^{IUT}}) = {\delta_o}^{\&} \cdot \left(a^{\&} + a \cdot b \cdot (\delta_d + {\delta_o}^{\&}) \cdot (a \cdot \left[{\delta_o}^{\&} \cdot ((a \cdot b)^{\&})^*\right]^* \cdot a^{\&})^{\&}\right)$$

$$Inter(\mathcal{P}_{\Sigma^{IUT}}) = {\delta_o}^{\&} \cdot \left((a \cdot b)^{\&} \cdot (\delta_d + {\delta_o}^{\&}) \cdot (a \cdot \left[{\delta_o}^{\&} \cdot ((a \cdot b)^{\&})^*\right]^* \cdot {\delta_o}^{\&})^{\&}\right)^{\&}$$



(a) IUT (b) Response *r-property* (c) Canonical tester

Fig. 6: Illustrating the usefulness of quiescence

Now let us consider the r-property defined by the Streett automaton depicted in Fig. 6b. Its vocabulary is $\{?a, !b\}$*, and it has one recurrent state:* $R = \{1\}$*. The underlying r-property states that every input* $?a$ *should be acknowledged by an output* $!b$*. Though being not testable under the conditions expressed in Section 5, this r-property is testable with quiescence. One can observe that* $Inter(\mathcal{P}_{\Sigma^{IUT}}) \not\subseteq \Pi$ *because the existence of* $?a{\cdot}!b{\cdot}?a{\cdot}!\delta_o$ *in* $Inter(\mathcal{P}_{\Sigma^{IUT}})$*. The synthesized canonical tester is depicted in Fig. 6c.*

7 Related work and discussion

In this section we overview related work or work that may be leveraged by the results proposed in this paper. Then, we propose a discussion on the results afforded by this paper. A deeper treatment of related work is provided in [1].

Testing oriented by properties for generating test purposes. One of the limits of conformance testing [19] lies in the size of the generated test suite which can be infinite or impracticable. Some testing approaches oriented by properties were proposed to face off this limitation by focusing on critical properties. In this case, properties are used as a complement to the specification in order to generate test purposes which will be then used to conduct and select test cases [3, 20]. For a presentation of some general approaches, the reader is referred to [21].

Combining testing and formal verification. In [6], the complementarity between verification techniques and conformance testing is studied. Notably, the authors shown that it is possible to detect (using testing) violations of safety (resp. satisfaction of co-safety) properties on the implementation and the specification.

Requirement-Based testing. In requirement-based testing, the purpose is to generate a test suite from a set of informal requirements. For instance, in [22, 23], test cases are generated from LTL formula using a model-checker. Those approaches were interested in defining a syntactic test coverage for the tested requirements.

Property testing without a behavioral specification. In previous approaches, we used the notion of tiles which are elementary test modules testing specific parts of an implementation and which can be combined to test more complex behaviors using a property (see [24, 25]).

Using the Safety-Progress classification in validation techniques. The *Safety-Progress* classification of properties is rarely used in validation techniques. We used (*e.g.*, [10]) the *Safety-Progress* classification to characterize the sets of properties that can be verified and enforced during the runtime of systems. In some sense, this previous endeavor similarly addressed the expressiveness question for runtime verification and runtime enforcement.

Discussion. Several approaches fall in the scope of the generic one proposed in this paper. For instance, our results apply and extend the approach where verification is combined to testing as proposed in [6]. Furthermore, this approach leverages the use of test purposes [2, 3] in testing to guide test selection. Indeed, the characterization of testable properties gives assets on the kind of test purposes that can be used in testing. Moreover, the properties considered in this paper are framed into the *Safety-Progress* classification of properties [12, 9] which is equivalently a hierarchy of regular properties. Thus the results proposed by this paper concern previous depicted approaches in which the properties at stake can be formalized by a regular language. Furthermore, classical conformance testing falls in the scope of the proposed framework. Indeed, suspended traces of an implementation preserving the *ioco* relation wrt. a given specification can be expressed as a safety property [6].

8 Conclusion and perspectives

Conclusion. In this paper, we study the space of testable properties. We use a testability notion depending on a relation between the set of execution sequences that can be produced by the underlying implementation and the *r-property*. Leveraging the notions of positive and negative determinacy of properties, we have identified for each *Safety-Progress* class and according to the relation of interest, the testable fragment. Moreover we have seen that the framework of *r-properties* in the *Safety-Progress* classification provides a decidable test oracle in order to produce a verdict depending on the interaction between the tester and the IUT. Furthermore, we also propose some conditions under which it makes sense for a tester to state weak verdicts. Finally, results of this paper are implemented in an available prototype tool for which a description is given in [1].

Perspectives. A first research direction is to investigate the set of testable properties for more expressive formalisms. Indeed, the *Safety-Progress* classification is concerned with regular properties, and classifying testable properties for *e.g.*, context-free properties would be of interest. Another perspective is to combine the approach proposed with weak verdicts to a notion of *test coverage*. Indeed, in order to bring any confidence in the fact that *e.g.*, the implementation respects the property, it involves to execute the test several times to make it relevant. The various approaches [22, 23] for defining test coverage for property-oriented testing could be used to reinforce a set of weak verdicts.

References

1. Falcone, Y., Fernandez, J.C., Jéron, T., Marchand, H., Mounier, L.: More Testable Properties. Technical Report 7279, INRIA (2010)
2. Koch, B., Grabowski, J., Hogrefe, D., Schmitt, M.: Autolink: A Tool for Automatic Test Generation from SDL Specifications. Industrial-Strength Formal Specification Techniques (1998)
3. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. International Journal on Software Tools for Technology Transfer (STTT) (2005) 297–315

4. Traon, Y.L., Mouelhi, T., Baudry, B.: Testing Security Policies: Going Beyond Functional Testing. Int. Symp. on Software Reliability Engineering (2007) 93–102
5. Mallouli, W., Orset, J.M., Cavalli, A., Cuppens, N., Cuppens, F.: A Formal Approach for Testing Security Rules. In: SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies, ACM (2007) 127–132
6. Constant, C., Jéron, T., Marchand, H., Rusu, V.: Integrating Formal Verification and Conformance Testing for Reactive Systems. IEEE Trans. Software Eng. **33** (2007) 558–574
7. Nahm, R., Grabowski, J., Hogrefe, D.: Test Case Generation for Temporal Properties. Technical report, Bern University (1993)
8. Grabowski, J.: SDL and MSC based test case generation– an overall view of the SAMSTAG method. Technical report, University of Berne IAM-94-0005 (1994)
9. Chang, E., Manna, Z., Pnueli, A.: Characterization of Temporal Property Classes. In: Automata, Languages and Programming. (1992) 474–486
10. Falcone, Y., Fernandez, J.C., Mounier, L.: Runtime Verification of Safety-Progress Properties. In: the 9^{th} Int. Workshop on Runtime Verification. (2009) 40–59
11. Pnueli, A., Zaks, A.: PSL Model Checking and Run-Time Verification Via Testers. In: FM06: Proceedings of the 14^{th} Int Symp. on Formal Methods. (2006) 573–586
12. Manna, Z., Pnueli, A.: A Hierarchy of Temporal Properties (invited paper, 1989). In: PODC '90: Proceedings of the 9^{th} symp. on Principles Of Distributed Computing, ACM (1990) 377–410
13. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Trans. Softw. Eng. (1977) 125–143
14. Alpern, B., Schneider, F.B.: Defining Liveness. Technical report, Cornell University, Ithaca, NY, USA (1984)
15. Chang, E., Manna, Z., Pnueli, A.: The Safety-Progress Classification. Technical report, Stanford University, Dept. of Computer Science (1992)
16. Streett, R.S.: Propositional Dynamic Logic of looping and converse. In: STOC '81: Proceedings of the 13^{th} Symp. on Theory Of computing, ACM (1981) 375–383
17. Falcone, Y., Fernandez, J.C., Mounier, L.: What can You Verify and Enforce at Runtime? Technical Report TR-2010-5, Verimag Research Report (2010)
18. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL Semantics for Runtime Verification. Journal of Logic and Computation (2009)
19. Tretmans, J.: Test Generation with Inputs, Outputs, and Quiescence. In: Tools and Algorithms for the Construction and Analysis of Systems. (1996) 127–146
20. de Vries, R.G.: Towards formal test purposes. In: FATES'01: Formal Approaches to Testing of Software. (2001) 61–76
21. Machado, P.D.L., Silva, D.A., Mota, A.C.: Towards Property Oriented Testing. Electron. Notes Theor. Comput. Sci. (2007) 3–19
22. Rajan, A., Whalen, M., Heimdahl, M.: Model Validation using Automatically Generated Requirements-Based Tests. In: HASE '07: 10th IEEE Symposium on High Assurance Systems Engineering. (2007) 95–104
23. Pecheur, C., Raimondi, F., Brat, G.: A Formal Analysis of Requirements-based Testing. In: ISSTA'09: Proceedings of the 18^{th} International Symposium on Software Testing and Analysis, ACM (2009) 47–56
24. Darmaillacq, V., Fernandez, J.C., Groz, R., Mounier, L., Richier, J.L.: Test Generation for Network Security Rules. In: TestCOM, Springer (2006) 341–356
25. Falcone, Y., Fernandez, J.C., Mounier, L., Richier, J.L.: A Compositional Testing Framework Driven by Partial Specifications. In: TestCom/FATES. (2007) 107–122