RAPPORT DE RECHERCHE

L R I

A NOTE ON TEST SELECTION FOR CONF REFINEMENT

CAVALCANTI A / GAUDEL M C

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

12/2013

Rapport de Recherche N° 1569

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 650
91405 ORSAY Cedex (France)

A note on test selection for *conf* refinement

Ana Cavalcanti

University of York, Department of Computer Science
York YO10 5GH, UK

Marie-Claude Gaudel

LRI, Université de Paris-Sud and CNRS
Orsay 91405, France

Résumé.

Cette note traite de la sélection de tests pour la relation de conformité *conf* qui requiert la réduction des blocages : un blocage du système sous test doit correspondre à un blocage de la spécification. Elle accompagne et complémente le rapport de recherche 1568 où le même problème est traité pour une autre relation de conformité, la réduction des traces ("traces refinement").

Etant données une spécification et une relation de conformité, un jeu de tests exhaustif a la propriété que si un système sous test passe avec succès tous ses tests, ce système satisfait la spécification selon la relation de conformité. Les jeux de test exhaustifs sont des constructions théoriques, basées sur la syntaxe et la sémantique des spécifications. Généralement, ils ne sont pas directement utilisables : ils sont souvent infinis. Mais ils sont utilisés comme une base pour définir des stratégies de test basé sur les spécifications, via la sélection de sous-ensembles finis à partir de critères de sélection couplés à des hypothèses d'uniformité et de régularité sur le système sous test.

Le problème traité dans ce rapport survient quand la relation de conformité ne demande pas que toutes les traces de la spécification soient exécutables par le système sous test, comme c'est le cas pour la réduction des traces et la réduction des blocages. Il est alors possible qu'un sous-ensemble sélectionné contienne des tests basés sur des traces non implémentées. De ce fait, un critère de sélection de tests ne peut pas se formaliser juste comme la définition d'un sous-ensemble : cette définition doit être accompagnée de la définition de controleurs de tests qui assurent des tentatives d'exécution des tests susceptibles de contribuer à la satisfaction du critère de manière à ce que, selon les cas, au moins l'un d'eux, ou tous ceux qui sont implémentés, soi(en)t exécuté(s). Ce rapport donne ces définitions pour la relation de conformité *conf*, i. e. la réduction des blocages, et des spécifications en CSP ou *Circus* .

Mots-clés. Test basé sur les spécifications, sélection de tests, controleurs de test, CSP, *Circus*, *conf*, réduction des blocages.

December 11, 2013

A note on test selection for *conf* refinement

Ana Cavalcanti

University of York, Department of Computer Science
York YO10 5GH, UK

Marie-Claude Gaudel

LRI, Université de Paris-Sud and CNRS
Orsay 91405, France

Abstract

This is a note on test selection for deadlock reduction, that is, the *conf* conformance relation. It is a companion report of [3], where a similar problem is studied for traces refinement.

Given a specification and a conformance relation, an exhaustive test set has the property that if a system under test passed all its tests, it would satisfy the specification with respect to the conformance relation. The exhaustive test set is a theoretical construct; it is generally not directly usable: it is often infinite. It is used as a basis to define practical specification-based testing strategies, via selection of finite subsets defined by some selection criterion coupled with uniformity or regularity hypotheses on the system under test.

The problem studied here arises when the conformance relation does not require all the traces of the specification to be executable by the system under test, as for traces refinement and deadlock reduction. In such cases, it is possible that the selected subset of the exhaustive test set contains tests based on non-implemented traces. Thus the selection criterion cannot be just formalised as the definition of a subset: it must be coupled with the definition of drivers that provide an adaptive execution of the tests satisfying the criterion, in order to ensure that those that the system under test can execute will be experimented. This report provides these definitions for the case of deadlock reduction and CSP or *Circus* specifications.

Keywords: specification-based testing, test selection, process algebra, CSP, *Circus*, *conf* conformance relation, test monitoring

1. Introduction

This note is an addition to the work initiated in [3] on selecting tests from an abstract specification for testing that a system behaves like one of its refinements. In particular, we consider specifications written using process algebra like CSP [7] or *Circus* [4]. In [3] we considered traces refinement and applied the results to the selection of tests achieving synchronisation coverage. Here we address another refinement, namely *deadlock reduction*, also known as the *conf* conformance relation.

There are two motivations for this note. First, we show that the approach developed for traces refinement is applicable to other conformance relations. Secondly, we have previously shown that failures refinement, another important conformance relation for both CSP and *Circus*, can be characterised as the conjunction of traces refinement and *conf* [1, 2]. An exhaustive test set for failures refinement is, therefore, the union of the exhaustive test sets for traces refinement and for *conf*. Thus, with this note, we provide a comprehensive formal framework of test selection for failures refinement.

The principles of testing based on formal specifications can be summarised as follows. Given a specification and a conformance relation, an exhaustive test set is stated with the proved property that if a system under test (SUT) passed all the tests in this set, it would satisfy the specification in the sense captured by the conformance relation. The exhaustive test set is of theoretical value: generally not directly usable and often infinite. It is a basis for the definition of practical specification-based testing strategies, via selection of finite subsets defined by selection criteria coupled with uniformity or regularity hypotheses on the SUT. Selection criteria often correspond to a coverage requirement of the specification, but there are many variants [5].

The problem studied here arises when the conformance relation does not require all the traces of the specification to be executable by the SUT, as for traces refinement and deadlock reduction. In such cases, it is possible that the selected subset of the exhaustive test set contains tests based on non-implemented traces. Thus the selection criterion cannot be just the definition of a subset: it must be coupled with the definition of drivers that provide an adaptive execution of the tests satisfying the criterion, in order to ensure that those that the SUT can execute will be experimented.

We do not recall here all the background material for our work: a description of *Circus* using an example of a simple protocol, and an introduction to the main concepts of its testing theory can be found in [3]. The next section gives the definition of deadlock reduction and recalls the notion of tests and exhaustive test set corresponding to this notion of refinement. Section 3 develops the formal definitions of drivers and monitors that ensures adaptive execution of the tests in order to fulfill various sorts of selection criteria.

2. Testing against deadlock reduction based on *Circus* or CSP specifications

2.1. Deadlock reduction

The *conf* relation captures reduction of deadlock; it is a widely used notion of conformance in testing processes. Given two processes P_1 and P_2, we have that P_1 *conf* P_2 if, and only if, whenever P_2 engages in a sequence of events, that is, a trace, that can be accepted by P_1 as well, then P_2 can only deadlock if P_1 may as well. Formally, *conf* can be defined as follows.

Definition 1.

$$P_2 \ \mathit{conf} \ P_1 \mathrel{\widehat=} \forall\, t : \mathit{traces}(P_1) \cap \mathit{traces}(P_2) \bullet \mathit{Ref}(P_2, t) \subseteq \mathit{Ref}(P_1, t)$$
$$\mathit{where}\ \mathit{Ref}(P, t) \mathrel{\widehat=} \{\, X \mid (t, X) \in \mathit{failures}(P) \,\}$$

As in [6, pages 94, 197], given a trace t of a process P and a subset $X = \{a_1, \ldots, a_n\}$ of the set of events of P, noted αP, the pair (t, X) is a failure for P if, and only if, after performing t, P may refuse all events of X: in other words, the parallel composition below may deadlock just after t.

$$P \llbracket\, \alpha P \,\rrbracket\, (t;\ (a_1 \to P_1 \,\Box \ldots \Box\, a_n \to P_n)$$

The symbol $\Box$ represents the external choice operator and $P \llbracket\, \alpha P \,\rrbracket\, Q$ is the parallel composition of the processes P and Q with synchronisation on all the events of P.

Example 1. *We consider channels a, b, c, d, e and f that are used for synchronisation, but do not communicate any value, and the stateless Circus process P defined below, where we use $\sqcap$ for the internal choice operator.*

$$\begin{array}{l} P \mathrel{\widehat=} \mathbf{begin} \bullet \\ \quad a \to e \to c \to \mathbf{Skip} \\ \quad \sqcap \\ \quad b \to e \to (d \to \mathbf{Skip}\ \sqcap f \to \mathbf{Skip}) \\ \mathbf{end} \end{array}$$

The parallelism $P \llbracket \alpha P \rrbracket\, b \to \mathbf{Stop}$ may deadlock after the empty trace, because of a possible internal choice of the process $a \to e \to c \to \mathbf{Skip}$. Similarly, $P \llbracket \alpha P \rrbracket\, a \to \mathbf{Stop}$ may deadlock if $a \to e \to c \to \mathbf{Skip}$ is chosen. We note, however, that $P \llbracket\, \alpha P \,\rrbracket\, (a \to \mathbf{Stop} \,\Box\, b \to \mathbf{Stop}$ cannot deadlock after the empty trace, because whatever the internal choice, one event of the parallel process is accepted. Thus, $(\langle\rangle, \{b\}) \in \mathit{failures}(P)$ and $(\langle\rangle, \{a\}) \in \mathit{failures}(P)$, but $(\langle\rangle, \{a, b\}) \notin \mathit{failures}(P)$. $\Box$

In the sequel, whenever we present a *Circus* stateless process like this, for brevity, we omit the **begin** and **end** keywords and identify the process with its action after the '•'.

Given a system under test SUT and a specification SP, for SUT $conf$ SP to hold, the definition requires that, after performing every one of their common traces, the failures of SUT are failures of SP. Consequently, after a trace t of SP, SUT may refuse all events refused by SP or accept some of them. Testing for $conf$ based on the refusals of SP would be, therefore, useless. What must be tested is that, after every trace t of SP, SUT cannot refuse all events in a set X of events such that $(t, X) \notin failures(SP)$.

Such sets of events are called *acceptance sets* of SP after t.

2.2. Tests and exhaustive test set for conf

Testing for deadlock reduction is achieved by executing in parallel with the SUT a process corresponding to a trace t of the specification followed by an external choice among all events of an acceptance set after t of the specification. Formally, given a system under test SUT and a specification SP, a test execution proposes to SUT the traces of the process $a_1 \longrightarrow a_2 \longrightarrow \ldots \longrightarrow \mathbf{Skip}\ ;\ (\Box\ a : X \bullet a \rightarrow \mathbf{Stop})$ where the trace $t = \langle a_1, a_2, \ldots \rangle$ is such that $t \in traces(SP)$ and $(t, X) \notin failures(SP)$.

The verdict of one execution of such a test is as follows. If t, followed by a deadlock is observed, the test execution is said to be a *failure*. If a trace $t \frown a$, with $a \in X$ is observed the result of the test is said to be a *success*. If a strict prefix of t followed by a deadlock is observed, the test execution is said to be *inconclusive*; the trace t has not been executed by SUT during this test execution, thus it was not possible to test the absence of additional deadlock after t.

For a trace t, and an acceptance set X, a test is characterised by $T_{conf}(t, X)$ as defined below. As explained above, it offers the trace t, and at the end a choice of the events in X. If SUT does not synchronise on all events of t, the test is inconclusive, since it is not necessary for SUT to allow for all the traces indicated in the specification. If the trace t is accepted, however, then SUT must accept at least one event of X. The verdict is indicated by extra special events inc, $pass$, and $fail$.

Definition 2.

$$T_{conf}(\langle\,\rangle, X) = fail \longrightarrow (\Box\ e : X \bullet e \longrightarrow pass \longrightarrow \mathbf{Stop})$$
$$T_{conf}(\langle\, e\,\rangle \frown t, X) = inc \longrightarrow e \longrightarrow T_{conf}(t, X)$$

Example 2. *In Example 1, we have seen that $\{a, b\}$ is an acceptance set of P after $\langle\,\rangle$. Thus we have:*

$$T_{conf}(\langle\,\rangle, \{a, b\}) = fail \longrightarrow ((a \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (b \longrightarrow pass \longrightarrow \mathbf{Stop}))$$

Another pair that is not in $failures(P)$ is $(\langle b, e\rangle, \{d, f\})$. This leads to the test below.

$$T_{conf}(\langle b, e\rangle, \{d, f\}) = inc \longrightarrow b \longrightarrow e \longrightarrow fail \longrightarrow ((d \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (f \longrightarrow pass \longrightarrow \mathbf{Stop}))$$

□

Execution of one of these tests T is carried out by executing T and SUT in parallel, synchronising on all events of the specification (that is, the non-verdict events).

Definition 3.

$$Execution^{SP}_{SUT}(T) = (SUT \llbracket \alpha SP \rrbracket\ T) \setminus \alpha SP$$

The exhaustive test set for $conf$ contains all the tests obtained from the traces and acceptances of SP.

Definition 4. *Given a specification SP, we define*

$$Exhaust_{conf}(\mathtt{SP}) = \{\ T_{conf}(t, X) \mid t \in traces(SP) \wedge (t, X) \notin failures(SP)\ \}$$

Exhaustiveness of $\mathit{Exhaust}_{\mathit{conf}}(SP)$ is formally established by the following theorem, proved in [2].

Theorem 1 (Exhaustiveness of $\mathit{Exhaust}_{\mathit{conf}}$). *Given two Circus processes, SP and SUT, we have that SUT conf SP if and only if,*

$$\begin{array}{l}\forall\, T \in \mathit{Exhaust}_{\mathit{conf}}(\mathtt{SP}), t', X' \mid (t', X') \in \mathit{failures}(\mathit{Execution}_{SUT}^{SP}(T)) \bullet \\ \quad \mathit{last}\ t' \neq \mathit{fail} \vee X' \neq \{\, \mathit{inc}, \mathit{pass}, \mathit{fail} \,\}\end{array}$$

Remark. Actually, it is sufficient to consider the minimal acceptance sets, as explained in [1]: acceptance sets have the property that if X is an acceptance set after a given trace, then it is also the case for all the sets containing X, and the tests corresponding to strict supersets of X are redundant. For the sake of brevity, we leave aside the technicalities of the removal of the unnecessary tests.

3. Selection and deadlock reduction

Just as for traces refinement, when the considered conformance relation is *conf*, it is acceptable that a trace of the specification is not implemented in the SUT. Moreover, in the case of *conf*, the SUT may have more traces than the specification. This is illustrated in the following example.

Example 3. *For instance, Q conf P, where P is the process of Example 1, and Q is as follows.*

$$Q \mathrel{\widehat=} (d \rightarrow \mathbf{Skip}) \sqcap (b \rightarrow d \rightarrow (e \rightarrow \mathbf{Skip} \ \sqcap f \rightarrow \mathbf{Skip}))$$

Due to the first internal choices of Q and P, $\mathit{Ref}(Q, \langle\rangle)$ is the set of subsets of $\alpha Q = \{b, d, e, f\}$ except those that contain both d and b, and $\mathit{Ref}(P, \langle\rangle)$ is the set of subsets of $\alpha P = \{a, b, c, d, e, f\}$ except those that contain both a and b. Since $a \notin \alpha Q$, all the sets in $\mathit{Ref}(Q, \langle\rangle)$ are in $\mathit{Ref}(P, \langle\rangle)$.

Considering the non empty traces of Q, $\langle d\rangle$ is a trace of Q but not a trace of P, thus there is no requirement on its refusals. The traces of Q that begin with b are traces of P with the same refusals.

Thus Q conf P, but traces $\langle a\rangle, \langle a, e\rangle, \langle a, e, c\rangle$ that are traces of P are not traces of Q. □

Therefore, it may be the case that a selected test is not executable, and if there is another test that may be able to fulfill the selection criterion, it must be executed, and we should proceed in this way, until either a satisfactory test is executed or it is certain that none exists. This has an impact on the way in which test selection can be defined and applied. Thus, as for traces refinement, there are two main interrelated tasks involved in the definition of a selection strategy for *conf*. Namely, the (formal) definitions of

1. the (possibly still infinite) subsets of tests that can contribute to the fulfillment of the selection criterion, and
2. a test driver that attempt to fulfill the criterion at run-time by executing some of these tests despite the fact that some traces, and thus some tests, are not implemented.

3.1. An introductory example: selection for coverage of an event

What does it mean to cover an event e when the conformance relation is *conf*? One may consider that it requires to exercise either one test or all tests $T_{\mathit{conf}}(t, X)$ such that:

- e occurs in trace t, or
- e is in the acceptance set X, or
- e either occurs in trace t, or is in the acceptance set X, or
- e occurs in t and is in X.

Example 4. *The tests of P for deadlock reduction are built, as explained above, from couples of trace and minimal acceptance sets. Those for P are given below.*

$\langle\,\rangle, \{a, b\}$
$\langle a\rangle, \{e\}$
$\langle a, e\rangle, \{c\}$
$\langle a, e, c\rangle, \varnothing$
$\langle b\rangle, \{e\}$
$\langle b, e\rangle, \{d, f\}$
$\langle b, e, d\rangle, \varnothing$
$\langle b, e, f\rangle, \varnothing$

This leads to the following tests.

$fail \longrightarrow (a \longrightarrow pass \longrightarrow \mathtt{Stop} \,\Box\, b \longrightarrow pass \longrightarrow \mathbf{Stop})$
$inc \longrightarrow a \longrightarrow fail \longrightarrow e \longrightarrow pass \longrightarrow \mathbf{Stop}$
$inc \longrightarrow a \longrightarrow inc \longrightarrow e \longrightarrow fail \longrightarrow c \longrightarrow pass \longrightarrow \mathbf{Stop}$
$inc \longrightarrow b \longrightarrow fail \longrightarrow e \longrightarrow pass \longrightarrow \mathbf{Stop}$
$inc \longrightarrow b \longrightarrow inc \longrightarrow e \longrightarrow fail \longrightarrow (d \longrightarrow pass \longrightarrow \mathbf{Stop} \,\Box\, f \longrightarrow pass \longrightarrow \mathbf{Stop})$

For ensuring the execution of one test where e occurs in the trace (that is, before fail), it is necessary to select the third and last tests, since one of them, or both, may correspond to a non-implemented trace. $\Box$

Remark. The couples where the acceptance set is empty do not yield any test.

3.2. Drivers and monitors for coverage of one event

Given a specification SP and the selection criterion that the event e must be covered in the trace of a conclusive test, we consider the subset of those tests where there is at least one occurrence of e in the trace.

$$Exhaust_{conf}(SP)\restriction_e^t = \{\, T_{conf}(t, X) \mid t \in traces(SP) \wedge t\restriction_e \neq \langle\,\rangle \wedge (t, X) \notin failures(SP)\,\}$$

where $t\restriction_e$ eliminates from the trace t all the events different from e. This (possibly infinite) subset of $Exhaust_{conf}(SP)$ covers all the traces of the specification where e occurs at least once. We define below a test driver that runs all these tests until a *fail* verdict, possibly followed by a *pass* verdict, is observed. This is ensured by the test monitor process $TMonitor_{conf}$ executed alongside the tests.

$$\begin{aligned} Driver_{conf}(SP, SUT)\restriction_e^t = \; & (\;|||\; T : Exhaust_{conf}(SP)\restriction_e^t \bullet Execution_{SUT}^{SP}(T)) \\ & \qquad [\![\{\!|\, inc, pass, fail \,|\!\}]\!] \\ & \quad TMonitor_{conf} \end{aligned}$$

where

$$TMonitor_{conf} = inc \to TMonitor_{conf} \,\Box\, fail \to pass \to \mathbf{Stop}$$

If e is not covered at runtime with the strategy above, it is because all the tests return an inconclusive verdict. On the other hand, the fact that all tests return an inconclusive verdict does not mean that e has not been executed, since some of the tests may return an inconclusive verdict after having exercised e.

In the case where the coverage criterion is that e is in the acceptance set, the subset of $Exhaust_{conf}(SP)$ to be considered by the driver above can be defined as follows.

$$Exhaust_{conf}(SP)\restriction_e^a = \{\, T_{conf}(t, X) \mid t \in traces(SP) \wedge (t, X) \notin failures(SP) \wedge e \in X\,\}$$

The above criterion is rather sensible in the case of deadlock reduction: it may be the case that, for critical reasons, a special event must never be blocked when it is not specified that this is possible. We note, however, that if the acceptance set is not a singleton and when the SUT is non-deterministic, one relies on the complete-test assumption to be sure that the event of interest is covered at runtime.

If the criteria is that e either occurs in the trace, or is in the acceptance set, the driver must consider

$$Exhaust_{conf}(SP)\restriction_e^T = \{ T_{conf}(t,X) \mid t \in traces(SP) \wedge (t,X) \notin failures(SP) \wedge (t\restriction_e \neq \langle\rangle \vee e \in X) \}$$

When e must occur in both the trace and the acceptance set, the $\vee$ above is replaced by $\wedge$

3.3. Generalisation

Leaving the introductory special case of the coverage of one event, we consider now the general issue of test selection for a given criterion. As for traces refinement, when selecting tests based on some property π, we consider existential selection of tests with respect to π, and universal selection with respect to π.

As seen above, either properties of traces or of tests can be considered for selection criteria. Below, we first discuss existential or universal selection of traces. In this case, the fact that there are several tests for one trace must be taken into account. Afterwards, we discuss selection based on properties of tests.

For each trace t of a specification SP, we have the set

$$AllTests_{conf}(t,SP) = \{ T_{conf}(t,X) \mid (t,X) \notin failures(SP) \}$$

of all tests built from t and one of its acceptance sets X. For a property π of traces, we have:

existential selection of traces: there is *at least one* trace t satisfying π, such that $AllTests_{conf}(t,SP)$ is actually executed (that is, all tests in $AllTests_{conf}(t,SP)$ are executed); or

universal selection of traces: for *all implemented* traces t satisfying π, $AllTests_{conf}(t,SP)$ is executed.

Coming back to test selection, for a given property π of tests, one may require either:

existential selection of tests: there is *at least one* test T satisfying π, that is actually executed, if there is one that is implemented, or

universal selection of tests: *all implemented* tests T satisfying π are executed.

Below, we consider exhaustive test sets and drivers for each of these classes of selection criteria.

3.3.1. Existential selection of traces

For existential selection, the selected test set $Existential_{conf}^{trace}(SP)\restriction_\pi$ turns out to be a set of test sets.

$$Existential_{conf}^{trace}(SP)\restriction_\pi = \{AllTests_{conf}(t,SP) \mid t \in traces(SP) \wedge t \vdash \pi\} \tag{1}$$

We use the notation $t \vdash \pi$ to indicate that trace t satisfies property π.

The execution of all the tests associated with an implemented trace t can be specified as a simple driver that runs independently and to conclusion all tests in $AllTests_{conf}(t,SP)$:

$$\vert\vert\vert\; T : AllTests_{conf}(t,SP) \bullet Execution_{SUT}^{SP}(T)$$

Using such basic drivers, a driver that ensures existential coverage can be designed as follows: independently, for each trace t that defines a set of tests $AllTests_{conf}(t,SP)$ in $Existential_{conf}^{trace}(SP)\restriction_\pi$, the driver above is run under the control of a monitor that observes whether a *fail* event takes place. When it does, it means that t is implemented. The monitor continues the execution of the other tests based on t, and sends to the other drivers for the other traces a *done* event to make them to stop. Formally, we have the following definition for a generic driver, for a specification SP and some property π of a trace.

$$\begin{array}{l} ExistDriver_{conf}^{trace}(SP,SUT)\restriction_\pi = \\ \quad \Big([\![\{\!| done |\!\}]\!]\; TS : Existential_{conf}^{trace}(SP)\restriction_\pi \bullet \\ \qquad (\vert\vert\vert\; T : TS \bullet Execution_{SUT}^{SP}(T)\,)\; [\![\{\!| inc, pass, fail |\!\}]\!]\; TSMonitor_{conf} \Big) \setminus \{\!| done |\!\} \end{array}$$

where $\mathit{TSMonitor}_{\mathit{conf}}$ is below. It stops when a *done* occurs, and raises a *done* event when a *fail* occurs.

$$\mathit{TSMonitor}_{\mathit{conf}} = \mathit{done} \longrightarrow \mathbf{Stop} \,\Box\, \mathit{inc} \longrightarrow \mathit{TSMonitor}_{\mathit{conf}} \,\Box\, \mathit{fail} \longrightarrow \mathit{done} \longrightarrow \mathit{Run}$$

Run just continues running the other tests based on the same trace.

$$\mathit{Run} = \mathit{inc} \to \mathit{Run} \,\Box\, \mathit{pass} \to \mathit{Run} \,\Box\, \mathit{fail} \to \mathit{Run}$$

In the definition above of $\mathit{ExistDriver}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP})\restriction_{\pi}$, each of the sets TS in $\mathit{Existential}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP})\restriction_{\pi}$ is considered. The drivers for each of these sets are run in parallel, synchronising on the event *done*. Each driver runs all tests T in TS independently, but under the control of $\mathit{TSMonitor}_{\mathit{conf}}$, which observes *done* as well as the verdict events. If a *done* event occurs, $\mathit{TSMonitor}_{\mathit{conf}}$ stops (and so the tests under its control stop). Otherwise, the tests proceed until a *fail* is observed, when *done* is raised, but the controlled tests proceed to conclusion, since *Run* just ignores all verdict events. The *done* event is local to the existential driver; it is used only for synchronisation between the monitors of the tests associated with each trace t.

Example 5. *Coming back to trace coverage of an event e we have:*

$$\mathit{Existential}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP})\restriction_{(t\restriction_e \neq \langle\rangle)} = \{\mathit{AllTests}_{\mathit{conf}}(t, \mathit{SP}) \mid t \in \mathit{traces}(\mathit{SP}) \land t \restriction e \neq \langle\rangle\}$$

The distributed union of the test sets in this set is exactly $\mathit{Exhaust}_{\mathit{conf}}(\mathit{SP})\restriction_e$. Here, their grouping based on the traces t as defined by $\mathit{AllTests}_{\mathit{conf}}(t, \mathit{SP})$ allows us to consider the alternative driver below.

$$\begin{array}{l}\mathit{ExistDriver}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP}, \mathit{SUT})\restriction_{(t\restriction_e \neq \langle\rangle)} = \\ \quad \Big([\![\{\!|\, \mathit{done} \,|\!\}]\!] \; \mathit{TS} : \mathit{Existential}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP}) \restriction_{(t\restriction_e \neq \langle\rangle)} \bullet \\ \qquad (\mathop{|||} T : \mathit{TS} \bullet \mathit{Execution}^{\mathit{SP}}_{\mathit{SUT}}(T) \,) \; [\![\{\!|\, \mathit{inc}, \mathit{pass}, \mathit{fail} \,|\!\}]\!] \; \mathit{TSMonitor}_{\mathit{conf}} \Big) \setminus \{\!|\, \mathit{done} \,|\!\}\end{array}$$

This driver ensures that for at least one implemented trace covering e, if there is one, all the tests are executed, that is, all the acceptance sets are attempted. $\Box$

As usual, if the SUT is nondeterministic, coverage has to rely on the complete-test assumption and on running the driver several times, possibly with some instrumentation of the SUT and of its execution environment. This is the case for all drivers that we present.

3.3.2. Universal selection of traces

The case of universal selection is easier to formalise; there is no need to consider a set of test sets as in the previous section. The test set to be considered is as follows.

$$\mathit{Universal}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP})\restriction_{\pi} = \{\, T_{\mathit{conf}}(t, X) \mid t \in \mathit{traces}(\mathit{SP}) \land t \vdash \pi \land (t, X) \notin \mathit{failures}(\mathit{SP}) \,\} \tag{2}$$

All tests that correspond to implemented traces must be run, so the driver attempts to execute all tests.

$$\mathit{UnivDriver}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP}, \mathit{SUT})\restriction_{\pi} = \mathop{|||} T : \mathit{Universal}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP}) \restriction_{\pi} \bullet \mathit{Execution}^{\mathit{SP}}_{\mathit{SUT}}(T)$$

Example 6. *Coming back again to coverage of an event e, we have*

$$\mathit{Exhaust}_{\mathit{conf}}(\mathit{SP})\restriction_e = \mathit{Universal}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP})\restriction_{(t\restriction_e \neq \langle\rangle)}$$

The corresponding driver $\mathit{UnivDriver}^{\mathit{trace}}_{\mathit{conf}}(\mathit{SP}, \mathit{SUT})\restriction_{(s\restriction_e \neq \langle\rangle)}$ executes all tests arising from all traces that include at least one occurrence of e. $\Box$

3.3.3. Existential or universal selection of tests

For a given property π of tests, we consider, for both existential and universal selection, the following exhaustive test set. We use $T \vdash \pi$ to denote the fact that test T satisfies π.

$$Exhaust^{test}_{conf}(SP)\restriction_{\pi} = \{\, T : Exhaust_{conf}(SP) \mid T \vdash \pi \}$$

There is no need to use a set of test sets like we have done for existential selection of traces. For existential selection of the tests satisfying π, the driver is:

$$\begin{array}{l} ExistDriver^{test}_{conf}(SP, SUT)\restriction_{\pi} = \\ \quad (\mathop{|||}\, T : Exhaust^{test}_{conf}(SP) \restriction_{\pi} \bullet Execution^{SP}_{SUT}(T)) \, [\![\, \{\!| \, inc, pass, fail \, |\!\} \,]\!] \; TMonitor \end{array}$$

where $TMonitor$ is

$$TMonitor = inc \rightarrow TMonitor \,\square\, fail \rightarrow pass \rightarrow \mathbf{Stop}$$

For universal selection of tests satisfying π, the driver is:

$$UnivDriver^{test}_{conf}(SP, SUT)\restriction_{\pi} \;=\; \mathop{|||}\, T : Exhaust^{test}_{conf}(SP) \restriction_{\pi} \bullet Execution^{SP}_{SUT}(T)$$

This is exactly the same driver used for universal selection based on traces, except that it uses the tests in $Exhaust^{test}_{conf}(SP)\restriction_{\pi}$ (rather than those in $Universal^{trace}_{conf}(SP)\restriction_{\pi}$).

3.4. Factorisation

Instead of using a monitor process to define special drivers, it is possible to factorise the tests in the exhaustive test set to define a single tree-shaped test. This test offers the choice between all the tests selected. In this case, the SUT drives the test by making the (external) choices offered in the test. Since just one path of the tree-shaped test is executed in a test experiment, there is an existential selection (by the SUT) of one test among those combined in the tree. Moreover, only the complete-test assumption and enough repeated executions of the test can guarantee that all its paths (and, therefore, all tests combined in the tree) are executed.

We define below the process $end(T)$; it captures the final behaviour of a test T, where, after signalling a *fail*, it attempts to execute an event in a acceptance set.

Definition 5.

$$end(fail \longrightarrow P) = fail \longrightarrow P \qquad\qquad end(inc \longrightarrow e \longrightarrow P) = end(P)$$

We now define a function $Fact^{SP}_{conf}(TS)$ that defines the factorised tree-shaped tests coming from a test set TS. It is defined in terms of a function $FactP^{SP}_{conf}(TS, t)$, which takes as an extra parameter the path (a trace) t that defines the branch of the tree-shaped test that is being constructed.

Definition 6 (Factorisation of tests for *conf*).

$$Fact^{SP}_{conf}(TS) = FactP^{SP}_{conf}(TS, \langle\rangle)$$

$$FactP^{SP}_{conf}(TS, t) = TPass(TS, t) \;\square\; TCont^{SP}_{conf}(TS, t)$$

$$TPass(TS, t) = \mathbf{Stop} \qquad \textit{[if } t \notin trace\,(\!|\, TS \,|\!)\textit{]}$$

$$TPass(TS, t) = \mathop{\square}\, T : \{\, T : TS \mid trace(T) = t \,\} \bullet end(T) \qquad \textit{[if } t \in trace\,(\!|\, TS \,|\!)\textit{]}$$

$$TCont^{SP}_{conf}(TS, t) = \mathbf{Stop} \qquad \textit{[if } initials(TS, t) = \varnothing\textit{]}$$

$$TCont^{SP}_{conf}(TS, t) = inc \longrightarrow \mathop{\square}\, e : initials(TS, t) \bullet e \longrightarrow FactP^{SP}_{conf}(TS, t \frown \langle e \rangle) \qquad \textit{[if } initials(TS, t) \neq \varnothing\textit{]}$$

For any set TS, we have that $Fact^{SP}(TS)$ is given by $FactP^{SP}(TS, \langle\rangle)$; initially, just the empty path (that is, trace) has been constructed. For any path t, this is an external choice between the tests defined by the functions $TPass(TS, t)$ and $TCont^{P}_{conf}(TS, t)$.

With $TPass(TS, t)$, we consider whether t is the trace of a test in TS or not. We define the set $trace \llparenthesis TS \rrparenthesis$ of traces of a set of tests TS as the set of specification traces used in its tests, not including the choice in the acceptance set. We use the relational image operator $f \llparenthesis S \rrparenthesis$; it provides the set of results obtained by applying f to each element in the set S. The trace of a single test T is defined as follows.

Definition 7.

$$trace(fail \longrightarrow P) = \langle\rangle \qquad\qquad trace(inc \longrightarrow e \longrightarrow P) = \langle e\rangle \frown trace(P)$$

It t is not a trace of a test in TS, then $TPass(TS, t)$ is **Stop**, which is a unit for external choice: $\mathbf{Stop} \Box P = P$. For example, TS may not have any test for the empty trace $\langle\rangle$, and in this case $TPass(TS, \langle\rangle)$ does not add anything to the tree-shaped test. On the other hand, if there are tests T in TS whose trace is t, then, for each of them, we add its treatment of the acceptance set, characterised by $end(T)$, to the tree.

With $TCont^{P}_{conf}(TS, t)$, we consider whether t is the prefix of a trace of a test in TS or not. We use the notion of initials of a test set TS after a trace t, which is formally defined as follows.

Definition 8.

$$initials(TS, t) = \{e \mid \exists\, T : TS;\ s \bullet trace(T) = t \frown \langle e\rangle \frown s\}$$

If t has no continuation, then $TCont^{P}_{conf}(TS, t)$ is just **Stop**. Otherwise, we add an intermediary verdict inc followed by a branch (in an external choice) for each of the continuations e.

Example 7. *We call TSP the set of tests of Example 4.*

$$Fact^{P}_{conf}(TSP) = FactP^{P}_{conf}(TSP, \langle\rangle) = TPass(TSP, \langle\rangle) \Box TCont^{P}_{conf}(TSP, \langle\rangle)$$

since $trace(fail \longrightarrow ((a \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (b \longrightarrow pass \longrightarrow \mathbf{Stop}))) = \langle\rangle$,

$$\begin{aligned}&TPass(TSP, \langle\rangle)\\ &= end(fail \longrightarrow ((a \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (b \longrightarrow pass \longrightarrow \mathbf{Stop})))\\ &= fail \longrightarrow ((a \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (b \longrightarrow pass \longrightarrow \mathbf{Stop}))\end{aligned}$$

and since $initials(TSP, \langle\rangle) = \{a, b\}$ *we have*

$$TCont^{P}_{conf}(TSP, \langle\rangle) = inc \longrightarrow (a \longrightarrow FactP^{P}_{conf}(TSP, \langle a\rangle) \Box (b \longrightarrow FactP^{P}_{conf}(TSP, \langle b\rangle)))$$

Finally, the factorised test is

$$\begin{aligned}&fail \longrightarrow ((a \longrightarrow pass \longrightarrow \mathbf{Stop}) \Box (b \longrightarrow pass \longrightarrow \mathbf{Stop}))\\ &\Box\\ &inc \longrightarrow (a \longrightarrow (fail \longrightarrow e \longrightarrow pass \longrightarrow \mathbf{Stop} \Box inc \longrightarrow e \longrightarrow fail \longrightarrow c \longrightarrow pass \longrightarrow \mathbf{Stop})\\ &\qquad\qquad \Box\\ &\qquad\qquad b \longrightarrow (fail \longrightarrow e \longrightarrow pass \longrightarrow \mathbf{Stop}\\ &\qquad\qquad\qquad \Box\\ &\qquad\qquad\qquad inc \longrightarrow e \longrightarrow fail \longrightarrow (d \longrightarrow pass \longrightarrow \mathbf{Stop} \Box f \longrightarrow pass \longrightarrow \mathbf{Stop}))\end{aligned}$$

No special driver is required in this case. □

The two approaches, namely, factorisation and the definition of exhaustive test sets and drivers, are not exactly equivalent. In the execution of a factorised test, the control, exercised via choice of events, is left to the SUT. In the execution of a set of tests, the driver controls the execution of the tests.

4. Conclusions

We have established that the work on selection for traces refinement presented in [3] is applicable to other conformance relations, for instance, *conf*. Since failures refinement can be characterised as the conjunction of traces refinement and *conf* [1, 2], the union of exhaustive test sets for traces refinement and for *conf* is an exhaustive test set for failures refinement. What we provide is, therefore, a comprehensive formal framework of test selection for failures refinement, not just *conf*.

The issue of partial implementation of symbolic tests can be dealt with similarly to what is done in [3] for traces refinement. There is nothing in that material specific to a particular conformance relation, since it addresses the treatment of existing previously selected tests of any shape.

Acknowledgements. We are grateful to the Royal Society and the CNRS for funding our collaboration via their International Exchange scheme.

References

[1] A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in CSP. In *9th International Conference on Formal Engineering Methods*, volume 4789 of *Lecture Notes in Computer Science*, pages 151 – 170. Springer-Verlag, 2007.

[2] A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in *Circus*. *Acta Informatica*, 48(2):97 – 147, 2011.

[3] A. L. C. Cavalcanti and M.-C. Gaudel. Test selection for traces refinement. Technical Report 1568, LRI, http://www.lri.fr/Rapports-internes, Université Paris-Sud XI, October 2013.

[4] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of Computing*, 15(2 - 3):146 – 181, 2003.

[5] R M. Hierons. Comparing test sets and criteria in the presence of test hypotheses and fault domains. *ACM Transactions on Software Engineering and Methodology*, 11(4):427–448, 2002.

[6] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.

[7] A. W. Roscoe. *Understanding Concurrent Systems.* Texts in Computer Science. Springer, 2011.