I N F O R M A T I K

Strategies for Conformance Testing

Solofo Ramangalahy

MPI–I–98–010 May 1998

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Im Stadtwald 66123 Saarbrücken Germany

Author's Address

Max-Planck-Institut für Informatik, im Stadtwald
D-66123 Saarbrücken Germany
solofo@mpi-sb.mpg.de

Acknowledgements

This work was supported by a grant from EDF: *Électricité de France*, Research and Development Group, and by the Max -Planck-Institut für Informatik. I would like to thank Harald Ganzinger for having hosted me in his group. I would also like to thank Sergei Vorobyov for having read an unreadable first version of this paper, correcting hundred of mistakes, and providing me valuable comments. Thanks go to Florent and Fritz.

Abstract

A new test generation method and algorithm for conformance testing is proposed. It is based on the interpretation of testing concepts from the ISO standard "Formal methods in conformance testing" in a game theory setting. A testing game is defined with a specification given as an Input/Output State Machine and a test purpose for test selection. A winning strategy for this game defines a tester for a class of implementations and a conformance relation.

Keywords

formal methods in conformance testing, test purposes, games strategies, test assumptions, input-output state machines.

1 Introduction

Correctness has become a more important issue for "software systems" as they are now of increasing complexity and in wider use. As software systems are assigned critical tasks—where failure can be more expensive than conception itself—the correctness problem has attracted much attention in the computer science community. One way to achieve correctness is to use *formal methods*. These enable a precise definition of the notion of correctness and the development of techniques to check it. Two of these methods are verification and formal conformance testing.

Verification is a method which uses a mathematical model of the system and a set of required properties for this model. Both model and properties are expressed in a formal language allowing to prove (in the mathematical sense) the correctness of the model with respect to the properties.

Formal conformance testing checks the correctness directly on the actual system by experimenting with it with *tests* (or *test cases*). During the execution of the tests, observations are made and verdicts about conformance can be produced. These verdicts are issued by a comparison of the actual observations with the expected ones described by the formal specification of the system—which is supposed to be correct.

Conformance testing and verification are complementary approaches to check the correctness of software systems: verification can prove correctness, but the proof is only valid if the model is appropriate for the real system. By contrast, testing can be performed on the real system, but can give certainty only about the non-correctness of the system with respect to its specification. Verification takes place in the early stages of development, whereas conformance testing comes at the end.

In this report, we will focus on formal conformance testing. Two important characteristics of the systems we study should be noted:

1. The systems are *open*. This means that the evolution of the system is dependent on choices, which are external to the system (resolved by the environment in which the system is used). This contrasts with verification in which the model of a system is usually closed[1].

2. The system is viewed as a *black-box* exhibiting behaviors (sending messages) and interacting with an environment. The system is a real object, hence, to reason formally about it, we assume that it can be mathematically modeled. Note that only the existence of the model is supposed, the actual model is unknown. This is referred to as the *test*

[1]See [33] for a survey of verification of open systems.

hypothesis [2, 30, 16]. Using this hypothesis, we will not always distinguish the (actual) implementation from the implementation model.

Formal conformance testing is the problem of relating a formal specification (as opposed to a natural language specification) and the unknown model of the implementation via a relation called the *implementation relation*. This problem is solved by a test generation algorithm producing a set of test cases (the *test suite*) starting from the specification. The test suite must be *sound*, i.e. experimenting the implementation with these tests should produce a negative verdict only if the implementation is not in relation with the specification via the implementation relation. In this case, it is said that the implementation *does not conform* to the specification w.r.t the implementation relation. Reciprocally, the test suite should have a high "probability" to produce a negative verdict if the implementation does not conform to the specification. Certainty is not possible here, because, contrary to verification, one cannot reason about infinite computations of the implementation (model) seen as a black-box. Testing is restricted to the observation of finite computations of a system.

In this report, we construct *testers* (a compact representation of the test suite) for implementation models and specifications described by automata with inputs and outputs[2]. The testers are also automata, their unfoldings to trees give the test cases of the test suite. These testers will check conformance as defined by the international ISO/ITU-T standard "Formal Methods in Conformance Testing" (FMCT) [16], with *test selection* (the choice of a tester among a set of possible ones) performed using *test purposes* and explicit use of *test hypotheses* as advocated in [2, 24] for helping test generation.

As testing is a broad subject, we will only briefly survey some steps relevant to the formal approach to conformance testing. In recent years, progress has been made toward automation of test case generation in the area of conformance testing via the use of formal description techniques (FDTs). The first steps were the standardization of formal specification languages: SDL [4], LOTOS [14], and Estelle [13]. Then the ISO 9646 standard [15] gave a "methodology and framework for conformance testing". This framework is not related to any particular specification language, but is rather a rationalization of the test experts knowledge. Recently the ISO/ITU-T standard "Framework: Formal Methods in Conformance Testing" (FMCT) reinterpreted concepts appearing in ISO 9646 from a formal point of view, taking advantage of FDTs, and making a synthesis of recent advances in research.[3].

[2]Similar to the classical Moore or Mealy machines [12].

[3]See [3] for a presentation of this work, and references for research background.

The FMCT standard, although providing interpretation of terminology in the standardized FDTs and guidelines on test generation methods, does not prescribe any formal notation nor a test generation method: finding algorithms for conformance testing is still an active field of research.

Here, the test generation method proposed is based on game theory. Some examples of the use of game theory relevant to the correctness problem are the synthesis of a correct program starting from a specification [34, 20] or model checking for the μ-calculus [26, 27, 22]. Use of games for testing has not been much investigated.

Interpreting Nerode-Yakhnis-Yakhnis view of the synthesis problem [21][4] in a test setting, we can view test execution as a game played between the tester and the implementation. The implementation wins if it can keep its bugs hidden from the tester. Conversely, the tester wins if it has been capable of revealing these subtly hidden mistakes in the implementation. We will take the point of view that a game represents the test execution process. Two players — Black player: the environment and Red player: the implementation — are playing again each other, each of them taking alternatively the control of the test execution process. When the environment has the control, it chooses which message to send to the implementation to change the state of the system (external control). Conversely when the implementation has the control it chooses how to change its internal state and sends a message back to the environment (internal control). We want the environment to drive the system in a certain state to emit a verdict mainly "fail" which will give a global verdict of non-conformance. A "pass" verdict does not allow coming to a firm conclusion about conformance but plays a role in increasing confidence in conformance.

To reach these states and emit a verdict, we construct a strategy, that is a function which chooses which message to send to the implementation so that the system goes to the desired states. This strategy is a winning strategy for the environment and defines the *tester*.

This is the key idea of this report. With hindsight, one can say that, given the connection between games and verification, and verification and testing, there exists a connection between games and testing by transitivity. The connection is dual to the one of the synthesis problem: the winning strategy for the implementation in the synthesis problem is such that whatever the environment does, the implementation will not fail. Dually in testing, if an implementation can fail, the winning strategy for the environment (i.e. the tester) will finally make it fail.

The work which is the closest related to ours is [7]. In this work, a veri-

[4]Roots of this view can already be found in [5].

fication technique (on-the-fly technique) is used to solve the test generation problem. Differences with [7], besides the use of game theory, are the suppression of the inconclusive verdict, the suppression of the possibility to emit different verdicts for the same test case and the verification of a stronger implementation relation.

The rest of this paper is as follows: first we will recall some testing concepts from the FMCT standard relevant to this paper. We will instantiate the framework by choosing the class of specifications and implementation models, and the conformance relations as required by the standard. We will briefly present game concepts to introduce the setting of testing from a game point of view. The main part of this paper is the construction of winning strategies for the defined games. We will then examine some complexity aspects of the construction of the tester and its use for conformance testing. In the last part we will establish some properties related to the soundness and exhaustiveness of the generated tester with respect to the implementation relation defining the meaning of conformance.

2 Testing and Games

2.1 Test concepts

Conformance testing is defined by a relation R between two sets of models: the implementation models MODS and the specifications SPECS. The *test hypothesis* [2, 30, 16] is the assumption that the real implementation IUT (Implementation Under Test) can be modeled by an element m_{IUT} of MODS.

Given a specification $S \in \mathsf{SPECS}$ and a conformance relation R, the *conformance testing problem* is to check that the model m_{IUT} belongs to the set of implementation models that are conform to S ($\{m \in \mathsf{MODS} | (m, S) \in R\}$). This is done by generating a set of test cases out of the set TESTS. Test execution is modeled by the parallel product of a test case t and m_{IUT}. The traces of the parallel product are the observations that can be made from the test execution. From these observations, a verdict can be emitted about conformance.

We do not construct the set of conforming models, nor do we know m_{IUT}. By testing experiments we are just able to observe some traces of m_{IUT}; that is sufficient in order to check non-conformance (fail verdict, $(m_{IUT}, S) \notin R$). Conformance (pass verdict) relies on the hypothesis that we observed enough traces of m_{IUT} to be confident in conformance. Although it would be possible in theory to observe all the traces (it would amount to run an infinite number of test cases), it is not possible in practice.

In order to use the FMCT framework, one has to choose a model for R, MODS, SPECS, and TESTS. Motivated by application to an Estelle-like language, we choose:

- Input/Output state machines on a same alphabet as a model for specifications, implementations models, and testers, as in [24] (but we will add special restrictions on MODS and TESTS, $\mathsf{MODS} \subsetneq \mathsf{SPECS}$ and $\mathsf{TESTS} \subsetneq \mathsf{SPECS}$).

- R_5 as implementation relation ([24], see also [16], Annex A). This relation deals with the equality between the outputs allowed by the implementation models and the specification after a trace belonging to both models.

Definition 1 (Input/Output State Machine, IOSM) An IOSM is a quadruple $\langle S, L, T, s_0 \rangle$, where:

- S is a finite non-empty set of states;

- L is a finite non-empty alphabet of interactions. The symbols $?, !, \tau$ are additional symbols not belonging to this set;

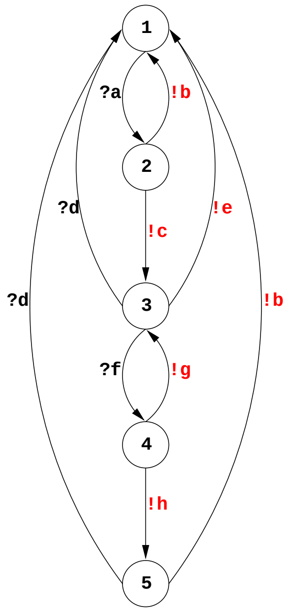
- $T \subseteq \Big(S \times \big((\{?, !\} \cdot L) \cup \{\tau\}\big) \times S\Big)$ is the transition relation. Each element of T is a transition, from an origin state to a destination state. This transition is associated either to an observable action (input ?a or output !a), or to the internal action τ.

- s_0 is the initial state of the IOSM.

Example 1 In the example of Figure 1, we have: $S = \{1, 2, 3, 4, 5\}$ and $L = \{a, b, c, d, e, f, g, h\}$. It may help to see this IOSM of Figure 1 as a specification of a phone. When the user introduces its card (`?a`) the machine can respond that the card is not valid – there being not enough credit on it, it being damaged card ... – or it can accept it (`!b` resp. `!c`). Then the user can dial a number (`?f`). If he waits too long, the phone will give the card back (`!e`). If he can reach his correspondent, he can talk (`!h`), otherwise he has to enter another number or have another try (`!g`). The user can hang up (`?d`), and the phone can stop the communication when his credit runs out or another problem occurs (second `!b`).

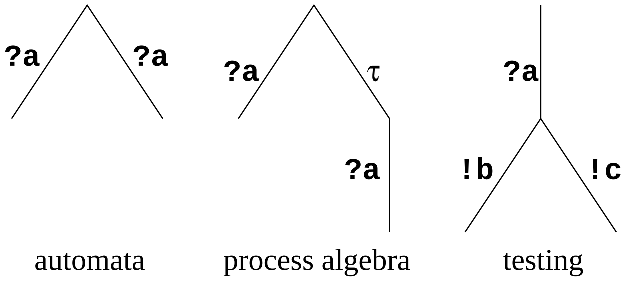
The IOSM model can be used to express the semantics of Estelle specifications. As a model for implementations we choose a special kind of IOSM, where all input actions are enabled in each state ($\forall a \in L, \forall s_1 \in$

Figure 1 an Input Output State Machine



$S, \exists s_2(s_1, ?a, s_2) \in E)$, meaning that an implementation should always accept an input even if it does not react ($s_1 = s_2$). This is called the *input-enabling* condition. This is similar to the IOTS model of [31] or Input/Output Automata [18]. From the testing point of view, one can assume that the IOSMs are without τ and deterministic in the usual sense of automata theory [12] because the power of testing is limited to the observation of traces ([24] chapter 2). In Figure 2 we see examples of three kinds of nondeterminism, for us nondeterminism (except explicitly stated) will mean nondeterminism in the testing sense, that is several outputs (possibly different) are emitted in response to an input.

Figure 2 three kinds of nondeterminism



We also suppose that the deadlocks are observable outputs of the implementation model: in theory this amounts to adding transitions to the automata and in practice to using timers. Finally, we suppose that specifications viewed as graphs are strongly connected to avoid the hypothesis of the existence of a *reset*. With these hypotheses, test execution can only deadlocks because of the choice of the tester.

Definition 2 (implementation relation) Let I and S be two IOSMs (the implementation model and the specification, $Tr(S)$ denote the set of traces of S, and $\mathcal{O}(\sigma, S)$ be the set of allowed outputs by S after the trace σ. Define

$$R_5(I, S) \text{ iff } \forall \sigma \in Tr(S) : \sigma \in Tr(I) \Rightarrow \mathcal{O}(\sigma, I) = \mathcal{O}(\sigma, S).$$

An important problem in test generation is test selection. It deals with the reduction of the size of the test suites to lower the test effort. One way to achieve this selection is to use *test purposes*. The use of test purpose originates in test practice: human test experts write the "meaning" of a test case as a comment in the test suites written in the test notation language TTCN.[5]. The test purpose here is used to limit the search in the state space by focusing on a certain part of the system. The formal definition of a test purpose associated with a test in FMCT is the set of all implementation models which pass the test. However it is not prescribed which of the test purpose or the test follows from the other. Here the test purpose will serve to select test cases. There are different formalisms for test purposes (Message Sequence Charts for example in [10]), we use one comparable to the one of [7].

Definition 3 (test purpose) A test purpose TP associated with a specification $S = \langle S^S, L, T^S, s_0 \rangle$ is a finite subtree of S with out-degree 1 for edges labeled with input actions. This is an acyclic IOSM $TP = \langle S^{TP}, L, T^{TP}, tp_0 \rangle$ such that:

- $\forall (s_1^{TP}, a, s_2^{TP}) \in T^{TP}, \exists s_1^S, s_2^S \in S^S : (s_1^S, a, s_2^S) \in T^S$ (this condition is the compatibility with the specification S);
- $(s_1^{TP}, ?a, s_2^{TP}), (s_1^{TP}, ?a, s_3^{TP}) \in T^{TP} \Rightarrow s_2^{TP} = s_3^{TP}$.

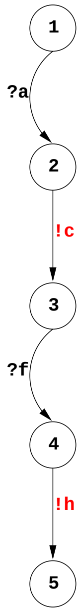
The condition on the out-degree is just a convention not to add further nondeterminism in the testing process. The compatibility between the test purpose and the specification is referred to as consistency in [7], with a further

[5] "[a test purpose is a] prose description of a well defined objective of testing, focusing on a single conformance requirement or a set of related conformance requirements as specified in the appropriate notation" [16]

reachability condition (taken into account in our case via the connectivity hypothesis).

Example 2 Figure 3 shows a test purpose. If we follow the informal meaning given to the specification of Figure 1, the meaning of this test purpose is a scenario for a successful phone call. The scenario is as follows: the user introduces his phone card (`?a`), the phone accepts it (`!c`), the user dials a number (`?f`), and reaches his correspondant (`!h`).

Figure 3 a test purpose for the IOSM of figure 1



A *tester* is also an IOSM. Test execution will be modelled by a parallel execution of a tester and the implementation model m_{IUT}. Observation of the traces of this parallel product will allow us to emit verdicts on conformance. This will be further defined in the rest of the paper.

2.2 Games

For the purposes of this paper, we can restrict ourselves to view games as board games, like chess.

Two players play alternatively on the board by moving pieces, having full knowledge of the previous moves[6]. The position of the pieces on the

[6]This is referred in the litterature as "games of perfect information"

board defines a *configuration*. Moving a piece according to the rules leads to another configuration chosen from several possible ones. We can then view the game as a finite bipartite graph with vertices representing the different configurations of the game, and whith edges representing possible moves. The vertices have two colors, Black and Red, and it is up to the player associated with the color of the vertex to choose the next move (as the graph is bipartite, the players play in turn). A player wins when certain configurations are reached. A play is an infinite sequence of moves. For technical reasons, it is easier to consider infinite plays, but we keep in mind that we are interested in finite plays and that finite plays are a special case of infinite plays.

As explained in the introduction, we view the testing process as a game between two opponents: the implementation (Red player) and the environment (Black player). The system is represented by a bipartite graph with black and red vertices. A token is moved along the edges of the graph by the player whose turn is defined by the color the token is currently on. A play is an infinite sequence of the vertices passed by the token. A play is winning if it belongs to a predefined set of plays, and a strategy for a player is a function giving the choice for the next move of the token depending on the previous moves.

Definition 4 (Game) A *game* is a bipartite, finite, directed graph $G = (V, V_B, V_R, E)$ where:

- V is the set of vertices partitioned into black vertices (V_B) and red vertices (V_R),
- E is the set of directed edges.

A *play* of a game is an infinite word of V^∞, $w = s_1 s_2 \ldots$ such that $\forall i, (s_i, s_{i+1}) \in E$.
Fix a set of plays: the *winning plays*. A play is *winning* for Black if it belongs to the set of winning plays (otherwise it is winning for Red).
A *strategy* for Black is a function $f : V^* V_B \to V$. A strategy for Black is *winning* if Black wins when following this strategy.

Definition 5 (Game associated with a specification and a test purpose) Let $\mathcal{S} = \langle S, L, T, s_0 \rangle$ be a specification and TP a test purpose. The game $G = (V, V_B, V_R, E)$ associated with S and TP is defined as follows:

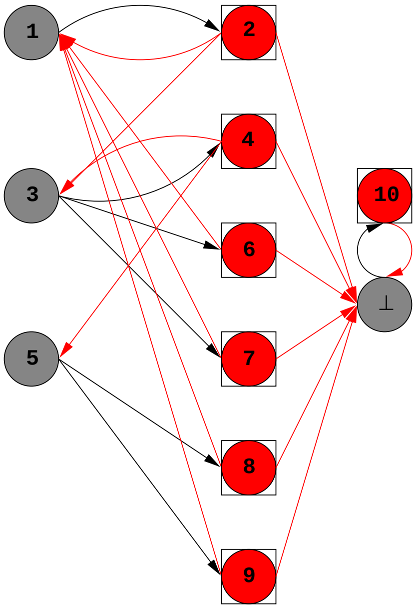
- $V = S \cup \bot$, $V_R = \{s \in S | \forall (s, t, s') \in T, \exists a \in L. t = !a\}$, $V_B = S \backslash V_R$;
- $\forall s \in V_R : (s, \bot) \in E$;

- the winning plays are the plays in $s_0V^*\perp V^\omega$ and $s_0V^*wV^*s_0V^\omega$, where w is a branch of TP.

The choice of red nodes reflect the convention that the tester has the control in an input vs. output action in the parallel execution of the tester and the implementation model. Without loss of generality, we may suppose that the graph is bipartite (the graph can be transformed into a bipartite graph by possibly adding linearly many states).

Example 3 Here is the game associated with the IOSM of example 1. The node $\perp$ will represent a state where non-conformance is detected. Black nodes are depicted by circles, red ones by squared circles. Note the additional nodes 6, 7, 8 and 9 used to transform the graph into a bipartite one, and 10 connected to $\perp$ to ensure that there are always outgoing edges. The test purpose is the one of Figure 3, so the winning plays are those of $1V^*\perp V^\omega$ and $1V^*12345V^*1V^\omega$

Figure 4 game associated with the IOSM of figure 1



This setting allows us from now on to reason only on games.

3 Constructing winning strategies

3.1 Reachability game

We first solve the problem of constructing a winning strategy for a simpler kind of game, called the "reachability game for t"[7], where the winning plays are the plays in $s_0V^*tV^\omega,\ t \in V$. The goal for Black is to reach the node t. For this, we construct an increasing sequence of set of states.

$$
\begin{aligned}
\mathcal{W}_0^0 &= t,\\
\mathcal{W}_{i+1}^j &= \mathcal{W}_i^j \cup \{p \in V_B \mid \exists q \in \mathcal{W}_i^j : (p,q) \in E\}\\
&\qquad \cup \{p \in V_R \mid \forall q \in V_B : (p,q) \in E \Rightarrow q \in \mathcal{W}_i^j\},\\
\mathcal{W}_0^{j+1} &= \mathcal{W}^j_{\min\{i \mid \mathcal{W}_{i+1}^j = \mathcal{W}_i^j\}}\\
&\quad \cup \{p \in V_R \backslash \mathcal{W}\min\{i \mid \mathcal{W}_{i+1}^j = \mathcal{W}_i^j\}^j \mid \exists q, r \in V_B : (p,q),(p,r) \in E \wedge q \in \lim_i \mathcal{W}_i^j\\
&\qquad\qquad \wedge\, r \notin \lim_i \mathcal{W}_i^j\}.
\end{aligned}
$$

These sets form a sequence $\mathcal{W}_0^0, \ldots, \mathcal{W}_{k_0}^0, \mathcal{W}_0^1, \ldots, \mathcal{W}_{k_1}^1, \ldots, \mathcal{W}_0^l, \ldots, \mathcal{W}_{k_l}^l$ where $k_j = \min\{i \mid \mathcal{W}_{i+1}^j = \mathcal{W}_i^j\}$. They are used to define an order on the states via their *rank*. The rank of a state p is:

$$rank(p) = \big(\min\{i|p \in \mathcal{W}_i^j\}, \min\{j \mid p \in \mathcal{W}_i^j\}\big).$$

The strategy for Black is defined by choosing one red node which is inferior with respect to the first component of the rank. This strategy is called "decreasing the rank". This lemma states that this strategy is defined everywhere[8] and uniquely.

Lemma 1 *The rank is defined on all the nodes of the games, and each node belong to one set of a partition of the nodes:*

- $\lim_{i,j} \mathcal{W}_i^j = S \backslash \perp$
- *The sets* $\{\mathcal{W}_{i+1}^j \backslash \mathcal{W}_i^j \mid i \in [0, k_j], j \in [0, l]\}$ *and* $\{\mathcal{W}_{k_i}^i \mid i \in [0, l]\}$ *form partitions of* $S \backslash \{\perp\}$.

Proof
- Let s_n be an element of $S \backslash \{\perp\}$. $S \backslash \{\perp\}$ being a strongly connected component of G, there exists a path $\omega = s_n s_{n-1} \ldots s_1 s_0$ from s to s_0. By recurrence, each of the s_i belongs to one of the $\mathcal{W}_l^k$: s_n

[7]This is an extension of the guarantee game presented in [29].
[8]We don't need to define a strategy on $\perp$

belongs to $\mathcal{W}_0^0$ and if $s_i \in \mathcal{W}_l^k$ and s_{i+1} is black then it belongs to $\mathcal{W}_l^k$ or s_{i+1} is red and all its ongoing edges go into $\mathcal{W}_l^k$; otherwise it belongs to $\mathcal{W}_l^{k+1}$.

- disjointness comes from construction.

In spite of the fact that the implementation must be seen as a black-box, there are several kinds of hypotheses that can be made to help in generating test cases. The notion of test hypotheses originates from [2] in the context of algebraic specifications, and has been extensively studied in [24] for conformance testing. Their effect is to reduce the set of conforming implementation models. Their choice is made from assumptions on the actual implementations. In the original context of games, the problem is to find the winning sets for each player (i.e., the set of states from which the player is assured to win if he follows a good strategy) and the associated winning strategies. Here we are interested in having Black always winning, possibly by adding constraints on Red moves. We allow Red to play only "fair" plays (fairness in the sense used for example in fair transition systems). The intuitive meaning of fairness is that if an implementation can exhibit several behaviors nondeterministically it will eventually show all of them. Formally:

Definition 6 (fair plays) Let $\pi = s_1 \dots s_n \dots$ be a play, $\mathit{Inf}(\pi) = \{s \mid$ s appears infinitely many times in $\pi\}$. Let $\{(r_i, B_i = \{b_{i_1}, \dots, b_{i_j}\}) \mid (r_i, b_{i_k}) \in E\} \subset V_R \times \mathcal{P}(V_B)$ be the sets of black nodes b_{i_k} adjacent to a red node r_i. The play π is *fair* if $\forall i : r_i \in \mathit{Inf}(\pi) \Rightarrow B_i \subset \mathit{Inf}(\pi)$.

This notion of fairness corresponds to the notion of fairness (also defined as strong fairness or compassion) or Street acceptance condition for automata on infinite words (see [28] or [19]). It states that if the implementation has infinitely many times the choice between some actions it will execute infinitely many times each of the actions. This is a weak hypothesis in the context of testing, the strongest version being the bounded fairness hypothesis where all the actions will be executed in a finite number of times known in advance. A *fair strategy* is a strategy that cannot lead to unfair plays.

Proposition 1 *The strategy "decrease the rank" is a winning strategy for Black against Red playing a fair strategy for the simple reachability game.*

Proof By contradiction. Let π a winning play for Red. Let $r_1, \dots, r_n$ be the minimal elements of $\mathit{Inf}(\pi) \cap V_R$ with respect to their rank. These elements belong to a set $\mathcal{W}_0^i$. By definition of fairness, as each of the r_i are connected to at least one b of $\mathcal{W}_{k_{i-1}}^{i-1}$, there will be at least one $b \in \mathit{Inf}(\pi) \cap \mathcal{W}_{k_{i-1}}^{i-1}$. There exists one red node r chosen by the strategy of Black such that $r \in \mathit{Inf}(\pi)$. This contradicts the minimality of the r_i.

3.2 Testing game

We now examine the general case. Recall that the winning plays are the elements of $s_0 V^* \bot V^\omega$, and $s_0 V^* w V^* s_0 V^\omega$ (for w a branch of the test purpose). The strategy is constructed by combining simpler strategies like the one defined in the previous section. As we will use two of them, we will add an argument to the sets $\mathcal{W}_i^j$ (the goal to reach). We split the game in three parts corresponding to the test preamble, the test body and the test postamble of a test case execution.

1. **test preamble:** starting from the initial state of the game, the goal is to reach the initial state tp_0 of the test purpose.
2. **test body:** once tp_0 has been reached, the strategy of Black is to stay in the test purpose or if Red chooses to leave the test purpose, the strategy will be to return to tp_0.
3. **test postamble:** once one leaf of the test purpose has been reached, the strategy for Black is to come back to the initial state s_0.

More precisely, the strategy is defined by a directed graph $\mathsf{G} = (\mathsf{Q}, \mathsf{E})$, where:

- $\mathsf{Q} = V \times \{\mathit{preamble}, \mathit{test\ body}, \mathit{postamble}, \mathit{end}\}$.
- E is defined by the following edges.
  - For black nodes $q \in V_B$, $q' \in V$ and $(q, q') \in E$:
    - $(q, \mathit{preamble}) \to (q', \mathit{preamble})$ if $q \in \mathcal{W}_i^j(tp_0)$, $q' \in \mathcal{W}_{i-1}^j(tp_0)$, and $q' \neq tp_0$
      $(q, \mathit{preamble}) \to (tp_0, \mathit{test\ body})$ if $(q, tp_0) \in E$
    - $(q, \mathit{test\ body}) \to (q', \mathit{test\ body})$ if (q, q') is an edge of the test purpose TP
      $(q, \mathit{test\ body}) \to (q', \mathit{postamble})$ if q' is a leaf of TP
    - $(q, \mathit{postamble}) \to (q', \mathit{postamble})$ if $q \in \mathcal{W}_i^j(s_0)$, $q' \in \mathcal{W}_{i-1}^j(s_0)$
      $$(q, \mathit{postamble}) \to \begin{cases} (s_0, \mathit{preamble}) \\ (s_0, \mathit{end}) \end{cases} \quad \text{if } (q, s_0) \in E$$
  - For red nodes $q \in V_R$ and $(q, q') \in E$:

* $(q, preamble) \to (q', preamble)$
  $(q, preamble) \to (tp_0, test\ body)$ if $(q, tp_0) \in E$
* $(q, test\ body) \to (q', test\ body)$
  $(q, test\ body) \to (q', postamble)$ if q' is a leaf of the test purpose.
  $(q, test\ body) \to (q', preamble)$ if $\exists q''.(q, q'') \in TP$
* $(q, postamble) \to (q', postamble)$
  $(q, postamble) \to \begin{cases} (s_0, preamble) \\ (s_0, end) \end{cases}$ if $(q, s_0) \in E$
* $\left.\begin{array}{l} (q, preamble) \\ (q, test\ body) \\ (q, postamble) \end{array}\right\} \to (\perp, end)$

This strategy defines the tester for the associated test purpose. We complete it into an IOSM by adding labels to the edges corresponding to the ones of the specification (edges leading to $\perp$ correspond to all the outputs, which are not yet used at one vertice). In section 5, we will study correctness notions w.r.t. implementation relations for this tester.

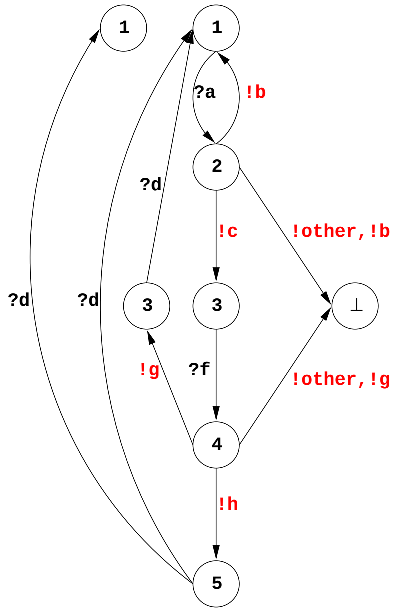
Example 4 Figure 5 shows a tester for the specification of Example 1, the test purpose $?a!c?f!h$ (this sequence of actions defines the branch of the test purpose) and the implementation relation R_5. Note the nondeterminism (in the automata sense this time) in states 2, 4, and 5. This is for convenience: in order to emit a verdict and because of the nondeterminism, we will have to carry out the same test several[9] times. After having executed a sufficient number of times this test, we can stop the experiment and emit a verdict. So, the nondeterministic choice in state 5 is resolved by first returning to node 1 several times and then stopping on the second node 1. The nondeterministic choice in 2 (resp. 4) is resolved by first going back to 1 (resp. 3) and then to stop on $\perp$ with a fail verdict. This nondeterminism can be suppressed by unfolding the graph for example (the other solution would be to add counters).

4 Complexity issues

In this section, we will examine some complexity aspects in terms of the size of the strategy and in terms of the running time of the test execution (duration of a play) with the number of states as the parameter.

[9] "Several" will be discussed in section 5.

Figure 5 a tester



Proposition 2 *The size of the tester, defined by the winning strategy* $\mathsf{G} = (\mathsf{Q}, \mathsf{E})$ *for the testing game, is linear in the size of the game and it takes a linear time to construct it.*

Proof By construction:

1. The strategy is made of reachability strategies defined in Section 3.1. Each reachability strategy being a subgraph of the initial game graph, the size of the final strategy is also linear.

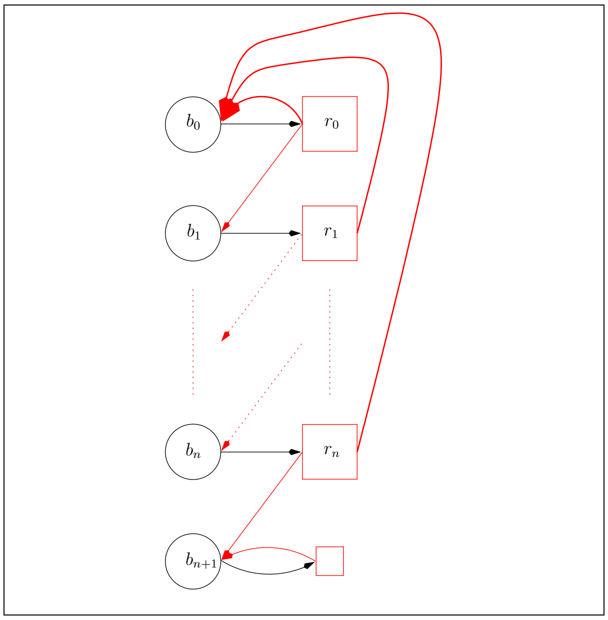
2. Each reachability strategy is made in at most n steps: as the constructed sets of Section 3.1 form a partition of the set of states, there are less steps than states. The time for constructing the strategy is also linear.

This matches the complexity of [7]. Note that this must not be seen as a good complexity for practical applications as the “state space explosion” problem has been abstracted (we claim however, that this method is compatible with orthogonal methods to reduce the state space explosion).

Proposition 3 *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a function from naturals to naturals. There exist a reachability game and a fair strategy for Red such that the length of a play is more than* $\prod_{i=1}^{n} f(i)$ *where* n *is the size of the game.*

Proof The game is a reachability game defined by: $V_B = b_0, \ldots b_{n+1}$, $V_R = r_1, \ldots, r_n$, $E = \{b_i, r_i \mid i \in [0,n]\} \cup \{(r_i, b_{i+1}) \mid i \in [0,n]\} \cup \{(r_i, b_0) \mid i \in [0,n]\}$. (see Figure 6) The winning plays are in $b_0 V^* b_{n+1} V^\omega$ (the length is taken until the token arrived in b_{n+1}. The strategy for Red in r_i is to play successively $f(i)$ times b_0 then b_{i+1}.

Figure 6 game of the proof



This example shows that the length of the test execution although being finite can be arbitrary long. The duration comes from the number l as defined in the sets defining the strategy for the reachability game in Section 3.1. The number is a measure of the degree of nondeterminism of the specification with respect to a given test purpose. For $l = 1$ the running time is linear. This complexity coming from the nondeterminism of a specification is often implicit in other test generation methods like [7] where testers are trees instead of automata: one has to apply several time the same test in order to get a non-inconclusive verdict. We showed here that this can be an arbitrary big number of times without any further hypotheses on the implementation. In the next section, we will study some hypotheses, which reduce the duration of a play.

5 Properties of generated testers

Having reasoned only in terms of games in the previous sections, we now switch back to testing for studying properties of the generated testers. For this, we recall some material from the FMCT concerning the execution of test.

Here, test execution consists of running the tester in combination with the implementation. During this run *observations* can be made and *verdicts* can be emitted. Test execution is modelled by parallel execution of the tester and the implementation. The observations that can be made are the traces of this product. Observations of interest are:

- the state $\perp$ of the tester has been reached. We can immediatly conclude with a **fail** verdict expressing non-conformance;
- starting from s_0, we went through the test purpose and went back to s_0. We are tempted to emit a **pass** verdict meaning that the implementation passes the test purpose. Nevertheless, as we are dealing with nondeterminism, this execution could also have led to emit a **fail** verdict. So we delay the emission of a **pass** verdict until we are sufficiently confident that no **fail** verdict could have been emitted (this is the bounded fairness hypothesis);
- being in the test purpose, we observe a trace, which is allowed by the specification, but which is not part of the test purpose. Often, at that stage, an *inconclusive* verdict is emitted. Here we continue the test execution trying to reach again the point to observe the desired action.

For the proofs, we will use notation from process algebra. To each IOSM, we associate a term of a process algebra written in lowercase letters (we refer the reader to [32] for full notation). $\|$ denotes parallel execution: $a \in \{?,!\} \times L, t_1 \xrightarrow{a} t_1', t_2 \xrightarrow{a} t_2' \vdash t_1 \| t_2 \xrightarrow{a} t_1' \| t_2'$.
Because of the hypotheses of input-enableness (an implementation always accept inputs from environment, a tester always accepts outputs from implementation), and deadlock observation (deadlocks of the implementation are supposed to be observable as a normal output), the parallel execution of the tester and the implementation model deadlocks only on $\perp$ or s_0.

5.1 Soundness

Soundness, as defined in FMCT, means that if an implementation is rejected by means of testing then it is not conform to the specification. An implementation is reject by the tester, if it reaches the state $\perp$.

Proposition 4 (soundness) *Let S, I, TP, T_{TP} be respectively a specification, an implementation model, a test purpose and the tester associated with the specification S and the test purpose TP; then T_{TP} is sound, ie if the parallel product of T_{TP} and I deadlocks on $\perp$, then I is not conform to S:*

$$\exists \sigma \in V^+ : (t_{TP} \| i \xrightarrow{\sigma} \perp \| i') \Rightarrow (I, S) \notin R_5.$$

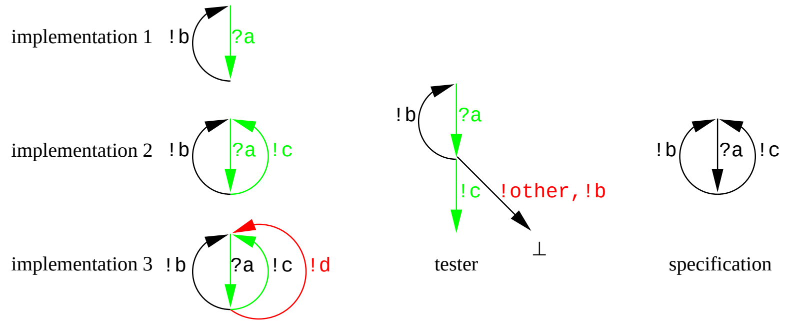
PROOF Suppose $t_{TP} \| i \xrightarrow{\sigma} \perp \| i'$. Let σ' be the minimal (with respect to prefix ordering) word such that $t_{TP} \| i \xrightarrow{\sigma} \perp \| i'$, and $a \in V$ be such that $\sigma' = \sigma''.!a$. By construction of the game and the minimality of σ', σ'' belongs to $Tr(S)$, and by definition of $\|$ it belongs also to $Tr(I)$ ($\sigma''.!a$, as well). By definition of $\perp$ in the game, $\sigma''.!a$ does not belong to $Tr(S)$. We have proved that $\mathcal{O}(\sigma'', I) \nsubseteq \mathcal{O}(\sigma'', S)$.

5.2 Fairness hypotheses

A problem due to the black-box nature of testing and nondeterminism of implementations is to know whether or not we observed all the possible outputs of an implementation after a given set of actions.

Take the example of Figure 7. The tester wants to observe the trace

Figure 7 Fairness hypotheses



$?a!c$. If it observes $(?a!b)^*$ when running in parallel with one of the implementations, it could be the case that it is the first implementation or the second one which only delayed the emission of $!c$. Having observed the trace $(?a(!b|!c))^*$, it is possible that the implementation 3 delayed the emission of $!d$ or that it is the implementation 2. Two notions of fairness hypotheses for the implementations will be used in the proof of the next section:

- fairness: as in the fair plays we used for games, it states that if the second implementation is running in parallel with the tester then $!c$ will eventually be observed. Fairness discriminates between implementation 1 and 2.

- bounded fairness: there exists an a priori known n depending on the length of the trace to be observed such that having observed $(?a(!b|!c))^n$, we will not observe $!d$ afterwards. Bounded fairness discriminates between implementation 2 and 3.

The definition follows from these examples ($\lhd$ is the sub-word relation):

Definition 7 (fairness hypotheses) The following hypotheses can be made about the implementation models:

- Fairness: we make a fairness hypothesis on an implementation model I, if we suppose that being repetitively able to send an output to the environment, it will eventually send it.

$$\begin{aligned}&\forall \sigma \in Tr(I) \cap Tr(S) : \forall a \in \mathcal{O}(\sigma, I) : \exists n \in \mathbb{N}\\ &\qquad (\exists t : \exists \omega = \omega_1, \ldots , \omega_n : t\|i \xrightarrow{\omega} t'\|i' \wedge \omega = \omega_1 \ldots \omega_n \wedge\\ &\qquad\qquad \bigwedge_i \sigma \lhd \omega_i) \Rightarrow \exists i : \sigma!a \lhd \omega_j.\end{aligned}$$

- Bounded fairness: We make a bounded fairness hypothesis on I if we suppose that all the outputs it can send at one state can be observed in a fixed time (or equivalently that it can be detected that I cannot send an output).

$$\begin{aligned}&\forall \sigma \in Tr(I) \cap Tr(S) : \exists n \in \mathbb{N} : \forall a \in \mathcal{O}(\sigma, S) :\\ &\qquad \big(\exists t : \exists \omega = \omega_1, \ldots , \omega_n : (t\|i \xrightarrow{\omega} t'\|i' \wedge\\ &\qquad\qquad \bigwedge_i (\sigma \lhd \omega_i \wedge \sigma!a \not\lhd \omega_i)\big) \Rightarrow \sigma!a \notin Tr(I).\end{aligned}$$

5.3 Exhaustivity

Exhaustivity, as defined in FMCT, means that if an implementation passes a test suite, it is conform to the specification. This is never achieved in practice because this would generally mean executing an infinite number of test cases (a non-bounded number of times as seen before). The property we prove here is less general. We prove that it is always possible to detect a non-conforming implementation provided one properly chooses the test purpose. In practice, this gives the limit of the automation of test generation: an algorithm based on the selection of testers by test purposes relies on the correct choice of these test purposes. The choice of the test purposes depends on a human test expert.

Proposition 5 (exhaustivity) *Let S and I be respectively a specification and an implementation model. If I satisfies a bounded fairness hypothesis, then*

$$(I, S) \notin R_5 \Rightarrow \exists TP \exists \sigma \in V^+ : t_{TP} \| i \xrightarrow{\sigma} \perp \| i'.$$

PROOF Suppose $(I, S) \notin R_5$. Two cases are possible.

1. $\mathcal{O}(\sigma, I) \not\subseteq \mathcal{O}(\sigma, S)$. In this case, there exists an a in L such that, for some σ, $\sigma.!a$ belongs to the traces of I but not the traces of of S. Let T be the tester associated with S and the test purpose σ.
   Let r be such that $t \xrightarrow{\sigma} r$ by construction of the tester. There exists σ_0 such that $t\|i \xrightarrow{\sigma_0} r\|i_0$ and σ_1 such that $t\|i \xrightarrow{\sigma_0\sigma_1} t\|i_1$. By applying the same construction again we construct a sequence which passes through r as often as needed. By fairness hypothesis, as the implementation can make the transition $!a$, there exists n such that $t\|i \xrightarrow{\sigma_0 \ldots \sigma_{2n}} r\|i_{2n} \xrightarrow{!a} \perp\|i_{2n+1}$.

2. $\mathcal{O}(\sigma, S) \not\subseteq \mathcal{O}(\sigma, I)$. In this case, there exists an a in L such that $\sigma!a$ belongs to the trace of S but not to the traces of of I. Let T the tester associated with S and the test purpose $\sigma!a$. Furthermore, we use the n of the bounded fairness hypothesis for the tester. At each choice point, where the implementation can choose to leave the test purpose, we allow only n tries, then we go to $\perp$. If after n tries, the parallel product did not deadlock on $\perp$, we stop on the next step through s_0 with a pass verdict.
   As the parallel product deadlocks only on $\perp$ or s_0, and will eventually always deadlock, it will deadlock on $\perp$.

6 Related work and conclusions

We have presented a new test generation algorithm for conformance testing. The test generating algorithm, which is the closest to ours is the one of [7, 8]. Although they come from different domains (the algorithm of [7] comes from the verification domain and on-the-fly algorithms, and ours comes from game theory), both are basically graph algorithms. Our test generation algorithm can be seen as an extension of the algorithm from [7] with respect to the correction of several aspects:

- Suppression of the inconclusive verdict: in [7], it is implicit that a test should be reexecuted when it leads to an inconclusive verdict. In our approach, reexecution of the test is explicitly coded into the tester. This, however, could lead to a tester which is not executable[10], so a (reasonable) fairness hypothesis on the implementation models ensures termination of the test execution (stronger hypotheses can shorten in practice the length of test execution). Reexecution of test cases also implicitly needs the existence of a well implemented reset. We took a less restrictive hypothesis, namely connectivity of the specification. This leads to shorter tests as going back to the initial state is not always the shortest path to reach again a given state. This is especially important when dealing with nondeterminism (in the testing sense) as the path we take is the one with the less nondeterministic choice points for the implementation, and, as we have seen, this nondeterminism can arbitrarily lengthen the duration of test execution.

- Emission of verdicts: in [7] two applications of the same tester may produce different verdicts. Under the fairness hypotheses, our testers cannot emit a pass and fail verdict for the same tester. This is also to be related with the definition of pass in [32]: a pass verdict is emitted if *all* the possible executions of a test lead to a pass verdict. This cannot be checked in practice. The explicit use of fairness hypotheses reduces these executions to a finite number.

- Stronger conformance relation: the conformance relation in [7] is based on inclusion, whereas ours is based on equality of allowed outputs. This is again due to the fairness hypotheses. It has been proved in [23][11] that it is the strongest implementation that can be checked by testing (intuitively, one can only observe traces of the black-box implementation, so the best we can do is to check trace equivalence).

These extensions rely on fairness hypotheses. One can argue whether these are "reasonable" hypotheses, but making them explicit offers the advantage of knowing under which assumptions an implementation viewed as a black-box can be declared practically "conform" to a specification. Any correctness method relies on hypotheses, even verification, which rely on the appropriateness between formal model and actual implementation. This hypothesis is often implicit and has led to misunderstandings [11][12]. In using explicitly

[10]Because the tester has infinite branches, it would be possible that it does not emit a verdict in finite time

[11]also [24] p.34 and p.42

[12]"Myth 1 - Formal Methods can guarantee that software is perfect."

test hypotheses, we follow the views of [2, 24].

Another aspect of our method is the use of game theory for testing, which to our knowledge has not been presented before for conformance testing (though [1] used game theory to study the complexity of the test related problem of producing distinguishing sequences for Mealy machines). The basic idea is to reformulate the game view of the synthesis problem of [21] for conformance testing. At a first glance, it could be objected that complexity of strategies construction make any approach based on game theory unrealistic for practical application. Nevertheless, as the obtained games are situated low in the Borel hierarchy [29] this leads to strategies which can be efficiently constructed. Furthermore the careful complexity analysis performed in [17, 20] reveals that particular cases can have reasonable complexity for practical application. This is the case here, because we do not need winning plays situated higher in the hierarchy..

It is our opinion that this game-theoretic view could be useful, because it establishes a link between verification and testing. It has already been advocated several times, from testing and verification community, that verification and testing should be brought closer. We tried to demonstrate here that game theory is a unifying theory for both domains by using it for conformance testing (see [27, 25] for the verification side). This could lead to a better understanding of testing problems as well as techniques for solving them.

References

[1] Rajeev Alur, Costas Courcoubetis, and Mihalis Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 363–372, 1995.

[2] Gilles Bernot. Testing against formal specifications: a theoritical view. In *TAPSOFT'91 Proceedings of the International Conference on Theory and Practice of Software Development. Advances in Distributed Computing (ADC) and Colloquium on Computing Paradigms for Software Development (CCPSD)*, volume 494 of *Lecture Notes in Computer Science*, pages 99–119, Brighton, UK, April 1991. Springer.

[3] Ana R. Cavalli, Jean-Philippe Favreau, and Marc Phalippou. Standardization of formal methods in conformance testing of communication protocols. *Computer Network and ISDN Systems*, 29:3–14, 1996.

[4] CCITT. Specification and description language (SDL). Recommandation Z.100, 1988.

[5] R. Devillers. Game interpretation of the deadlock avoidance problem. *Communications of the ACM*, 20(10):741–745, 1977.

[6] Jean-Claude Fernandez, Jard Claude, Thierry Jéron, Laurence Nedelka, and César Viho. Using on-the-fly verification techniques for the generation of test suites. Research Report RR-2987, INRIA, September 1996. Also, [7].

[7] Jean-Claude Fernandez, Jard Claude, Thierry Jéron, and César Viho. Using on-the-fly verification techniques for the generation of test suites. In *Conference on Computer-Aided Verification 8th International Conference (CAV)*, volume 1102 of *LNCS*, 1996. Also, [6].

[8] Jean-Claude Fernandez, Claude Jard, Thierry Jéron, Laurence Nedelka, and César Viho. An experiment in automatic generation of test suites for protocols with verification technology. Research Report RR-2923, INRIA, June 1996. Also, [9].

[9] Jean-Claude Fernandez, Claude Jard, Thierry Jéron, and César Viho. An experiment in

automatic generation of test suites for protocols with verification technology. *Science of Computer Programming. Methods of Software Design: Techniques and Applications*, 29(1–2):123–146, July 1997. Special Issue on COST247, Verification and Validation Methods for Formal Descriptions. Guest Editors: J.F. Groote and M. Rem. Also, [8].

[10] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. A method for the generation of test cases based on SDL and MSCs. Technical Report IAM-93-10, Institut für Informatik, Universität Bern, Switzerland, April 1993.

[11] J.A. Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.

[12] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[13] ISO. Information Processing Systems, Open Systems Interconnection, Estelle—A Formal Description Technique Based on an Extended State Transition Model. International Standard IS-9074, 1989.

[14] ISO. Information Processing Systems, Open Systems Interconnection, LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behavior. International Standard IS-8807, 1989.

[15] ISO. Information Technology, Open Systems Interconnection—Conformance Testing Methodology and Framework. International Standard IS-9646, 1991.

[16] ISO. Information Technology—Framework: Formal Methods in Conformance Testing, 1997. ISO/IEC DIS 13245-1 ISO international standard. Also Z500 ITU-T recommendation.

[17] Helmut Lescow. On polynomial-size programs winning finite-state games. In *7th International Conference on Computer Aided Verification (CAV)*, volume 939 of *Lecture Notes in Computer Science*, pages 239–252, July 1995.

[18] Nancy Lynch and Mark Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, 1989. Also, Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology.

[19] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.

[20] Amil Nerode, Jeffrey B. Remel, and Alexander Yakhnis. McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78:203–242, 1996.

[21] Anil Nerode, Alexander Yakhnis, and Vladimir Yakhnis. *Logic from computer science*, volume 21 of *Mathematical sciences research institute publications*, chapter Concurrent programs as strategies in games. Springer, 1992.

[22] Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theorical Computer Science*, 163(2):99–116, 1997.

[23] Marc Phalippou. The limited power of testing. In *Proceedings of the 5th International Workshop on Protocol Test Systems (IWPTS)*, Montréal, Canada, 1992.

[24] Marc Phalippou. *Relations d'implantation et hypothèses de test sur des automates à entrées sorties.* Thèse de doctorat (Ph.D.), Université de Bordeaux I, Septembre 1994. In french.

[25] Perdita Stevens and Colin Stirling. Practical model-checking using games. In *Proceedings of the 4th International Conference Tools and Algorithms for the Construction and Analysis of Systems, (TACAS)*, number 1384 in Lecture Notes in Computer Science, 1998. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'98.

[26] Colin Stirling. Local model checking games. In Insup Lee and Scott A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR)*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11, Philadelphia, PA, USA, August 1995.

[27] Colin Stirling. Games and modal mu-calculus. In *Proceedings of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 298–312, Passau, Germany, March 1996.

[28] Wolfgang Thomas. *Handbook of Theoritical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 133–191. Elsevier Science, 1990.

[29] Wolfgang Thomas. Finite-state strategies in regular infinite games. In P.S. Thiagarajan, editor, *14th Conf. on Foundations of Software Technology and Theorical Computer Science (FST & TCS)*, volume 880 of *Lecture Notes in Computer Science*, pages 149–158, 1994.

[30] Jan Tretmans. *A formal Approach to Conformance Testing*. Proefschrift (Ph.D.), Universiteit Twente, December 1992.

[31] Jan Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Network and ISDN Systems*, 29:49–79, 1996.

[32] Jan Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – concepts and tools*, 17:103–120, 1996.

[33] Moshe Y. Vardi. Verification of open systems. In *Foundations of Software Technology and Theorical Computer Science, 17th Conference*, volume 1346 of *Lecture Notes in Computer Science*, 1997. Invited paper.

[34] Alexander Yakhnis and Vladimir Yakhnis. Gurevich-Harrington's games defined by finite automata. *Annals of Pure and Applied Logic*, 62:265–294, 1993.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid *E*-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values
MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda\rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-004	W. Charatonik, A. Podelski	Solving set constraints for greatest models
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics