Synthesis of Distributed Testers from True-concurrency Models of Reactive Systems

Claude Jard

To cite this version:

Claude Jard. Synthesis of Distributed Testers from True-concurrency Models of Reactive Systems. [Research Report] RR-4567, INRIA. 2002. inria-00072021

HAL Id: inria-00072021
https://hal.inria.fr/inria-00072021

Submitted on 23 May 2006

Synthesis of Distributed Testers from True-concurrency Models of Reactive Systems

Claude JARD

N˚4567

Octobre 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4567--FR+ENG

Synthesis of Distributed Testers from True-concurrency Models of Reactive Systems

Claude JARD

Thème 1 — Réseaux et systèmes
Projet Triskell

Rapport de recherche n˚4567 — Octobre 2002 — 32 pages

Abstract: Automatic synthesis of test cases for conformance testing has been principally developed with the objective of generating sequential test cases. In the distributed system context, it is worth extending the synthesis techniques to the generation of multiple testers. We base our work on our experience in using model-checking techniques, as successfully implemented in the TGV tool. Continuing the works of A. Ulrich and H. König, we propose to use a true-concurrency model based on graph unfolding. The article presents the principles of a complete chain of synthesis, starting from the definition of test purposes and ending with a projection onto a set of testers.

Key-words: Testing, Distributed systems, Synthesis, True-concurrency models, Interoperability

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Synthèse de tests répartis
à l'aide d'un modèle de vrai-parallélisme
des systèmes réactifs

Résumé : La synthèse automatique de tests de conformité a été jusqu'à présent principalement développée dans l'objectif d'obtention de tests séquentiels. Dans le contexte des systèmes répartis, il est tentant d'étendre les techniques de synthèse à la génération de testeurs multiples. Nous fondons notre proposition sur notre expérience dans l'utilisation des techniques de "model-checking" comme mis en oeuvre dans l'outil TGV. Continuant les travaux de A. Ulrich et H. König, nous proposons d'utiliser un modèle de vrai parallélisme ("true-concurrency" en anglais) basé sur le dépliage de graphes. Le rapport présente les principes d'une chaîne complète de synthèse, partant de la définition d'objectifs de test, et allant jusqu'à la projection sur un ensemble de testeurs.

Mots-clé : Test, Systèmes répartis, Synthèse, Modèles de vrai parallélisme, Interopérabilité

1 Introduction

Building distributed applications and systems is a complex task. Not only must the assembly of numerous components inherent to classical software engineering be mastered, the intrinsic complexity of asynchronous communicating systems, revealed by non-deterministic behaviours must also be dealt with. In this context, attention must be paid to validation. Among the different means of validation, testing is a pragmatic way to gain some confidence. When theoretically founded, it can provide an efficient mean for early detection of functional errors. It is relatively easy to write test cases; the difficult question is to know what is actually tested: what is the meaning of a given verdict? Can false errors be detected? How is coverage to be estimated?

For many years these questions have been tackled in protocol engineering, and particularly in telecommunications, from the point of view of black-box conformance testing with respect to a specification. This scientific community now has a serious methodological basis (2) and tools based on formal methods.

Algorithms for automatic test synthesis have been proposed both in the academic world, and in industry. However, the use of these tools reaches a limit when testing distributed systems. This is because they are dedicated to the synthesis of sequential test cases (represented by event sequences or finite automata). Such synthesis is not always well-suited to test systems containing parallel activities. It is also known that a state representation of a specification with parallelism often suffers from a combinatorial explosion. The interest in generating distributed test cases was recognised a few years ago, as demonstrated by the inclusion of concurrent constructs in the TTCN standard.

We retain three main motivations to synthesise distributed test cases:

- It can be naturally imposed by the test architecture under consideration. Let us consider a system geographically scattered on a network. The idea is to design a set of testers, each tester being located at the communicating entity to be checked, and communicating with the other testers to co-ordinate the test activity and the production of diagnosis.

- It allows more compact and clear test cases to be obtained. This is the case when the system under test produces concurrent observable events: a sequential representation would require all possible interleavings to be computed.

This rapidly suffers from a combinatorial explosion as the concurrency increases.

- In certain cases parallel testing is needed to check particular behaviours. For example, one often considers for controllability reasons that the testers must wait for the system stabilisation before injecting new interactions. Under this assumption, it was shown by (3) that a distributed test case can position the system under test into states which are not reachable by a sequential test. More generally, the situation will also occur in the context of real-time testing.

One can distinguish two main approaches to synthesising distributed test cases:

- The generation of sequential test cases, followed by their automated distribution (18). The idea is to produce a set of communicating testers which behave like the sequential test (i.e. in the sense of trace equivalence). The advantage of this approach is that it requires no more than the current state of the art; it can even be used on hand-written test cases. The major drawback is that it does not take into account the intrinsic parallelism of the system under test. In general, one does not know how to distinguish between parallelism and interleaving; in practice this leads to useless synchronisation between the local testers.

- The re-examination of the synthesis, retaining the parallelism information contained in the formal specification during the construction of the test cases. We discuss this extreme approach in the paper. The main difficulty is the use of a true-concurrency model in which causality and concurrency are explicitly represented, in place of the usual automata or transition system models. This kind of model has been mainly developed by theoreticians and has not yet been fully exploited. The synthesis of distributed test cases appears to be an interesting context to use the explicit parallelism included in the model.

The question of the automatic synthesis of distributed testers is relatively recent. It has appeared gradually from the notion of multiple, then distributed interfaces. For example, in (19), the system under test is modelled by a single finite state machine with several distributed interfaces. A test generation method is sketched, based on the idea of synchronisable test suites. In (20), multiple testers are generated by considering only particular synchronous behaviours of the parallel specification. Co-ordination of the testers makes the assumption that the communications between entities of the system under test are observable. The idea of using true-concurrency models in the case of asynchronous systems came from two research groups separately (one in Korea, driven by M. Kim, the other in Germany, driven by A. Ulrich and H. König). In Kim's approach (21; 22), they adopt a specific model, which consists in computing particular concurrent paths from a communicating finite state machines view. The introduction of event duration makes the computation easier. It is not clear however to know the algorithmic complexity of the method and how it scales up in a real testing methodology (abstraction and selection for example). We chose to follow the Ulrich and König's approach (23), mathematically based on theoretical and algorithmic results on Petri nets. The partial order semantics of Petri nets and its implementation in the "unfolding" algorithmic has been developed for many years, but rather confined in the theoretical computer science community. We think it is enough sound and advanced to be applied in several domains, like distributed testing. In (23), the unfolding of "behaviour machines" is used to propose a "partial order transition cover" as a general heuristics to select partial order test cases, which could be later projected on parallel testers. In the same vein, (24) has tempted to avoid the use of Petri nets and to directly generate the partial orders (event structures) in the context of asynchronous communication. In this article, we contribute to this direction, in precising the different algorithms, from the notion of test purpose to the final construction of test automata. The first principles were briefly presented to the Testcom conference (1).

The rest of the paper is organised as follows: First, we give an overview of a test synthesis method based on model-checking and sequential transition systems, as implemented in the TGV tool. We then propose to revisit the whole test-production chain using partial-order representations of the behaviours. This is presented in Section three, following the different steps of the methodology: the partial order view of a specification (the notion of "tile") and of a test purpose, the construction of an unfolding (the "puzzle game"), its partial order abstraction, and its final projection onto several testers. Particular attention is paid to the algorithmic complexity and its potential to scale up, in the perspective of developing a real prototype. Some indications of possible future developments are given in the conclusion.

2 Synthesis based on transition systems

We mainly rely on our experience in conformance testing. The TGV tool (Test Generation using the Verification technology) (4; 5), jointly developed by our group at Irisa and a group at Vérimag, is a real-size implementation of synthesis techniques based on transition systems. We thus begin by recalling the main principles.

2.1 Our example

Let us consider the small example depicted in Figure 1. This is a simple connection-disconnection protocol, modelled with two interacting finite automata communicating through one-bounded channels. The user of site A can use the protocol by asking for a new connection (observable event **a**). The disconnection of a previously opened connection can be spontaneously initiated by the protocol entity of site A, or explicitely requested by the user of site B (observable event **c**). Disconnections are locally reported to the users by the observable events $\underline{\mathbf{b}}$ on site A and $\underline{\mathbf{c}}$ on site B. The protocol manages a possible collision of disconnect messages by exchanging a disconnect confirmation, named d. The completion of the disconnection is reported to user A by the observable event $\underline{\mathbf{e}}$, allowing him to ask for a new connection. The events controllable by the users (or testers) are: **a** (Connect_Request), and **c** (Disconnect_Request from B). The observable events are: $\underline{\mathbf{b}}$ (Disconnect_Confirm of A), $\underline{\mathbf{c}}$ (Disconnect_Confirm of B), $\underline{\mathbf{e}}$ (Disconnection_Completed). The others are internal communication (emissions are denoted by !, and receptions by ?). They are neither controllable, nor observable.

The scenario presented in Figure 2 illustrates the three possible repeatable behaviours (connection is closed by the initiator, connection is closed by the other side, and collision).

We consider now a particular test objective that is to check the possibility to complete a disconnection after collision (collision management is known to be a fragile aspect of this kind of protocol). This can be naturally described by the partial order given in Figure 3: $\underline{\mathbf{e}}$ is a consequence of **a** then $\underline{\mathbf{b}}$, in the context in which $\underline{\mathbf{c}}$ occurs concurrently with $\underline{\mathbf{b}}$. We thus consider that the test objective is to select a particular pattern in which these four events occur and are linked by the indicated causal and concurrent relations.

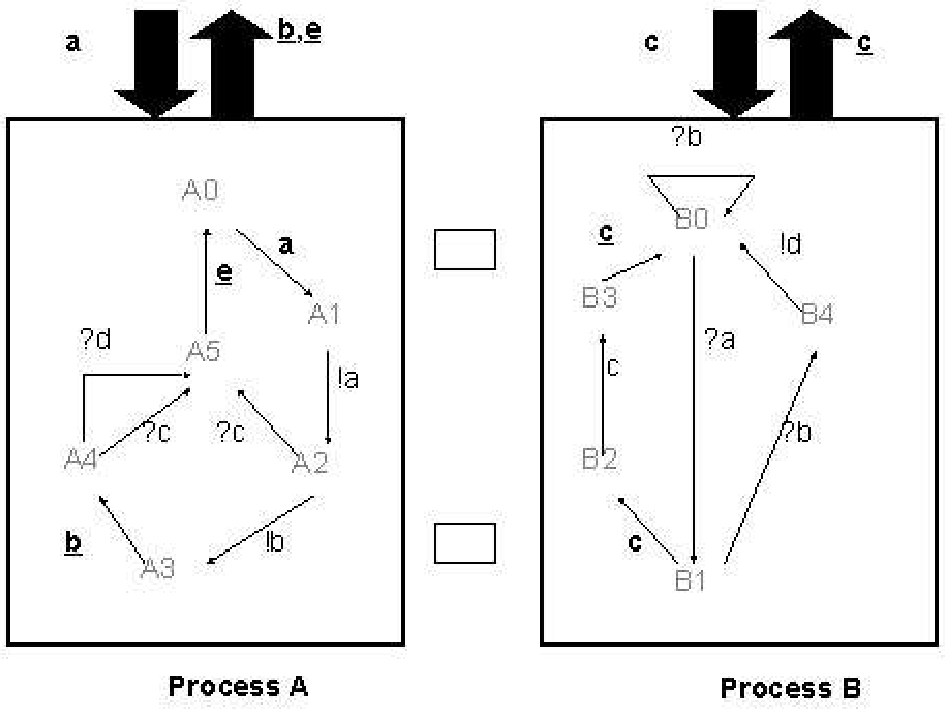

Figure 1: A small example of connection-disconnection protocol.

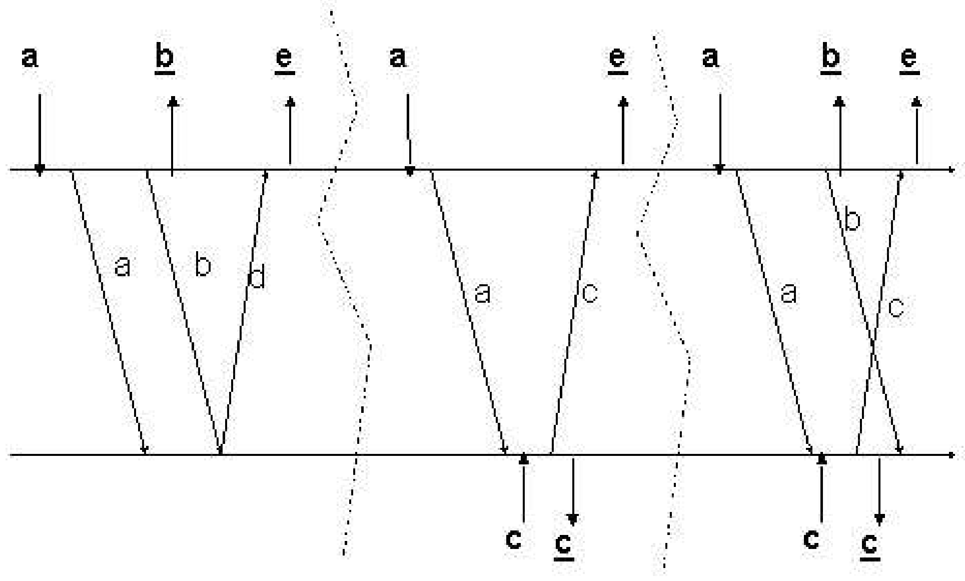

Figure 2: Possible behaviours

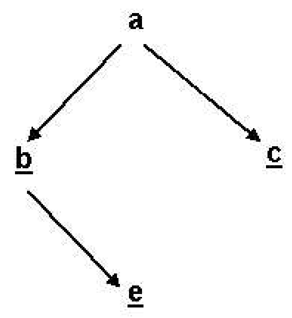

Figure 3: A partial ordered test purpose

The description by communicating finite automata is just for illustration purpose. In the real use of TGV, models are described in higher-level languages like SDL, Lotos or UML (6; 7; 8), the associated compiler providing a simulation code to access the state representation.

2.2 The state-graph representation

A usual way to synthesize test cases for conformance testing of such protocols is to build an automaton from a state-graph representation of all the behaviours of the specification. In such a representation, the behaviours are totally ordered sequences of events. The whole set of behaviours can be coded in a labelled transition system. For instance, in TGV, the state graph is obtained by co-simulation of the specification of the object under test and a test purpose. This graph can be abstracted in order to retain only the sequences of controllable and observable events. The final controllable test case is then extracted. The different steps illustrated in Figure 4 are actually implemented by on-the-fly algorithms using APIs on the different considered graphs (11). The algorithms are mainly based on adaptations of Tarjan's algorithm (10), computing the strongly connected components of a graph via a depth-first search. The complexity is linear in the size of the state graph (though the graph itself may be exponential in the size of the model, notably when the concurrency is significant). Determinisation remains exponential, but is applied to the graph of visible behaviours, which is much smaller than the state graph.

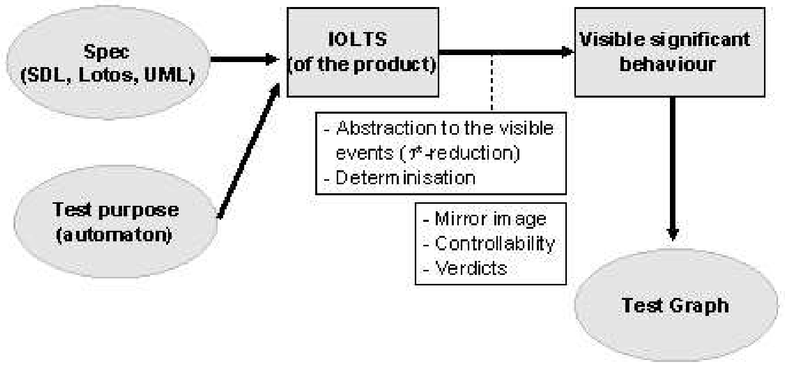

Figure 4: The different steps in TGV

A test purpose in TGV is given as a finite state automaton. We assume that there is one sink state, denoted by $accept$. Its reachability defines the test objective. The other sink states can be labelled by $refuse$. This allows the simulator to select a sub-graph of particular interest, since reaching a "refuse-state" in the product between the specification and the test purpose cuts its traversal. For our example, we propose to interpret the partial order test purpose of Figure 3 as the following automaton of Figure 5. Notice that concurrency is coded as an interleaving and that collision is selected using refuse-states.

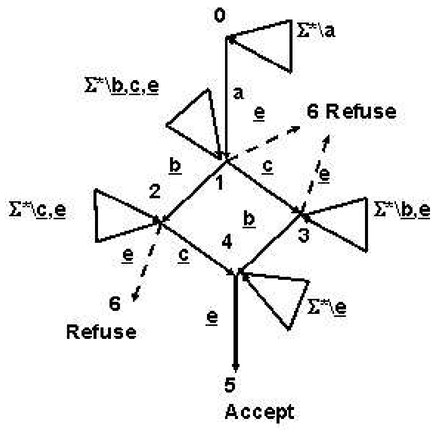

Figure 5: The “collision” test purpose in TGV

The exhaustive simulation of the protocol, guided by the test purpose, starting in the initial state A0B0 with empty channels, and keeping track of the global states reached by the simulator gives the graph depicted in Figure 6. Notice that cycles are built when reaching a previously generated state. This state graph captures all the possible traces of the protocol (the interleaving is computed in the case of concurrent events). From this graph, one can compute a trace-equivalent automaton restricted to the alphabet of visible events.

2.3 Test synthesis

The resulting graph representing all the visible traces of the specification, consistent with the test purpose, defines a set of possible test cases (up to the inversion of interactions). The result on our example is shown in Figure 7 (on the left). In general, to reduce the complexity, one extracts only one test case using some heuristics. For

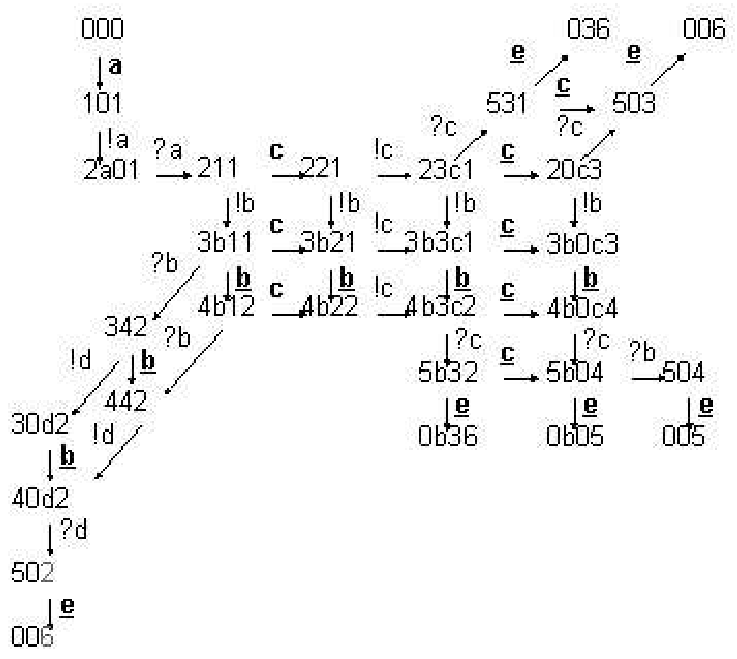

Figure 6: The state graph of the example

example, we consider that a test case must be (globally) controllable, which means that there is no choice between an emission from the tester and another interaction. In our example, the resulting test case is given in the right part of Figure 7. To be complete, we must mention that the test case is augmented with verdicts (PASS in the final state, FAIL for possible receptions that are not in the graph, INCONCLUSIVE when the reception is not on a path selected by the test purpose) and timers (to prevent the test from dead-, live- and output-locks). The generated test cases are guaranteed to be safe (they cannot reject a conformant implementation in the sense of ioco conformance (9)). The method is also complete in the sense that it is able to reject any non-conformant implementation (assuming the provision of a corresponding test purpose, and some fairness assumptions about the implementation).

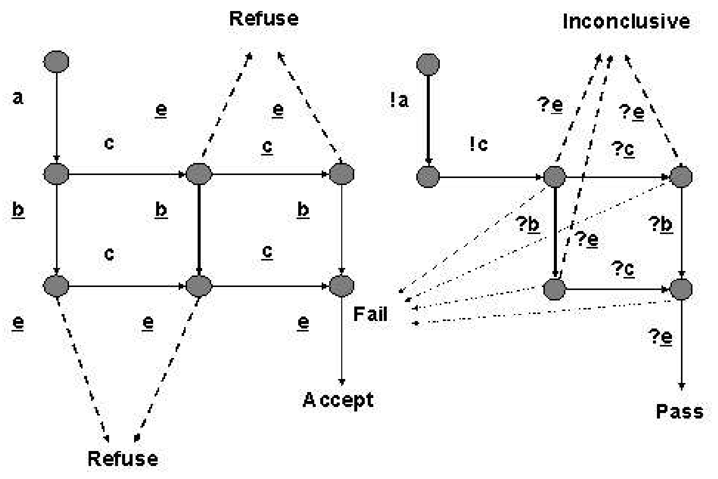

Figure 7: The abstract state graph (on the left) and the resulting controllable test graph (on the right)

Figure 8 gives a few examples of possible test executions when running the test graph of Figure 7 on a real implementation of the protocol.

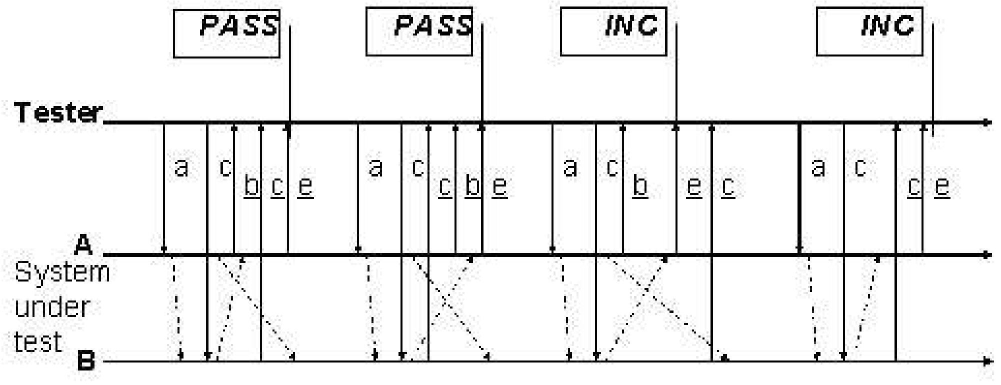

Figure 8: Some possible test executions

3 Partial order view of the specification

3.1 Tile systems

In this section we introduce our mathematical framework. Tiles correspond to partial transitions and a system is defined as a collection of tiles.

Let $\mathcal{V}$ be a finite set of variables. Each variable v takes its values in some finite domain $\mathcal{D}_v$. For $V \subseteq \mathcal{V}$, we set $X_V = \prod_{v \in V} \mathcal{D}_v$. Elements of X_V are denoted by x_V and are called V-states, or local states. For $v \in \mathcal{V}$, we denote by $v(x_V)$ the value of the variable v in state x_V. We shall consider local transitions relating local states, very much in the same way as transitions relate states in standard automata. These local transitions will be referred to as tiles in the sequel. Formally, a tile is a 4-tuple $\tau =< V, {x_V}^-, \alpha, {x_V}^+ >$, where V is a subset of variables, and $({x_V}^-, \alpha, {x_V}^+)$ is a local transition, relating the previous V-state ${x_V}^-$, and performing event α where α ranges over some set A of possible event labels. For τ a tile, we shall sometimes denote by V_τ its set of variables. A system is a triple $\Sigma =< V, X_0, T >$, where V is a finite set of variables, X_0 is a set of initial states, and T is a finite set of tiles and $V = \bigcup_{\tau \in T} V_\tau$. Figure 9 shows the tile system of our example, as it is entered in our prototype. Variables are of enumerated type. ${x_V}^-, \alpha, {x_V}^+$ are introduced respectively by the keywords `pre`, `label` and `post`.

The interleaved sequence of states and events $x_0, \alpha_1, x_1, \alpha_2, ..., \alpha_k, x_k, ...$ is a *run* of system σ if $x_0 \in X_0$ and, for each $k > 0$, there exists $\tau =< V, {x_V}^-, \alpha, {x_V}^+ >$ such that:

1. $\forall v \in V : v(x_{k-1}) = v({x_V}^-), \alpha_k = \alpha, v(x_k) = v({x_V}^+)$ and,
2. $\forall v \notin V : v(x_{k-1}) = v(x_k)$.

Since tiles define local transitions, it may be the case that two successive tiles of a given run involve disjoint sets of variables, i.e. modify different local states. In this case, exchanging the order of the tiles yields to an equivalent run. This is why we will adopt a partial ordering of tiles instead of considering the different runs.

3.2 Guiding by test purposes

Test purposes are a very interesting feature when dealing with large specifications. Their role is to mark out a relevant part of the specification in which a test must

```
% Variables
var A : 0..5 init 0;
    B : 0..4 init 0;
    M : (O,a,b) init O;
    N : (O,c,d) init O;

% Tiles
?A  pre A(0) label ?A post A(1);
?b  pre B(0) M(b) label ?b post B(0) M(0);
!a  pre A(1) M(0) label !a post A(2) M(a);
?a  pre B(0) M(a) label ?a post B(1) M(0);
!b  pre A(2) M(0) label !b post A(3) M(b);
?C  pre B(1) label ?C post B(2);
!B  pre A(3) label ?B post A(4);
!c  pre B(2) N(0) label !c post B(3) N(c);
?c  pre A(2) N(c) label ?c post A(5) N(0);
!C  pre B(3) label !C post B(0);
?b2 pre B(1) M(b) label ?b post B(4) M(0);
!d  pre B(4) N(0) label !d post B(0) N(d);
?c2 pre A(4) N(c) label ?c post A(5) N(0);
?d  pre A(4) N(d) label ?d post A(5) N(0);
!E  pre A(5) label !E post A(0);
```

Figure 9: Tile system of the example: variables A and B contain the local state of the A and B communicating state machines. Variables M and N code the channels (since channels are bounded by one in our example, the possible values are: empty 0, or full with a message a, b, c, or d). Each tile corresponds to a local transition. A tile is defined by a name, a set of pre-conditions giving the expected values of the relevant variables for the transition, a label and a post-condition defining the resulting values of the variable after having put the tile. For example, the tile “?c2” is the second possibility of reveiving a message “c” by machine A provided its current state is 4 (pre-condition A(4)) and there is a message c in transit (N(c)). The receipt will put the system in a new state where machine A is in state 5 and its input channel is empty. The observable events are represented by the tiles named !A, !B, !C or !E. The controllable events are named ?A and ?C.

be found. This concept, present since the beginning in the ISO methodology, is rich enough to continue to arouse discussions in a broader community (15; 16). In TGV, a test purpose is given by a finite automaton with sink states labelled by accept or refuse. A transition of the specification is triggered if there exists a similar transition in the test purpose. It thus allows some transitions to be cut in the state graph representation. The accept state will become the PASS state in the final test case.

This point of view can be easily ported to partial order models, by considering that test purposes are particular tile systems. It is possible to consider a guided tile system of the specification, coding the product $TP \times S$. The principle is as follows: for each tile of the test purpose, let us consider a tile of the specification with a similar event label and build a new tile by making the conjunction of pre-conditions and the conjunction of post-conditions. This can increase the number of tiles. The original tiles of the specification are kept in the system in order to ensure that the specification can evolve even there is no corresponding tile in the test purpose (the analog of the role of self-loops in the TGV representation of test purposes, as depicted in Figure 5). In case of conflict to put a duplicated tile or its original, only the duplicate is considered. The unfolding is carried out on this new tile system. In contrast, putting a tile labelled by $acccept$ or $refuse$ will stop the construction of the unfolding (see the next section, $accept$ or $refuse$ will be considered as cut-off events).

Figure 10 shows the tile system of the test purpose chosen in our example.

```
% Variables
var TA : 0..4, Accept init 0;
    TB : 0..3 init 0;
% Tiles
?A  pre TA(0) TB(0) label ?A post TA(1) TB(1);
!B  pre TA(1) label !B post TA(2);
!C  pre TB(1) label !C post TB(2);
!E  pre TA(2) label !E post TA(3);
Accept pre TA(3) TB(2) label Accept post TA(4) TB(3);
```

Figure 10: A partial ordered test purpose represented as a tile system

3.3 Construction of the unfolding

Given a run, the sequence of successive tiles forms a graph, by superimposing the pre-condition of a tile $x_V{}^-$ onto an equivalent condition in the existing graph (like a puzzle game). This graph contains two types of nodes: the conditions (the different values of the variables used in pre and post conditions of the tiles), and the events of the tiles. Given two nodes n and n' (condition or event), we say that n *causes* n', written $\preceq$, if either $n = n'$ or there is a path of arrows from n to n'. We say that n and n' are in *conflict*, written $n \# n'$, if there is a condition m, different from n and n', from which one can reach n and n', exiting m by different arrows. Finally we say that n and n' are *concurrent* if neither $n \preceq n'$, nor $n' \preceq n$, nor $n \# n'$ hold. A *co-set* is a set of concurrent nodes. From a tile system $\sigma =< V, X_0, T >$, the basic algorithm for the construction of the graph is the following:

```
Puzzle := $X_0$ ;
repeat
    if there exists a tile $\tau$ such that $x_V{}^-$ is a co-set of Puzzle
    then append $\tau$ to Puzzle
forever
```

Figure 11 shows the graph obtained after 8 steps of the above algorithm, considering in sequence the tiles numbered from 1 to 8. Conflict between tiles 6 and 8 is pointed out by the branching from condition A2.

This graph is generally infinite (in the case of infinite behaviour), it has no circuits, every condition has at most one input node, every node has a finite number of predecessors in the graph, and no node is in self-conflict. It is in fact an occurrence net in the framework of Petri nets. We can use the corresponding terminology. A *cut* is a set of conditions c satisfying the following two properties: c is a co-set, and c is maximal (it is not properly included in any other co-set). A *configuration* is a set of nodes k satisfying the two following properties: k is causally closed (if $n \in k$ and $n' \prec n$, then $n' \in k$) and conflict-free (no two nodes of k are in conflict). Furthermore, we require for convenience that all maximal nodes (if any) of configurations shall be conditions. Finite configurations and cuts are closely related. In particular, given a finite configuration k the set of conditions $Cut(k)$ is a reachable global state, which we denote $GS(k)$. The basic algorithm will eventually produce

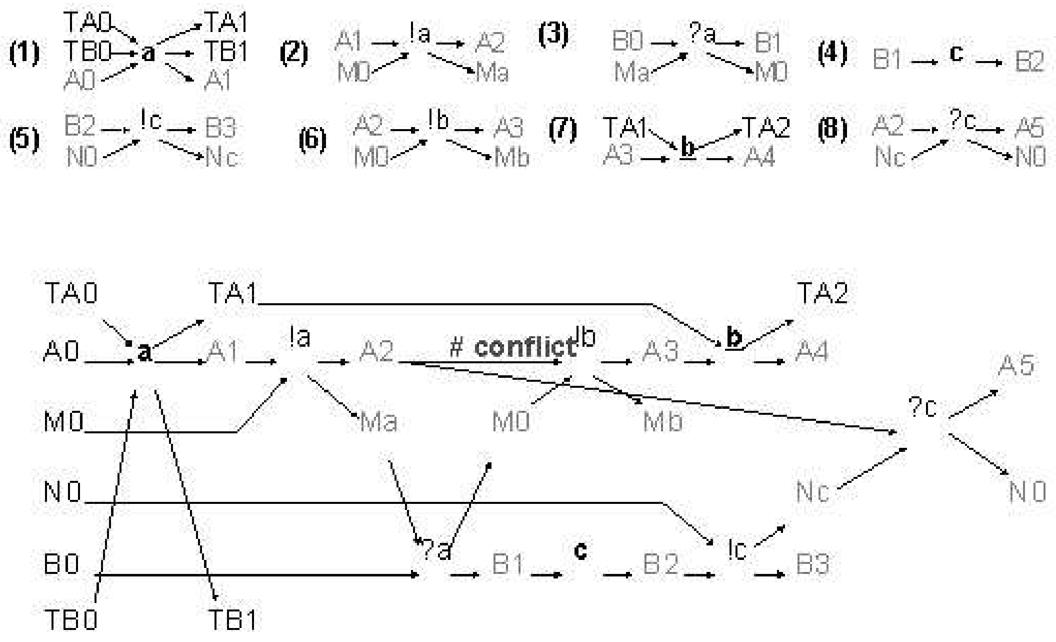

Figure 11: A particular unfolding containing 8 tiles

any reachable global state under only the fairness assumption that every tile candidate to be added is eventually chosen to extend the puzzle (the correctness proof follows from the definitions and from the results of (12)).

It appears that the unfolding is of fractal nature, and can be reduced to a finite generator part, called a finite complete prefix. A prefix U of the unfolding is complete if for every reachable global state S there exists a configuration C in U such that $GS(C) = S$ and for every tile τ enabled by S there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and e is the event node of τ. A complete prefix contains as much information as the unfolding, in the sense that we can construct the unfolding from it as the least fix-point of a concatenation operation on patterns defined by maximal configurations of the prefix. In order to construct such a prefix, the question is to locate the event nodes (called the *cut-off events*) from which the extension in the unfolding can be stopped. We will denote $[e]$ the set of predecessors of e (the set of events e' such that $e' \preceq e$). An event e of the prefix is a cut-off event (with respect to a particular order $<$) if the prefix contains an event e' such that $GS([e]) = GS([e'])$, and $[e'] < [e]$. Let E be the finite set of event nodes of the prefix. The algorithm to construct a finite complete prefix is the following:

```
Finite_Puzzle := $X_0$ ;
cut_off := {} ;
repeat
        Select a tile $\tau$ such that ${x_V}^-$ is a co-set of Finite_Puzzle ;
        live := $\tau$ exists and ${x_V}^- \cap$ cut_off={} ;
        if live then append $\tau$ to Finite_Puzzle ; $v$ denotes this new event node
          if $\exists u < v : GS(u) = GS(v)$ then cut_off := cut_off $\cup {x_V}^+$
until not live
```

The correctness of the algorithm requires that the partial order $<$ be correctly chosen. In (13), it is proved that $<$ must be adequate, that is defined as an order that is well-founded, which refines the set inclusion and which is preserved by finite extensions. The size of the prefix also depends on this order, but it is possible to guarantee that the prefix is never larger than the global reachability graph (states + transitions). The running time of the algorithm is $O(\frac{|C|^\varphi}{\varphi}$, where C is the set of conditions of the prefix, and φ denotes the maximal size of the pre-conditions of the tiles in the original system.

Figure 12 shows the complete prefix of our example as computed by the Esparza-Römer-Vogler's unfolding algorithm (available through the "Model-Checking Kit" of the Technical University of Münich (27)). The slowest part of the algorithm is locating the possible conditions that can be covered by a new tile. This is implemented by coding the concurrency relation and providing a method of maintaining it. This deteriorates as the size of the prefix increases, since the amount of memory needed to store the concurrency relation may be quadratic in the number of conditions in the already built part of the prefix. A recent improvement proposed in (14) structures the set of events in order to speed up the search in practice, not by trying the events one by one, but several at once, merging the common parts of the work.

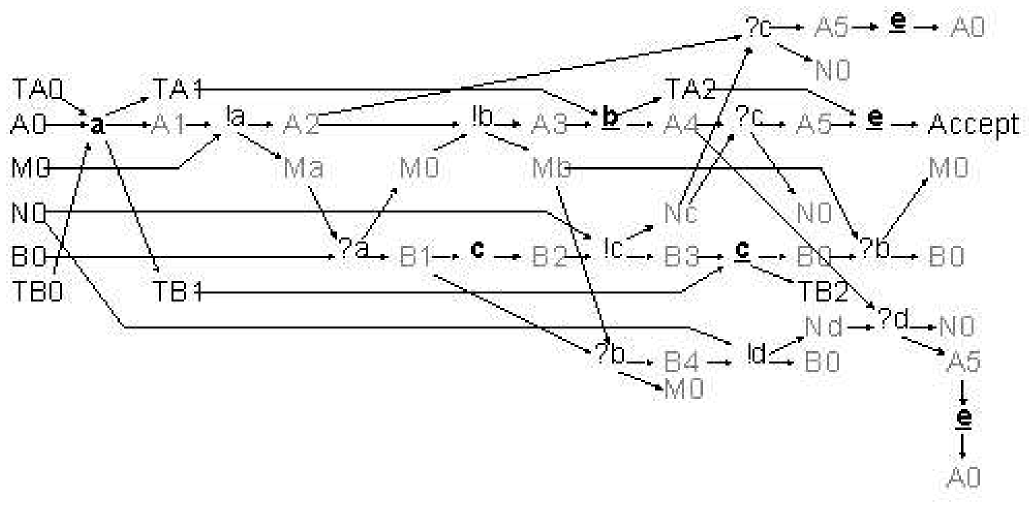

Figure 12: Complete finite prefix of our example

3.4 Abstraction

At this step, the complete prefix contains all the information needed to generate a test case (no further unfolding is needed, since all the realisation of the test purpose has been considered in the new tile system, resulting from the product of the specification with the test purpose). We consider that the relevant semantics for a test case is the partial order of its events. This implies that conditions in the prefix are just intermediary construction "pins", which can be deleted after having finished the construction. This leads to the underlying event structure $< E, \preceq, \# >$, composed of the causal order relation $\preceq$ and of the conflict relation $\#$. The causal relation is transitive and conflict is inherited by causality:

$$\forall e, e' \in E,\ e\#e' \Rightarrow \forall e'', e' \prec e'', e\#e''$$

The deletion of pins can be done in a linear complexity in the size of the unfolding using the following rules, being given two different transitions e and e':

- if there exists a place P of the unfolding, which is both an output place of e and an input place of e', then $e \prec e'$.
- if there exists a place P of the unfolding, which is an input place of both e and e', then $e\#e'$.

The result constitutes a graph (the "ES graph", whose nodes are the events (or transitions), and edges of two types, representing the causal and conflict relations.

The complete computation of the $\prec$ and $\#$ relations demand to compute the transitive closure of the resulting graph, which is of cubic complexity in its simplest form. But abstraction can be directly defined on the ES graph. One usually draws these relations by a covering graph in which black arrows code the pairs of the transitive reduction of the causal relation, and dashed lines between events code the initial conflict. Figure 13 shows the event structure associated with the prefix of the unfolding of our example.

The interest of this representation is that abstraction to the observable (and controllable) events can be easily defined (which is not the case, considering directly the structure of the unfolding). Abstraction (or restriction to the observable events) is just defined as a sub-structure. Let O be the subset of observable and controllable events ($O \subseteq E$). Abstraction $< O, \preceq_a, \#_a >$ is defined by:

$$\forall o, o' \in O,\ o \preceq_a o' \text{ iff } o \preceq o'\ ;\ \ o\#_a o' \text{ iff } o\#o'$$

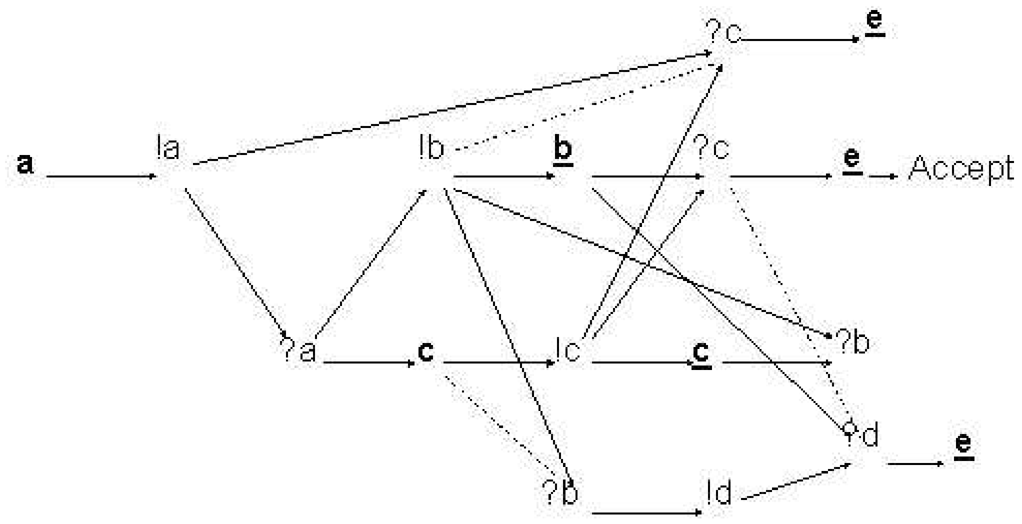

Figure 13: The event structure of our example

The abstract ES graph can be computed in a quadratic complexity in the number of events.

Notice that in general this operation may lead to non-deterministic event structures in which two immediate successors of a node can be labelled by the same action. This can be avoided by considering that actions located on different processes have disjoint labels: two concurrent events have thus different labels (this is naturally ensured when modelling communicating machines). They cannot be in conflict too, if the initial processes are deterministic, since conflicts are locally described. Figure 14 shows the result of the abstraction of the event structure of our example.

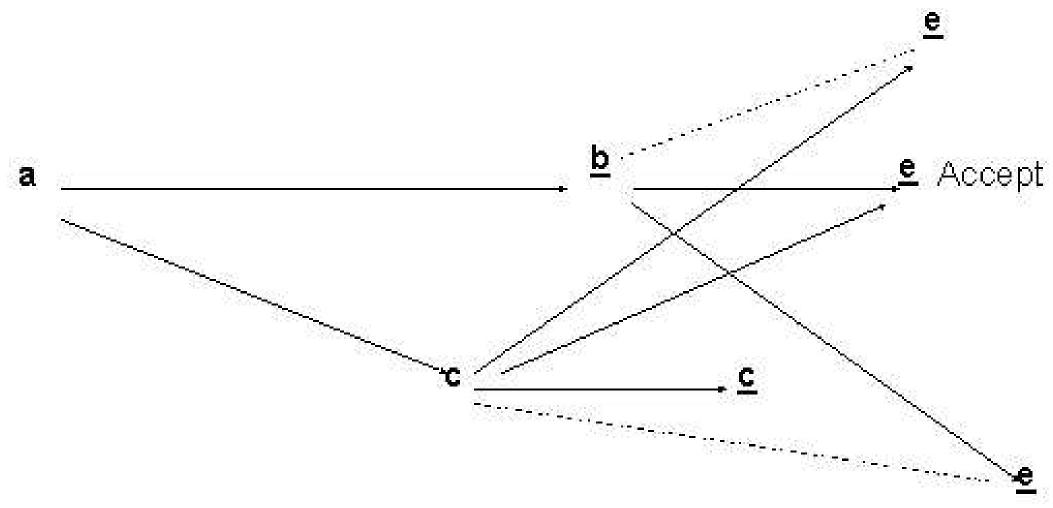

Figure 14: Restriction of the event structure to the observable and controllable events

3.5 Projection

The abstracted event structure is called a *test graph*. The last step of the method is the projection of the test graph onto the different testers. For the sake of simplicity here, we consider that there is one tester associated with each process. The verdict events must be also assigned. We have considered in our example that tester B has the responsability to emit the *Pass* verdict. The principle is the following:

- the events of the testers are the mirror images of the events of the test graph (receptions from the point of view of the specification are the emissions of the testers, and vice versa);
- causality between distant events is implemented by inserting the exchange of a synchronisation message between them. The sending of the message is devoted to the site owning the first event, and the reception will occur on the site of the second event. Before this step, it is advisable to build effectively the transitive reduction of the test graph, in order to avoid redundant synchronisation (this is of cubic complexity, but the algorithm is applied generally on small graphs);
- a local test graph for each tester is obtained by projection of the test graph with synchronisations, keeping only the local events (another sub-structure construction);
- a sequential automaton for each tester can then be obtained by constructing the graph of all the configurations of the local event structures. There exists linear algorithms in the size of the automaton to do that. Of course, the size of the automaton is exponential in the size of the event structure, since all the interleavings have to be computed. In practice, it is expected by construction of the test architecture that this local explosion due to a possible parallelism inside a local tester remains limited.

The technique does not ensure a minimal number of synchronisation messages, since it is not able to discover two different messages play the same role and could be merged (this is the case in our example). This question of optimisation is out of the scope of this paper. But it ensures that causality and concurrency is preserved, that is the primary goal.

By construction, the partial order defined by the test graph is preserved by projection. Thus, all the traces are preserved too [16,17]. The result of the projection for our example is shown on Figure 15.

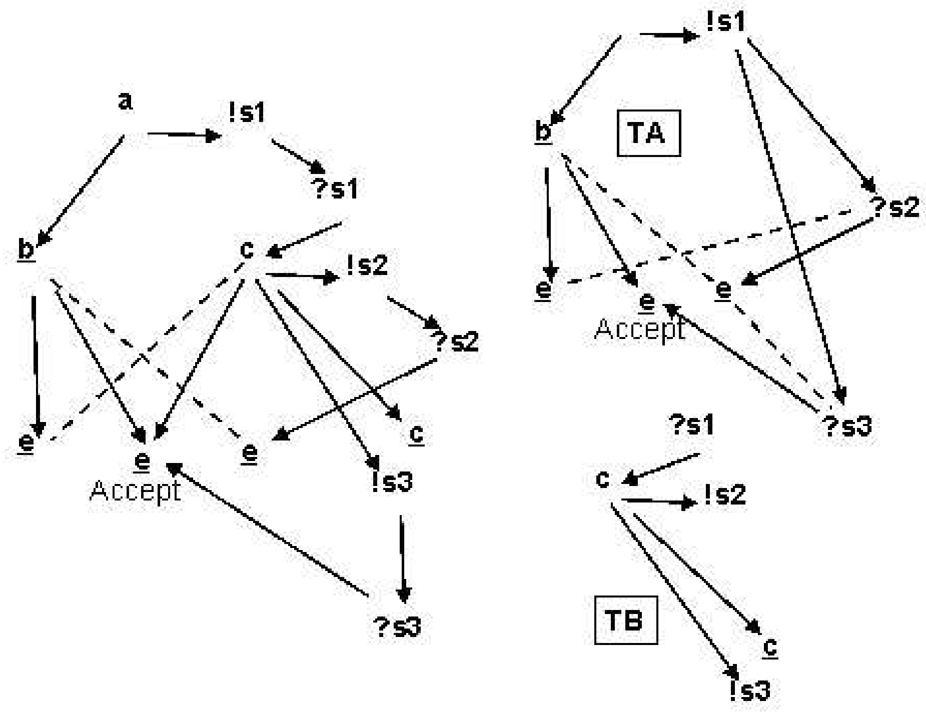

Figure 15: Insertion of synchronisation messages (on the left). Projection onto tester A (TA) and tester B (TB) on the left

We can see that concurrency between the events **b** and **c** in the test purpose is preserved, while the sending of event **c** has been synthesized.

Figure 16 shows the resulting test cases, obtained by computing the covering graph of the configurations (the set of different configurations are ordered by set inclusion, forming a particular partial order, often referred as a “budding lattice”). The exposition of the construction of such graphs is out of scope of this paper. The interest reader can consult (28; 29) to find different algorithms to compute off-line or on-line these graphs in the absence of conflicts. Their extension to take into account conflicts is straightforward.

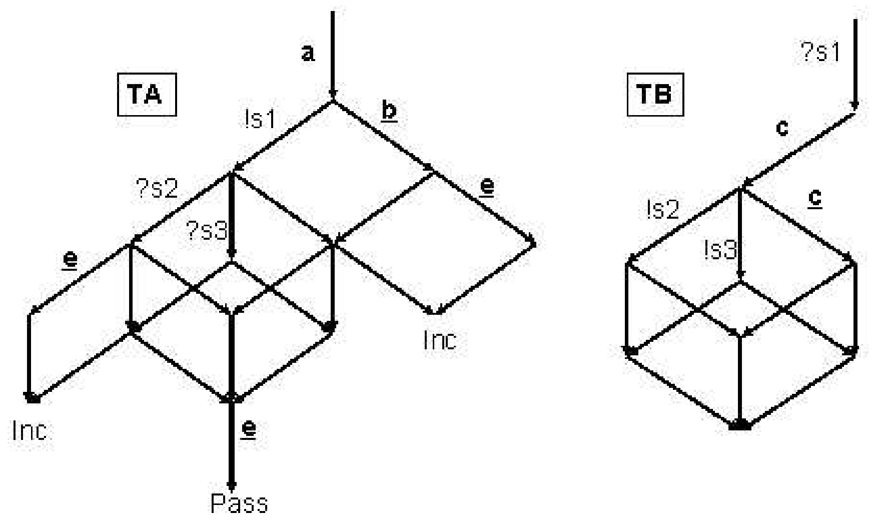

Figure 16: The resulting test cases

The test cases TA and TB could be simplified, considering a “local controllability” in which the testers can decide themselves of a particular total order of synchronisation, avoiding the consideration of all the interleavings between sending (and thus receiving) synchronisation messages.

Figure 17 shows different possible scenarios when applying the testers to the distributed implementation.

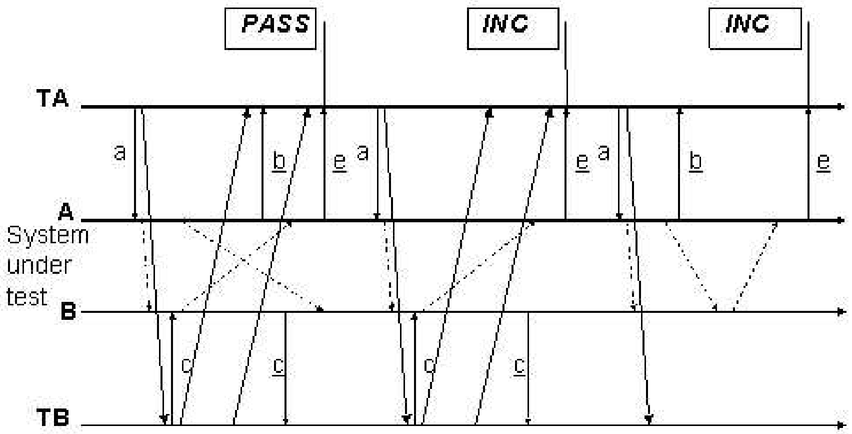

Figure 17: Some possible distributed test scenarios

4 Conclusions and perspectives

Pursuing the approach initiated by Ulrich and König, we propose a complete chain of test synthesis based on a true-concurrency model. This is done by revisiting the TGV methodology based on test purposes and of graph manipulation (product, abstraction, projection). The theoretical basis of our proposal relies on sound and scalable algorithmic, based on:

- the construction of prefixes of unfoldings,
- the computation of sub-structures,
- the computation of transitive reductions, and
- the computation of configurations.

The main perspective is to continue the implementation of these ideas. We have also several research directions to explore:

- The use of UML as modelling language. Beyond its popularity, there is a real challenge to deal with the partial order semantics of UML, as given in the action semantics currently specified. The idea is to consider symbolic tiles in which pre- and post-conditions are symbolic expressions. Most of the steps could resist to this genericity.
- The required algorithms seem to be implementable on-the-fly, like in TGV. But the situation is much more complex on unfoldings than in simple transition systems.
- There are specific questions of controllability in a distributed context (25), which deserve further study.
- Even it is clear that standard formal conformance relation based on the inclusion of sequential traces (9) applies in our approach, it is tempting to refine it to a kind of partial order inclusion. This could be achieved by considering a distributed observation of the communication between the entities under test (26), i.e. by instrumenting the implementation by a vector clock mechanism.

References

[1] C. Jard. *Principles of synthesis of distributed test cases using true-concurrency models*, Testcom'2002. Berlin, March 2002.

[2] ISO/IEC 9646 IT-OSI, *OSI Conformance Testing Methodology and Framework*

[3] M. Törö. *Decision on Tester Configuration for Multiparty Testing*. Proc. of the 12th int. Workshop on Testing of Communicating Systems. Budapest, Hungary, 1999.

[4] JC. Fernandez, C. Jard, T. Jéron and C. Viho. *Using on-the-fly verification techniques for the generation of test suites,* Conference on Computer-Aided Verification (CAV'96), New Brunswick, New Jersey, USA, Alur, A. and Henzinger, T. editors, Springer, LNCS 1102, july 1996.

[5] JC. Fernandez, C. Jard, T. Jéron and C. Viho. *An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology*, Science of Computer Programming, Groote, J.-F. and Rem, M. editors, Elsevier Science, number 29, pages 123-146, 1997

[6] L. Doldi, V. Encontre, JC. Fernandez, T. Jéron, S. Le Bricquir, N. Texier and M. Phalippou. *Assessment of Automatic Generation Methods of Conformance Test Suites in an Industrial Context*, IFIP TC6 9th International Workshop on Testing of Communicating Systems, Baumgarten, B. and Burkhardt, H.-J. and Giessler, A., Chapman & Hall, september 1996.

[7] A. Kerbrat, C. Rodriguez, and Y. Lejeune. *Interconnecting the ObjectGéode and CADP toolsets*. In Proceedings of SDL forum'97. Elsevier Science (North Holland), 1997.

[8] T. Jéron, JM. Jézéquel and A. Le Guennec. *Validation and Test Generation for Object-Oriented Distributed Software*. International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), Tokyo, Japan, April 1998.

[9] J. Tretmans, *Test Generation with Inputs, Outputs and Repetitive Quiescence*. Software, Concepts and Tools (1996) 17: 103-120.

[10] R. Tarjan. *Depth-first Search and Linear Graph Algorithms*. SIAM Journal Computing, 1(2):146-160. June 1972.

[11] T. Jéron and P. Morel. *Test Generation Derived from Model-Checking*. 11th intern. Conf. On Computer Aided Verification (CAV'99), Trento, Italy, July 1999. LNCS 1633, pp. 108-122.

[12] J. Engelfriet. *Branching Processes of Petri Nets*. Acta Informatica 28, pp. 575-591 (1991).

[13] J. Esparza, S. Römer. *An Unfolding Algorithm for Synchronous Products of Transition Systems*. Proc. Concur'99, Springer, LNCS 1664 (1999) 2-20.

[14] V. Khomenko and M. Koutny. *Towards an Efficient Algorithm for Unfolding Petri Nets*. Proc. Concur'2001. Springer, LNCS 2154 (2001): 366-380.

[15] Y. Ledru, L. Du Bousquet, P. Bontron, O. Maury, C. Oriat, and ML. Potet. *Test Purposes: Adapting the Notion of Specification to Testing*. Proc. of the IEEE Automated Software Engineering Conference (ASE'2001). San Diego, November 2001.

[16] RG. De Vries and J. Tretmans. *Towards Formal Test Purposes*. In Formal Approaches to Testing of Software (FATES), Aalbord, Denmark, August 2001.

[17] B. Caillaud, P. Caspi, A. Girault and C. Jard. *Distributing Automata for Asynchronous Network of Processors*. European Journal on Automated Systems (JESA), 31(3): 503-504. May 1997.

[18] C. Jard, T. Jéron, H. Khalouche and C. Viho. *Towards Automatic Distribution of Testers for Distributed Conformance Testing*. Formal Description Techniques and Protocol Specification, Testing and Verification, 18, pp. 353-368. IFIP, Kluwer, November 1998.

[19] G. Luo, R. Dssouli, Gv. Bochmann, P. Venkaratan and A. Ghedamsi. *Test Generation with respect to Distributed Interfaces*. Computer Standards and Interfaces 16 (1994): 119-132.

[20] R. Castanet and O. Koné. *Deriving Co-ordinated Testers for Interoperability*. Protocol Test Systems, VI (C-19), O. Rafiq (Ed). Elsevier Science B.V. (North-Holland). 1994 IFIP.

[21] M. Kim, S.T. Chanson, S. Kang and J. Shin. *An Approach for Testing Asynchronous Communicating Systems*. Proc. of the 9th int. Workshop on Testing of Communicating Systems. Darmstadt, 1996, pp. 141-155.

[22] M. Kim, J. Shin, S.T. Chanson and S. Kang. *An Enhanced Model for Testing Asynchronous Communicating Systems*. Formal Description Techniques and Protocol Specification, Testing and Verification, 19, pp. 337-355. IFIP 1999. Beijing, China.

[23] A. Ulrich and H. König. *Specification-based Testing of Concurrent Systems*. Formal Description Techniques and Protocol Specification, Testing and Verification, 17. T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.), 1997 IFIP. Published by Chapman & Hall.

[24] O. Henniger. *On Test Case Generation from Asynchronously Communicating State Machines*. Testing of Communicating Systems. Vol. 10. M. Kim, S. Kang & Al. (Eds). Chapman & Hall, pp. 255-271, September 1997.

[25] A. Ulrich and H. König. *Architectures for Testing Distributed Systems*. Proc. of the 12th int. Workshop on Testing of Communicating Systems. Budapest, Hungary, 1999, pp. 93-108.

[26] L. Cacciari and O. Rafiq. *Controllability and Observability in Distributed Testing*. Information and Software Technology 41 (1999): 767-780. Elsevier.

[27] http://wwwbrauer.informatik.tu-muenchen.de/gruppen/theorie/KIT/

[28] R. Bonnet and M. Pouzet. *Linear extension of ordered sets*. In I. Rival, editor, Ordered Sets, pp. 125-170, D. Reidel Publishing Company, 1982.

[29] C. Jard, GV. Jourdan and JX. Rampon. *On-line Computations of the Ideal Lattice of Posets*. RAIRO ITA, Theoretical Informatics and Applications, Vol. 29, Nu. 3, pp. 227-244, 1995.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399