Activity Report 2012

Project-Team VERTECS

Verification models and techniques applied to testing and control of reactive systems

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real Time Systems

Table of contents

Project-Team VERTECS

Keywords: Formal Methods, Verification, Model-Checking, Program Testing, Control, Embedded Systems

Creation of the Project-Team: January 01, 2002 .

1. Members

Research Scientists

Thierry Jéron [Team Leader, Senior Researcher, INRIA, HdR]
Nathalie Bertrand [Junior Researcher, INRIA, on sabbatical at Liverpool University until August 2012]
Hervé Marchand [Junior Researcher, INRIA]

PhD Students

Sébastien Chédor [UNIVERSITÉ DE RENNES 1, since September 2009]
Paulin Fournier [ENS DE CACHAN, since September 2012]
Srinivas Pinisetty [INRIA, since December 2011]
Amélie Stainer [UNIVERSITÉ DE RENNES 1, since October 2010]

Visiting Scientists

Christophe Morvan [Assistant Professor, Univ. de Marne-la-Vallée]
Laurie Ricker [Associate Professor, Mount Allison University, January 2012 to June 2012]

Administrative Assistant

Lydie Mabil [TR Inria, (80%)]

2. Overall Objectives

2.1. Introduction

The VerTeCs team is focused on the use of formal methods to assess the reliability, safety and security of reactive software systems. By reactive software system we mean a system controlled by software which reacts with its environment (human or other reactive software). Among these, critical systems are of primary importance, as errors occurring during their execution may have dramatic economical or human consequences. Thus, it is essential to establish their correctness before they are deployed in a real environment, or at least detect incorrectness during execution and take appropriate action. For this aim, the VerTeCs team promotes the use of formal methods, i.e. formal specification of software and their required properties and mathematically founded validation methods. Our research covers several validation methods, all oriented towards a better reliability of software systems:

- Verification, which is used during the analysis and design phases, and whose aim is to establish the correctness of specifications with respect to requirements, properties or higher level specifications.
- Control synthesis, which consists in “forcing” (specifications of) systems to behave as desired by coupling them with a supervisor.
- Conformance testing, which is used to check the correctness of a real system with respect to its specification. In this context, we are interested in model-based testing, and in particular automatic test generation of test cases from specifications.
- Diagnosis and monitoring, which are used at run time to detect erroneous behaviour.
- Combinations of these techniques, both at the methodological level (combining several techniques within formal validation methodologies) and at the technical level (as the same set of formal verification techniques - model checking, theorem proving and abstract interpretation - are required for control synthesis, test generation and diagnosis).

Our research is thus concerned with the development of formal models for the description of software systems, the formalization of relations between software artifacts (e.g. satisfaction, conformance between properties, specifications, implementations), the interaction between these artifacts (modelling of execution, composition, etc). We develop methods and algorithms for verification, controller synthesis, test generation and diagnosis that ensure desirable properties (e.g. correctness, completeness, optimality, etc). We try to be as generic as possible in terms of models and techniques in order to cope with a wide range of application domains and specification languages. Our research has been applied to telecommunication systems, embedded systems, smart-cards application, and control-command systems. We implemented prototype tools for distribution in the academic world, or for transfer to the industry.

Our research is based on formal models and our basic tools are **verification** techniques such as model checking, abstract interpretation, the control theory of discrete event systems, and their underlying models and logics. The close connection between testing, control and verification produces a synergy between these research topics and allows us to share theories, models, algorithms and tools.

2.2. Highlights of the Year

The article [6] entitled Probabilistic omega-automata and co-authored by Nathalie Bertrand, together with Christel Baier and Marcus Grösser from TU Dresden, has been published in the Journal of the ACM. This article extends a paper published in 2008 in the proceedings of FoSSaCS, which received the EATCS best paper award, and already had a strong impact in the verification community.

3. Scientific Foundations

3.1. Underlying models

The formal models we use are mainly automata-like structures such as labelled transition systems (LTS) and some of their extensions: an LTS is a tuple $M = (Q, \Lambda, \to, q_o)$ where Q is a non-empty set of states; $q_o \in Q$ is the initial state; A is the alphabet of actions, $\to \subseteq Q \times \Lambda \times Q$ is the transition relation. These models are adapted for testing and controller synthesis.

To model reactive systems in the testing context, we use Input/Output labeled transition systems (IOLTS for short). In this setting, the interactions between the system and its environment (where the tester lies) must be partitioned into inputs (controlled by the environment), outputs (observed by the environment), and internal (non observable) events modeling the internal behavior of the system. The alphabet Λ is then partitioned into $\Lambda_! \cup \Lambda_? \cup \mathcal{T}$ where $\Lambda_!$ is the alphabet of outputs, $\Lambda_?$ the alphabet of inputs, and $\mathcal{T}$ the alphabet of internal actions.

In the controller synthesis theory, we also distinguish between controllable and uncontrollable events ($\Lambda = \Lambda_c \cup \Lambda_{uc}$), observable and unobservable events ($\Lambda = \Lambda_O \cup \mathcal{T}$).

In the context of verification, we also use Timed Automata. A timed automaton is a tuple $A = (L, X, E, \mathcal{I})$ where L is a set of locations, X is a set of clocks whose valuations are positive real numbers, $E \subseteq L \times \mathcal{G}(X) \times 2^X \times L$ is a finite set of edges composed of a source and a target state, a guard given by a finite conjunction of expressions of the form $x \sim c$ where x is a clock, c is a natural number and $\sim \in \{<, \leq, =, \geq, >\}$, a set of resetting clocks, and $\mathcal{I} : \mathcal{L} \to \mathcal{G}(\mathcal{X})$ assigns an invariant to each location [22]. The semantics of a timed automaton is given by a (infinite states) labelled transition system whose states are composed of a location and a valuation of clocks.

Also, for verification purposes, we use graph grammars that are a general tool to define families of graphs. Such grammars are formed by a set of rules whose left-hand sides are hyperedges and right-hand sides are hypergraphs. For graphs with finite degree, these grammars characterise transition graphs of pushdown automata (the correspondence between graphs generated by grammars and transition graphs of pushdown automata is bijective). Graph grammars provide a simple yet powerful setting to define and study infinite state systems.

In order to cope with models closer to practical specification languages, we also need higher level models encompassing both control and data aspects. We defined (input-output) symbolic transition systems ((IO)STS), which are extensions of (IO)LTS that convey or operate on data (i.e., program variables, communication parameters, symbolic constants) through message passing, guards, and assignments. Formally, an IOSTS is a tuple (V, Θ, Σ, T), where V is a set of variables (including a counter variable encoding the control structure), Θ is the initial condition defined by a predicate on V, Σ is the finite alphabet of actions, where each action has a signature (just like in IOLTS, Σ can be partitioned as e.g. $\Sigma_? \cup \Sigma_! \cup \Sigma_\tau$), T is a finite set of symbolic transitions of the form $t = (a, p, G, A)$ where a is an action (possibly with a polarity reflecting its input/output/internal nature), p is a tuple of communication parameters, G is a guard defined by a predicate on p and V, and A is an assignment of variables. The semantics of IOSTS is defined in terms of (IO)LTS where states are vectors of values of variables, and transitions between them are labelled with instantiated actions (action with valued communication parameter). This (IO)LTS semantics allows us to perform syntactical transformations at the (IO)STS level while ensuring semantical properties at the (IO)LTS level. We also consider extensions of these models with added features such as recursion, fifo channels, etc. An alternative to IOSTS for specifying systems with data variables is the model of synchronous dataflow equations.

Our research is based on well established theories: conformance testing, supervisory control, abstract interpretation, and theorem proving. Most of the algorithms that we employ take their origins in these theories:

- graph traversal algorithms (breadth first, depth first, strongly connected components, ...). We use these algorithms for verification as well as test generation and control synthesis.
- BDDs (Binary Decision Diagrams) algorithms, for manipulating Boolean formulae, and their MTBDDs (Multi-Terminal Decision Diagrams) extension for manipulating more general functions. We use these algorithms for verification, test generation and control.
- abstract interpretation algorithms, specifically in the abstract domain of convex polyhedra (for example, Chernikova's algorithm for the computation of dual forms). Such algorithms are used in verification and test generation.
- logical decision algorithms, such as satisfiability of formulas in Presburger arithmetics. We use these algorithms during generation and execution of symbolic test cases.

3.2. Verification

Verification in its full generality consists in checking that a system, which is specified by a formal model, satisfies a required property. Verification takes place in our research in two ways: on the one hand, a large part of our work, and in particular controller synthesis and conformance testing, relies on the ability to solve some verification problems. Many of these problems reduce to reachability and coreachability questions on a formal model (a state s is *reachable from an initial state* s_i if an execution starting from s_i can lead to s; s is *coreachable from a final state* s_f if an execution starting from s can lead to s_f). These are important cases of verification problems, as they correspond to the verification of safety properties.

On the other hand we investigate verification on its own in the context of complex systems. For expressivity purposes, it is necessary to be able to describe faithfully and to deal with complex systems. Some particular aspects require the use of infinite state models. For example asynchronous communications with unknown transfer delay (and thus arbitrary large number of messages in transit) are correctly modeled by unbounded FIFO queues, and real time systems require the use of continuous variables which evolve with time. Apart from these aspects requiring infinite state data structure, systems often include uncertain or random behaviours (such as failures, actions from the environment), which it make sense to model through probabilities. To encompass these aspects, we are interested in the verification of systems equipped with infinite data structures and/or probabilistic features.

When the state space of the system is infinite, or when we try to evaluate performances, standard model-checking techniques (essentially graph algorithms) are not sufficient. For large or infinite state spaces, symbolic model-checking or approximation techniques are used. Symbolic verification is based on efficient representations of sets of states and permits exact model-checking of some well-formed infinite-state systems.

However, for feasibility reasons, it is often mandatory to use approximate computations, either by computing a finite abstraction and resort to graph algorithms, or preferably by using more sophisticated abstract interpretation techniques. For systems with stochastic aspects, a quantitative analysis has to be performed, in order to evaluate the performances. Here again, either symbolic techniques (e.g. by grouping states with similar behaviour) or approximation techniques should be used.

We detail below verification topics we are interested in: abstract interpretation, quantitative model-checking and analysis of systems defined by graph grammars.

3.2.1. Abstract interpretation and data handling

Most problems in test generation or controller synthesis reduce to state reachability and state coreachability problems which can be solved by fixpoint computations of the form $x = F(x), x \in C$ where C is a lattice. In the case of reachability analysis, if we denote by S the state space of the considered program, C is the lattice $\wp(S)$ of sets of states, ordered by inclusion, and F is roughly the "*successor states*" function defined by the program.

The big change induced by taking into account the data and not only the (finite) control of the systems under study is that the fixpoints become uncomputable. The undecidability is overcome by resorting to approximations, using the theoretical framework of Abstract Interpretation [24]. The fundamental principles of Abstract Interpretation are:

1. to substitute to the *concrete domain* C a simpler *abstract domain* A (static approximation) and to transpose the fixpoint equation into the abstract domain, so that one has to solve an equation $y = G(y), y \in A$;
2. to use a *widening operator* (dynamic approximation) to make the iterative computation of the least fixpoint of G converge after a finite number of steps to some upper-approximation (more precisely, a post-fixpoint).

Approximations are conservative so that the obtained result is an upper-approximation of the exact result. In simple cases the state space that should be abstracted has a simple structure, but this may be more complicated when variables belong to different data types (Booleans, numerics, arrays) and when it is necessary to establish *relations* between the values of different types.

3.2.2. Model-checking quantitative systems

Model-checking techniques for finite-state systems are now quite developed, and a current challenge is to adapt them as much as possible to infinite-state systems. We detail below two types of models we are interested in: timed automata and infinite-state probabilistic systems.

Model-checking timed automata The model of timed automata, introduced by Alur and Dill in the 90's [22] is commonly used to represent real-time systems. Timed automata consist of an extension of finite automata with continuous variables, called clocks, that evolve synchronously with time, and can be tested and reset along an execution. Despite their uncountable state space, checking reachability, and more generally ω-regular properties, is decidable *via* the construction of a finite abstraction, the so-called region automaton. The recent developments in model-checking timed automata have aimed at modelling and verifying quantitative aspects encompassing timing constraints, for example costs, probabilities, frequencies. These quantitative questions demand advanced techniques that go far beyond the classical methods.

Model-checking infinite state probabilistic systems Model-checking techniques for finite state probabilistic systems are now quite developed. Given a finite state Markov chain, for example, one can check whether some property holds almost surely (i.e. the set of executions violating the property is negligible), and one can even compute (or at leat approximate as close as wanted) the probability that some property holds. In general, these techniques cannot be adapted to infinite state probabilistic systems, just as model-checking algorithms for finite state systems do not carry over to infinite state systems. For systems exhibiting complex data structures (such as unbounded queues, continuous clocks) and uncertainty modeled by probabilities, it can thus be hard to design model-checking algorithms. However, in some cases, especially when considering qualitative verification, symbolic methods can lead to exact results. Qualitative questions aim neither at computing nore at approximating a probability, but are only concerned with almost-sure or non neglectible behaviours (that is events of probability either one or non zero). In some cases, qualitative model-checking can be derived from a combination of techniques for infinite state systems (such as abstractions) with methods for finite state probabilistic systems. However, when one is interested in computing (or rather approximating) precise probability values (neither 0 nor 1), exact methods are scarce. To deal with these questions, we either try to restrict to classes of systems where exact computations can be made, or look for approximation algorithms.

3.2.3. Analysis of infinite state systems defined by graph grammars

Currently, many techniques (reachability, model checking, ...) from finite state systems have been generalised to pushdown systems, that can be modeled by graph grammars. Several such extensions heavily depend on the actual definition of the pushdown automata, for example, how many top stack symbols may be read, or whether the existence of ε-transitions (silent transitions) is allowed. Many of these restrictions do not affect the actual structure of the graph, and interesting properties like reachability or satisfiability (of a formula) only depend on the structure of a graph.

Deterministic graph grammars enable us to focus on structural properties of systems. The connection with finite graph algorithms is often straightforward: for example reachability is obtained by iterating the finite graph algorithm iterated on the right hand sides of the rules. On the other hand, extending these grammars with time or probabilities is not straightforward: qualitative values associated to different copies (in the graph) of the same vertex (in the grammar) may differ, introducing complex equations. Furthermore, the fact that the left-hand sides of rules are single hyperarcs is a strong restriction. But removing this restriction would lead to non-recursive graphs. Identifying decidable families of graphs defined by contextual graph grammars is also very challenging.

3.3. Automatic test generation

We are mainly interested in conformance testing, which consists in checking whether a black box implementation under test (the real system that is only known by its interface) behaves correctly with respect to its specification (the reference which specifies the intended behavior of the system). In the line of model-based testing, we use formal specifications and their underlying models to unambiguously define the intended behavior of the system, to formally define conformance and to design test case generation algorithms. The difficult problems are to generate test cases that correctly identify faults (the oracle problem) and, as exhaustiveness is impossible to reach in practice, to select an adequate subset of test cases that are likely to detect faults. Hereafter we detail some elements of the models, theories and algorithms we use.
We use IOLTS (or IOSTS) as formal models for specifications, implementations, test purposes, and test cases. We adapt a well established theory of conformance testing [30], which formally defines conformance as a relation between formal models of specifications and implementations. This conformance relation, called **ioco** compares the visible behaviors (called *suspension traces*) of the implementation I (denoted by $STraces(I)$) with those of the specification S ($STraces(S)$). Suspension traces are sequence of inputs, outputs or quiescence (absence of action denoted by δ), thus abstracting away internal behaviors that cannot be observed by testers. Intuitively, I *ioco* S if after a suspension trace of the specification, the implementation I can only show outputs and quiescences of the specification S. We re-formulated ioco as a partial inclusion of visible behaviors as follows:

$$I\ ioco\ S \Leftrightarrow STraces(I) \cap [STraces(S).\Lambda_!^\delta \setminus STraces(S)] = \varnothing.$$

In other words, suspension traces of I which are suspension traces of S prolongated by an output or quiescence, should still be suspension traces of S.

Interestingly, this characterization presents conformance with respect to S as a safety property of suspension traces of I. The negation of this property is charaterized by a *canonical tester* $Can(S)$ which recognizes exactly $[STraces(S).\Lambda_!^\delta \setminus STraces(S)]$, the set of non-conformant suspension traces. This *canonical tester* also serves as a basis for test selection.

Test cases are processes executed against implementations in order to detect non-conformance. They are also formalized by IOLTS (or IOSTS) with special states indicating *verdicts*. The execution of test cases against implementations is formalized by a parallel composition with synchronization on common actions. A *Fail* verdict means that the implementation under test (IUT) is rejected and should correspond to non-conformance, a *Pass* verdict means that the IUT exhibited a correct behavior and some specific targeted behaviour has been observed, while an *Inconclusive* verdict is given to a correct behavior that is not targeted.

Test suites (sets of test cases) are required to exhibit some properties relating the verdict they produce to the conformance relation. Soundness means that only non conformant implementations should be rejected by a test suite and exhaustiveness means that every non conformant implementation may be rejected by the test suite. Soundness is not difficult to obtain, but exhaustiveness is not possible in practice and one has to select test cases.
Test selection is often based on the coverage of some criteria (state coverage, transition coverage, etc). But test cases are often associated with *test purposes* describing some abstract behaviors targeted by a test case. In our framework, test purposes are specified as IOLTS (or IOSTS) associated with marked states or dedicated variables, giving them the status of automata or observers accepting runs (or sequences of actions or suspension traces). Selection of test cases amounts to selecting traces of the canonical tester accepted by the test purpose. The resulting test case is then both an observer of the negation of a safety property (non-conformance wrt. S), and an observer of a reachability property (acceptance by the test purpose). Selection can be reduced to a model-checking problem where one wants to identify states (and transitions between them) which are both reachable from the initial state and co-reachable from the accepting states. We have proved that these algorithms ensure soundness. Moreover the (infinite) set of all possibly generated test cases is also exhaustive. Apart from these theoretical results, our algorithms are designed to be as efficient as possible in order to be able to scale up to real applications.

Our first test generation algorithms are based on enumerative techniques, thus adapted to IOLTS models, and optimized to fight the state-space explosion problem. On-the-fly algorithms where designed and implemented in the TGV tool, which consist in computing co-reachable states from a target state during a lazy exploration of the set of reachable states in a product of the specification and the test purpose [25]. However, this enumerative technique suffers from some limitations when specification models contain data.

More recently, we have explored symbolic test generation techniques for IOSTS specifications [29]. The objective is to avoid the state space explosion problem induced by the enumeration of values of variables and communication parameters. The idea consists in computing a test case under the form of an *IOSTS*, i.e., a reactive program in which the operations on data are kept in a symbolic form. Test selection is still based on test purposes (also described as IOSTS) and involves syntactical transformations of IOSTS models that should ensure properties of their IOLTS semantics. However, most of the operations involved in test generation (determinisation, reachability, and coreachability) become undecidable. For determinisation we employ heuristics that allow us to solve the so-called bounded observable non-determinism (i.e., the result of an internal choice can be detected after finitely many observable actions). The product is defined syntactically. Finally test selection is performed as a syntactical transformation of transitions which is based on a semantical reachability and co-reachability analysis. As both problems are undecidable for IOSTS, syntactical transformations are guided by over-approximations using abstract interpretation techniques. Nevertheless, these over-approximations still ensure soundness of test cases [26]. These techniques are implemented in the STG tool (see 5.1), with an interface with NBAC used for abstract interpretation.

3.4. Control synthesis

The supervisory control problem is concerned with ensuring (not only checking) that a computer-operated system works correctly. More precisely, given a system model and a required property, the problem is to control the model's behavior, by coupling it to a supervisor, such that the controlled system satisfies the property [28]. The models used are LTSs and the associated languages, where one makes a distinction between *controllable* and *non-controllable* actions and between *observable* and *non-observable* actions. Typically, the controlled system is constrained by the supervisor, which can block on the system's controllable actions in order to force it to behave as specified by the property. The control synthesis problem can be seen as a constructive verification problem: building a supervisor that prevents the system from violating a property. Several kinds of properties can be enforced such as reachability, invariance (i.e. safety), attractivity, etc. Techniques adapted from model checking are used to compute the supervisor. Optimality must be taken into account as one often wants to obtain a supervisor that constrains the system as few as possible.

Supervisory control theory overview. Supervisory control theory deals with control of Discrete Event Systems. In this theory, the behavior of the system S is assumed not to be fully satisfactory. Hence, it has to be reduced by means of a feedback control (named Supervisor or Controller) in order to achieve a given set of requirements [28]. Namely, if S denotes the model of the system and Φ a safety property to be enforced on S, the problem consists of computing a supervisor $\mathcal{C}$ such that

$$S \| \mathcal{C} \models \Phi \tag{1}$$

where $\|$ is the classical parallel composition of LTSs. Given S, some events of S are said to be uncontrollable (Σ_{uc}), i.e., the occurrence of these events cannot be prevented by a supervisor, while the others are controllable (Σ_c). It means that all the supervisors satisfying (1) are not good candidates. The behavior of the controlled system must respect an additional condition that happens to be similar to the $ioco$ conformance relation previously defined in 3.3. This condition is called the *controllability condition* and it may be stated as

$$\mathcal{L}(S\|\mathcal{C})\Sigma_{uc} \cap \mathcal{L}(S) \subseteq \mathcal{L}(S\|\mathcal{C}) \tag{2}$$

Namely, when acting on S, a supervisor is not allowed to disable uncontrollable events. Given a safety property Φ, that can be modeled by an LTS A_Φ, there actually exist many different supervisors satisyfing both (1) and (2). Among all the valid supervisors, we are interested in computing the supremal one, ie the one that restricts the system as few as possible. It has been shown in [28] that such a supervisor always exists and is unique. It gives access to a behavior of the controlled system that is called the supremal controllable sub-language of A_Φ w.r.t. S and Σ_{uc}. In some situations, it may also be interesting to force the controlled system to be non-blocking (See [28] for details).

The underlying techniques are similar to the ones used for Automatic Test Generation. They consist of computing the product of the system model and A_Φ and to remove the states of the product that may lead to subsequent states violating the property by triggering only uncontrollable events.

4. Application Domains

4.1. Overview

The methods and tools developed by the VERTECS project-team for test generation and control synthesis of reactive systems are intended to be as generic as possible. This allows us to apply them in many application domains where the presence of software is predominant and its correctness is essential. In particular, we apply our research in the context of telecommunication systems, for embedded systems, for smart-cards application, and control-command systems.

4.2. Telecommunication systems

Our research on test generation was initially proposed for conformance testing of telecommunication protocols. In this domain, testing is a normalized process [21], and formal specification languages are widely used (SDL in particular). Our test generation techniques have already proved useful in this context, going up to industrial transfer. New standardized component-based design methodologies such as UML and OMG's MDE increase the need for formal techniques in order to ensure the compositionality of components, by verification and testing. Our techniques, by their genericity and adaptativity, have also proved useful at different levels of these methodologies, from component testing to system testing. The telecommunication industry now also tries to provide more and more services to the users. These services must be validated.

4.3. Software embedded systems

In the context of transport, software embedded systems are increasingly predominant. This is particularly important in automotive systems, where software replaces electronics for power train, chassis (e.g. engine control, steering, brakes) and cabin (e.g. wiper, windows, air conditioning) or new services to passengers are increasing (e.g. telematics, entertainment). Car manufacturers have to integrate software components provided by many different suppliers, according to specifications. One of the problems is that testing is done late in the life cycle, when the complete system is available. Faced with these problems, but also with the complexity of systems, compositionality of components, distribution, etc, car manufacturers now try to promote standardized interfaces and component-based design methodologies. They also develop virtual platforms which allow for testing components before the system is complete. It is clear that software quality and trust are one of the problems that have to be tackled in this context. This is why we believe that our techniques (testing and control) can be useful.

4.4. Control-command systems

The main application domain for our techniques is control-command systems. In general, such systems control costly machines (see, e.g., robotic systems, flexible manufacturing systems), that are connected to an environment (e.g., a human operator). Such systems are often critical systems and errors occurring during their execution may have dramatic economical or human consequences. In this field, the controller synthesis methodology (CSM) is useful to ensure by construction the interaction between 1) the different components, and 2) the environment and the system itself. For the first point, the CSM is often used as a safe scheduler, whereas for the second one, the supervisor can be interpreted as a safe discrete tele-operation system. Also in the context of the Vacsim ANR project, we investigate the testing, monitoring and verification of control-command systems.

5. Software

5.1. STG

Participant: Thierry Jéron.

STG (Symbolic Test Generation) is a prototype tool for the generation and execution of test cases using symbolic techniques. It takes as input a specification and a test purpose described as IOSTS, and generates a test case program also in the form of IOSTS. Test generation in STG is based on a syntactic product of the specification and test purpose IOSTS, an extraction of the subgraph corresponding to the test purpose, elimination of internal actions, determinisation, and simplification. The simplification phase now relies on NBAC, which approximates reachable and coreachable states using abstract interpretation. It is used to eliminate unreachable states, and to strengthen the guards of system inputs in order to eliminate some *Inconclusive* verdicts. After a translation into C++ or Java, test cases can be executed on an implementation in the corresponding language. Constraints on system input parameters are solved on-the-fly (i.e. during execution) using a constraint solver. The first version of STG was developed in C++, using Omega as constraint solver during execution. This version has been deposited at APP under number IDDN.FR.001.510006.000.S.P.2004.000.10600.

A new version in OCaml has been developed in the last years. This version is more generic and will serve as a library for symbolic operations on IOSTS. Most functionalities of the C++ version have been re-implemented. Also a new translation of abstract test cases into Java executable tests has been developed, in which the constraint solver is LUCKYDRAW (VERIMAG). This version has also been deposited at APP and is available for download on the web as well as its documentation and some examples.

Finally, in collaboration with ULB, we implemented a prototype SMACS, derived from STG, devoted to the control of infinite systems modeled by STS.

5.2. SIGALI

Participant: Hervé Marchand.

SIGALI is a model-checking tool that operates on ILTS (Implicit Labeled Transition Systems, an equational representation of an automaton), an intermediate model for discrete event systems. It offers functionalities for verification of reactive systems and discrete controller synthesis. It is developed jointly by the ESPRESSO and VERTECS teams. The techniques used consist in manipulating the system of equations instead of the set of solutions, which avoids the enumeration of the state space. Each set of states is uniquely characterized by a predicate and the operations on sets can be equivalently performed on the associated predicates. Therefore, a wide spectrum of properties, such as liveness, invariance, reachability and attractivity, can be checked. Algorithms for the computation of predicates on states are also available [27] [23]. SIGALI is connected with the Polychrony environment (ESPRESSO project-team) as well as the Matou environment (VERIMAG), thus allowing the modeling of reactive systems by means of Signal Specification or Mode Automata and the visualization of the synthesized controller by an interactive simulation of the controlled system. SIGALI is registered at APP.

Sigali is also integrated as part of the compiler of the language BZR.

6. New Results

6.1. Verification

6.1.1. Probabilistic ω-automata

Participant: Nathalie Bertrand.

Probabilistic ω-automata are a variant version of nondeterministic automata over infinite words where all choices are resolved by probabilistic distributions. Acceptance of a run for an infinite input word can be defined using traditional acceptance criteria for ω-automata, such as Büchi, Rabin or Streett conditions. The accepted language of a probabilistic ω-automata is then defined by imposing a constraint on the probability measure of the accepting runs. Together with Christel Baier and Marcus Grösser from TU Dresden, we studied a series of fundamental properties of probabilistic ω-automata with three different language-semantics: (1) the probable semantics that requires positive acceptance probability, (2) the almost-sure semantics that requires acceptance with probability 1, and (3) the threshold semantics that relies on an additional parameter λ in]0,1[that specifies a lower probability bound for the acceptance probability. We provided a comparison of probabilistic ω-automata under these three semantics and nondeterministic ω-automata concerning expressiveness and efficiency. Furthermore, we addressed closure properties under the Boolean operators union, intersection and complementation and algorithmic aspects, such as checking emptiness or language containment. This work was published in Journal of the ACM [6].

6.1.2. Petri nets reachability graphs

Participant: Christophe Morvan.

In the article [10], we investigate the decidability and complexity status of model-checking problems on unlabelled reachability graphs of Petri nets by considering first-order and modal languages without labels on transitions or atomic propositions on markings. We consider several parameters to separate decidable problems from undecidable ones. Not only are we able to provide precise borders and a systematic analysis, but we also demonstrate the robustness of our proof techniques.

6.1.3. Frequencies in timed automata

Participant: Amélie Stainer.

A quantitative semantics for infinite timed words in timed automata based on the frequency of a run was introduced earlier by Bertrand, Bouyer, Brihaye and Stainer. Unfortunately, most of the results are obtained only for one-clock timed automata because the techniques do not allow to deal with some phenomenon of convergence between clocks. On the other hand, the notion of forgetful cycle was introduced by Basset and Asarin, in the context of entropy of timed languages, and seems to detect exactly these convergences. In [20], we investigate how the notion of forgetfulness can help to extend the computation of the set of frequencies to n-clock timed automata.

6.1.4. Bounded satisfiability for PCTL

Participant: Nathalie Bertrand.

While model checking PCTL for Markov chains is decidable in polynomial-time, the decidability of PCTL satisfiability, as well as its finite model property, are long standing open problems. While general satisfiability is an intriguing challenge from a purely theoretical point of view, we argue that general solutions would not be of interest to practitioners: such solutions could be too big to be implementable or even infinite. Inspired by bounded synthesis techniques, we turn to the more applied problem of seeking models of a bounded size: we restrict our search to implementable – and therefore reasonably simple – models. In [14] and together with John Fearnley and Sven Schewe from University of Liverpool, we propose a procedure to decide whether or not a given PCTL formula has an implementable model by reducing it to an SMT problem. We have implemented our techniques and found that they can be applied to the practical problem of sanity checking – a procedure that allows a system designer to check whether their formula has an unexpectedly small model.

6.1.5. Graph transformation systems

Participant: Nathalie Bertrand.

In [13], we study decidability issues for reachability problems in graph transformation systems, a powerful infinite-state model. For a fixed initial configuration, we consider reachability of an entirely specified configuration and of a configuration that satisfies a given pattern (coverability). The former is a fundamental problem for any computational model, the latter is strictly related to verification of safety properties in which the pattern specifies an infinite set of bad configurations. In this paper we reformulate results obtained, e.g., for context-free graph grammars and concurrency models, such as Petri nets, in the more general setting of graph transformation systems and study new results for classes of models obtained by adding constraints on the form of reduction rules.

6.2. Active and passive testing

6.2.1. More testable properties

Participants: Thierry Jéron, Hervé Marchand.

Testing remains a widely used validation technique for software systems. However, recent needs in software development (e.g., in terms of security concerns) may require to extend this technique to address a larger set of properties. In [11], we explore the set of testable properties within the Safety-Progress classification where testability means to establish by testing that a relation, between the tested system and the property under scrutiny, holds. We characterize testable properties w.r.t. several relations of interest. For each relation, we give a sufficient condition for a property to be testable. Then, we study and delineate a fine-grain characterization of testable properties: for each Safety-Progress class, we identify the subset of testable properties and their corresponding test oracle. Furthermore, we address automatic test generation for the proposed framework by providing a general synthesis technique that allows to obtain canonical testers for the testable properties in the Safety-Progress classification. Moreover, we show how the usual notion of quiescence can be taken into account in our general framework, and, how quiescence improves the testability results. Then, we list some existing testing approaches that could benefit from this work by addressing a wider set of properties. Finally, we propose Java-PT, a prototype Java toolbox that implements the results introduced in this article.

6.2.2. *Runtime enforcement of timed properties*

Participants: Thierry Jéron, Hervé Marchand, Srinivas Pinisetty.

Runtime enforcement is a powerful technique to ensure that a running system respects some desired properties. Using an enforcement monitor, an (untrusted) input execution (in the form of a sequence of events) is modified into an output sequence that complies to a property. Runtime enforcement has been extensively studied over the last decade in the context of untimed properties. The paper [19], introduces runtime enforcement of timed properties. We revisit the foundations of runtime enforcement when time between events matters. We show how runtime enforcers can be synthesized for any safety or co-safety timed property. Proposed runtime enforcers are time retardant: to produce an output sequence, additional delays are introduced between the events of the input sequence to correct it. Runtime enforcers have been prototyped and our simulation experiments validate their effectiveness.

6.2.3. *Test generation for tiles systems*

Participants: Sébastien Chédor, Thierry Jéron, Christophe Morvan.

In [17] we explore test generation for Recursive Tile Systems (RTS) in the framework of the classical ioco testing theory. The RTS model allows the description of reactive systems with recursion, and is very similar to other models like Pushdown Automata, Hyperedge Replacement Grammars or Recursive State Machines. We first present an off-line test generation algorithm for Weighted RTS, a determinizable sub-class of RTS, and second, an on-line test generation algorithm for the full RTS model. Both algorithms use test purposes to guide test selection through targeted behaviours.

6.2.4. *Partially observed recursive tiles systems*

Participants: Sébastien Chédor, Hervé Marchand, Christophe Morvan.

The analysis of discrete event systems under partial observation is an important topic, with major applications such as the detection of information flow and the diagnosis of faulty behaviors. In [18] we consider recursive tile systems, which are infinite systems generated by a finite collection of finite tiles, a simplified variant of deterministic graph grammars. Recursive tile systems are expressive enough to capture classical models of recursive systems, such as the pushdown systems and the recursive state machines. They are infinite-state in general and therefore standard powerset constructions for monitoring do not always apply. We exhibit computable conditions on recursive tile systems and present non-trivial constructions that yield effective computation of the monitors. We apply these results to the classic problems of opacity and diagnosability.

6.2.5. *Off-line test selection with test purposes for non-deterministic timed automata*

Participants: Nathalie Bertrand, Thierry Jéron, Amélie Stainer.

The LMCS article [7] proposes novel off-line test generation techniques from non-deterministic timed automata with inputs and outputs (TAIOs) in the formal framework of the tioco conformance theory. In this context, a first problem is the determinization of TAIOs, which is necessary to foresee next enabled actions after an observable trace, but is in general impossible because not all timed automata are determinizable. This problem is solved thanks to an approximate determinization using a game approach. The algorithm performs an io-abstraction which preserves the tioco conformance relation and thus guarantees the soundness of generated test cases. A second problem is the selection of test cases from a TAIO specification. The selection here relies on a precise description of timed behaviors to be tested which is carried out by expressive test purposes modeled by a generalization of TAIOs. Finally, an algorithm is described which generates test cases in the form of TAIOs equipped with verdicts, using a symbolic co-reachability analysis guided by the test purpose. Properties of test cases are then analyzed with respect to the precision of the approximate determinization: when determinization is exact, which is the case on known determinizable classes, in addition to soundness, properties characterizing the adequacy of test cases verdicts are also guaranteed.

6.2.6. *Monitor-based statistical model checking of timed systems*

Participant: Amélie Stainer.

In [16], we present a novel approach and implementation for analysing weighted timed automata (WTA) with respect to the weighted metric temporal logic ($\mathrm{WMTL}_{\leq}$). Based on a stochastic semantics of WTAs, we apply statistical model checking (SMC) to estimate and test probabilities of satisfaction with desired levels of confidence. Our approach consists in the generation of deterministic monitors for formulas in $\mathrm{WMTL}_{\leq}$, allowing for efficient SMC by run-time evaluation of a given formula. By necessity, the deterministic observers are in general approximate (over- or under-approximations), but are most often exact and experimentally tight. The technique is implemented in the new tool CASAAL. that we seamlessly connect to Uppaal-smc. in a tool chain. We demonstrate the applicability of our technique and the efficiency of our implementation through a number of case-studies.

6.3. Control synthesis

6.3.1. Synthesis of opaque systems

Participant: Hervé Marchand.

Opacity is a security property formalizing the absence of (secret) information leakage. We address the problem of synthesizing opaque systems. A secret predicate S over the runs of a system G is opaque to an external user having partial observability over G, if he can never infer from the observation of a run of G that the run belongs to S. We choose to control the observability of events by adding a device, called a mask, between the system G and the users. We first investigate the case of static partial observability where the set of events the user can observe is fixed once and for all by a static mask. In this context, we show that checking whether a system is opaque is PSPACE-complete, which implies that computing an optimal static mask ensuring opacity is also a PSPACE-complete problem. Next, we introduce dynamic partial observability where the set of events the user can observe changes over time and is determined by a dynamic mask. We show how to check that a system is opaque w.r.t. to a dynamic mask and also address the corresponding synthesis problem: given a system G and secret states S, compute the set of dynamic masks under which S is opaque. Our main result is that the set of such masks can be finitely represented and can be computed in EXPTIME and that this is a lower bound. We also address the problem of computing an optimal mask. This work was published in FMSD [9].

6.3.2. Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation

Participant: Hervé Marchand.

In the DEDS article [12], we propose algorithms for the synthesis of state-feedback controllers with partial observation of infinite state discrete event systems modelled by Symbolic Transition Systems. We provide models of safe memoryless controllers both for potentially deadlocking and for deadlock free controlled systems. The termination of the algorithms solving these problems is ensured using abstract interpretation techniques which provide an overapproximation of the transitions to disable. We then extend our algorithms to controllers with memory and to online controllers. We also propose improvements in the synthesis of controllers in the finite case which, to our knowledge, provide more permissive solutions than previously proposed in the literature. Our tool SMACS gives an empirical validation of our methods by showing their feasibility, usability and efficiency.

6.3.3. Playing optimally on timed automata with random delays

Participant: Nathalie Bertrand.

In [15], we marry continuous time Markov decision processes (CTMDPs) with stochastic timed automata into a model with joint expressive power. This extension is very natural, as the two original models already share exponentially distributed sojourn times in locations. It enriches CTMDPs with timing constraints, or symmetrically, stochastic timed automata with one conscious player. Our model maintains the existence of optimal control known for CTMDPs. This also holds for a richer model with two players, which extends continuous time Markov games. But we have to sacrifice the existence of simple schedulers: polyhedral regions are insufficient to obtain optimal control even in the single-player case.

7. Partnerships and Cooperations

7.1. National initiatives

7.1.1. ANR VACSIM: Validation of critical control-command systems by coupling simulation and formal analysis

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

The Vacsim project (2011-2014) is a 3-year project with EDF R&D, Dassault Systèmes, LURPA Cachan, I3S Nice and Labri Bordeaux. The project aims at developping both methodological and formal contributions for the simulation and validation of control-command systems. The role of the Vertecs team will be to contribute to the advance of validation techniques for timed systems, including quantitative analysis and its application to testing, monitoring of timed systems, and verification of communicating timed automata. The VACSIM project funds the PhD thesis of Srinivas Pinisetty.

7.1.2. ANR Ctrl-Green (Autonomic management of green data centers)

Participant: Hervé Marchand.

The project Ctrl-Green (2011-2014) is a 3-year project with UJF/LIG, INPT/IRIT, Inria, EOLAS, Scalagent. This project aims at developing techniques for the automatic optimal management of reconfigurable systems in the context of data centers using discrete controller synthesis methodology applied in the synchronous paradigm. The role of the Vertecs team will be to contribute to the development of new controller synthesis methodology for symbolic synchronous systems handling variables and to its application to the autonomic management of data centers.

7.2. European Initiatives

7.2.1. Artist design network of excellence

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

Program: FP7

Project acronym: Artist Design

Project title: Artist - European Network of Excellence on Embedded System Design

Duration: 01/08 - 03/12

Coordinator: VERIMAG

Abstract: The central objective for ArtistDesign is to build on existing structures and links forged in Artist2, to become a virtual Center of Excellence in Embedded Systems Design. This will be mainly achieved through tight integration between the central players of the European research community. Also, the consortium is smaller, and integrates several new partners. These teams have already established a long-term vision for embedded systems in Europe, which advances the emergence of Embedded Systems as a mature discipline.
The research effort aims at integrating topics, teams, and competencies, grouped into 4 Thematic Clusters: “Modelling and Validation”, “Software Synthesis, Code Generation, and Timing Analysis”, “Operating Systems and Networks”, “Platforms and MPSoC”. “Transversal Integration” covering both industrial applications and design issues aims for integration between clusters.
The Vertecs EPI is a partner of the “Validation” activity of the “Modeling and Validation” cluster. This year, the Vertecs EPI has contributed to quantitative verification of timed automata [20], test generation from nondeterministic timed automata [7], and control sysnthesis using abstract interpretation for infinite state systems [12].

7.2.2. Major European Organizations with which the Team has followed Collaborations

Université Libre Bruxelles (Belgium), Prof. Thierry Massart, Testing and control of symbolic transitions systems.

University of Kaiserslautern (Germany), Roland Meyer, Petri nets.

University of Dresden (Germany), Prof. Christel Baier, Probabilistic automata over infinite words.

University of Mons (Belgium), Prof. Thomas Brihaye, Stochastic timed automata.

7.3. International Research Visitors

7.3.1. Visits of International Scientists

Laurie Ricker, associate professor at the Mathematics & Computer Science department of Mount Allison University (Canada) has visited Vertecs for 6 months, from January 2012 to June 2012. We collaborate on control of discrete event systems for distributed and decentralized systems.

7.3.2. Visits to International Teams

Nathalie Bertrand spent 9 months at University of Liverpool, from November 1st 2011 to July 31st 2012. Her visit was supported by the Leverhulme Trust and the Sabbatical program of Inria, which also permitted Paulin Fournier to spend 5 months at University of Liverpool for his Master thesis.

8. Dissemination

8.1. Scientific Animation

Nathalie Bertrand was PC member of QAPL'12, QEST'12, Infinity'12 and SYNT'12. She is since september 2012 a Steering Committee member of QEST. She was invited to give a talk at the research seminar of the Quantitative Logics and Automata (QuantLA) Doctoral School in Dresden, in December 2012. She is also a member of the panel for the Gilles Kahn prize awarded each year to an excellent PhD thesis in computer science.

Thierry Jéron is Vice President of the Project Committee of Inria Rennes - Bretagne Atlantique. He is member of the IFIP Working Group 10.2 on Embedded Systems. He is a member of the Steering Committe of the School MOVEP (held in Marseille in December 2012), and PC member of ICFEM 2012, ICST 2012, ICTSS 2012, RV 2013, TAP 2012 and 2013. He gave a lecture at the Summer School TAROT 2012. He co-organizes the Dagstuhl Seminar 13021 on "Symbolic Methods in Testing" in January 2013.

Hervé Marchand is Associate Editor of the IEEE Transactions on Automatic Control since 2009 and member of the IFAC Technical Committees (TC 1.3 on Discrete Event and Hybrid Systems) since 2005. He is member of the steering committee of MSR (Modélisation de systèmes réactifs). He was PC member of WODES'12, ICINCO'12. He is organizer of MSR 2013 in Rennes.

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Nathalie Bertrand

Master : Techniques de vérification avancées, 10h, niveau M2, ISTIC, Université de Rennes 1.

Agrégation : Langages formels, 16h, niveau M2, ENS-Cachan-Antenne de Bretagne.

Sébastien Chédor

Licence : Programmation Java, 20h, niveau L2, ISTIC, Université de Rennes 1.
APF, 20h, niveau L1, ISTIC, Université de Rennes 1.

Initiation LaTeX, 6h, niveau L3, ENS-Cachan-Antenne de Bretagne.
Programmation, 20h, niveau L3, ENS-Cachan-Antenne de Bretagne.

Paulin Fournier

Licence : Scheme (TP), 20h, niveau L1, INSA-Rennes.

Christophe Morvan

Licence : Object Oriented programming with Java, 120h, niveau L2, Université de Marne-la-Vallée.
Systèmes d'exploitation (Cours), 36h, niveau L3, Université de Marne-la-Vallée.
Compilation (Cours-TD), 60h, niveau L3, Université de Marne-la-Vallée.

Amélie Stainer

Licence : Base de données (Cours-TD-TP), 28h, niveau L2, INSA-Rennes.

Agrégation : Modélisation (Cours), 11h, niveau M2, ENS-Cachan-Antenne de Bretagne.
Oraux blancs, 12h, niveau M2, ENS-Cachan-Antenne de Bretagne.
Préparation de leçons d'informatique, 10h, niveau M2, ENS-Cachan-Antenne de Bretagne.

8.2.2. Supervision

PhD in progress: **Sébastien Chédor**, Verification and test of systems modeled by regular graphs, started in September 2009, supervised by Christophe Morvan and Thierry Jéron.

PhD in progress: **Paulin Fournier**, Parameterized Verification of probabilistic systems, started in September 2012, supervised by Nathalie Bertrand and Thierry Jéron.

PhD in progress: **Srinivas Pinisetty**, Runtime validation of critical control-command systems, started in December 2011, supervised by Hervé Marchand and Thierry Jéron.

PhD in progress: **Amélie Stainer**, Quantitative verification of timed automata, started in October 2010, supervised by Nathalie Bertrand and Thierry Jéron.

8.2.3. Juries

Nathalie Bertrand was a member of the PhD defense jury of Youssouf Oualhadj (LaBRI, Bordeaux) in december 2012.

8.3. Popularization

Christophe Morvan has been involved in the organization of a public talk on Turing (celebrating the centenary of his birth) by Jean Lassègue. See the webpage.

9. Bibliography

Major publications by the team in recent years

[1] C. Baier, N. Bertrand, M. Groesser. *Probabilistic omega-automata*, in "Journal of the Association for Computing Machinery", 2012, vol. 59, n^{o} 1, p. 1:1-1:52 [*DOI :* 10.1145/2108242.2108243], http://hal.inria.fr/hal-00743907.

[2] N. Bertrand, T. Jéron, A. Stainer, M. Krichen. *Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata*, in "Logical Methods in Computer Science", October 2012, vol. 8, n^{o} 4:8, p. 1-33, http://hal.inria.fr/hal-00744074.

[3] C. Constant, T. Jéron, H. Marchand, V. Rusu. *Integrating formal verification and conformance testing for reactive systems*, in "IEEE Transactions on Software Engineering", August 2007, vol. 33, n^{o} 8, p. 558-574.

[4] J. DUBREIL, P. DARONDEAU, H. MARCHAND. *Supervisory Control for Opacity*, in "IEEE Transactions on Automatic Control", May 2010, vol. 55, n^{o} 5, p. 1089-1100 [*DOI :* 10.1109/TAC.2010.2042008], http://hal.inria.fr/inria-00483891.

[5] B. GAUDIN, H. MARCHAND. *An Efficient Modular Method for the Control of Concurrent Discrete Event Systems: A Language-Based Approach*, in "Discrete Event Dynamic System", 2007, vol. 17, n^{o} 2, p. 179-209.

Publications of the year

Articles in International Peer-Reviewed Journals

[6] C. BAIER, N. BERTRAND, M. GROESSER. *Probabilistic omega-automata*, in "Journal of the Association for Computing Machinery", 2012, vol. 59, n^{o} 1, 1:52 [*DOI :* 10.1145/2108242.2108243], http://hal.inria.fr/hal-00743907.

[7] N. BERTRAND, T. JÉRON, A. STAINER, M. KRICHEN. *Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata*, in "Logical Methods in Computer Science", October 2012, vol. 8, n^{o} 4:8, p. 1-33, http://hal.inria.fr/hal-00744074.

[8] N. BERTRAND, A. LEGAY, S. PINCHINAT, J.-B. RACLET. *Modal event-clock specifications for timed component-based design*, in "Science of Computer Programming", 2012, n^{o} 77, p. 1212-1234 [*DOI :* 10.1016/J.SCICO.2011.01.007], http://hal.inria.fr/hal-00752449.

[9] F. CASSEZ, J. DUBREIL, H. MARCHAND. *Synthesis of opaque systems with static and dynamic masks*, in "Formal Methods in System Design", 2012, vol. 40, n^{o} 1, p. 88-115 [*DOI :* 10.1007/S10703-012-0141-9], http://hal.inria.fr/hal-00662539.

[10] P. DARONDEAU, S. DEMRI, R. MEYER, C. MORVAN. *Petri Net Reachability Graphs: Decidability Status of First Order Properties*, in "Logical Methods in Computer Science", October 2012, vol. 8, n^{o} 4:9, p. 1-28 [*DOI :* 10.2168/LMCS-8(4:9)2012], http://hal.inria.fr/hal-00743935.

[11] Y. FALCONE, J.-C. FERNANDEZ, T. JÉRON, H. MARCHAND, L. MOUNIER. *More testable properties*, in "International Journal on Software Tools for Technology Transfer", 2012, vol. 14, n^{o} 4, p. 407-437 [*DOI :* 10.1007/S10009-011-0220-Z], http://hal.inria.fr/hal-00743981.

[12] G. KALYON, T. LE GALL, H. MARCHAND, T. MASSART. *Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation*, in "Discrete Event Dynamic Systems", 2012, vol. 22, n^{o} 2, p. 121-161 [*DOI :* 10.1007/S10626-011-0101-3], http://hal.inria.fr/inria-00586169.

International Conferences with Proceedings

[13] N. BERTRAND, G. DELZANNO, B. KÖNIG, A. SANGNIER, J. STÜCKRATH. *On the Decidability Status of Reachability and Coverability in Graph Transformation Systems*, in "RTA - 23rd International Conference on Rewriting Techniques and Applications - 2012", Nagoya, Japan, LIPIcs, 2012, vol. 15, p. 101-116 [*DOI :* 10.4230/LIPICS.RTA.2012.101], http://hal.inria.fr/hal-00752446.

[14] N. BERTRAND, J. FEARNLEY, S. SCHEWE. *Bounded Satisfiability for PCTL*, in "CSL - 21st EACSL Annual Conferences on Computer Science Logic - 2012", Fontainebleau, France, 2012, vol. 16, p. 92-106 [*DOI :* 10.4230/LIPICS.CSL.2012.92], http://hal.inria.fr/hal-00752445.

[15] N. BERTRAND, S. SCHEWE. *Playing Optimally on Timed Automata with Random Delays*, in "Formats - 10th International Conference on Formal Modeling and Analysis of Timed Systems - 2012", London, United Kingdom, Lecture Notes in Computer Science, September 2012, vol. 7595, p. 43-58 [*DOI :* 10.1007/978-3-642-33365-1_5], http://hal.inria.fr/hal-00752440.

[16] P. BULYCHEV, A. DAVID, K. G. LARSEN, A. LEGAY, G. LI, D. BOGSTEN POULSEN, A. STAINER. *Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic*, in "Logic for Programming, Artificial Intelligence, and Reasoning", Merida, Venezuela, A. V. NIKOLAJ BJØRNER (editor), Lecture Notes in Computer Science, Springer, 2012, vol. 7180, p. 168-182, http://hal.inria.fr/hal-00744100.

[17] S. CHÉDOR, T. JÉRON, C. MORVAN. *Test generation from recursive tiles systems*, in "TAP - 6th International Conference on Tests & Proofs - 2012", Prague, Czech Republic, A. D. BRUCKER, J. JULLIAND (editors), LNCS, Springer, May 2012, vol. 7305, p. 99-114, http://hal.inria.fr/hal-00743941.

[18] S. CHÉDOR, C. MORVAN, S. PINCHINAT, H. MARCHAND. *Analysis of partially observed recursive tile systems*, in "11th Int. Workshop on Discrete Event Systems", Guadalajara, Mexico, October 2012, p. 265-271, http://hal.inria.fr/hal-00743196.

[19] S. PINISETTY, Y. FALCONE, T. JÉRON, H. MARCHAND, A. ROLLET, O. L. NGUENA TIMO. *Runtime Enforcement of Timed Properties*, in "3rd International Conference on Runtime Verification", Istanbul, Turkey, October 2012, http://hal.inria.fr/hal-00743270.

[20] A. STAINER. *Frequencies in Forgetful Timed Automata*, in "Formal Modeling and Analysis of Timed Systems", London, United Kingdom, M. JURDZINSKI, D. NICKOVIC (editors), Lecture notes in computer science, Springer, September 2012, vol. 7595, p. 236-251, http://hal.inria.fr/hal-00744081.

References in notes

[21] *Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Part 1 : General Concept - Part 2 : Abstract Test Suite Specification - Part 3 : The Tree and Tabular Combined Notation (TTCN)*, 1992, n^{o} International Standard ISO/IEC 9646-1/2/3.

[22] R. ALUR, D. L. DILL. *A Theory of Timed Automata*, in "Theor. Comput. Sci.", 1994, vol. 126, n^{o} 2, p. 183-235.

[23] L. BESNARD, H. MARCHAND, É. RUTTEN. *The Sigali Tool Box Environment*, in "Workshop on Discrete Event Systems, WODES'06 (Tool Paper)", Ann-Arbor (MI, USA), July 2006, p. 465-466.

[24] P. COUSOT, R. COUSOT. *Abstract intrepretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in "Conference Record of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles (CA, USA)", January 1977, p. 238-252.

[25] C. JARD, T. JÉRON. *TGV: theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems*, in "Software Tools for Technology Transfer (STTT)", October 2004, vol. 6.

[26] B. JEANNET, T. JÉRON, V. RUSU, E. ZINOVIEVA. *Symbolic Test Selection based on Approximate Analysis*, in "11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05),

Volume 3440 of LNCS", Edinburgh (Scotland), April 2005, p. 349-364, http://www.irisa.fr/vertecs/Publis/Ps/tacas05.pdf.

[27] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment*, in "Discrete Event Dynamic System : Theory and Applications", Octobre 2000, vol. 10, n^{o} 4, p. 347-368, http://www.irisa.fr/vertecs/Publis/Ps/2000-J-DEDS.pdf.

[28] P. J. RAMADGE, W. M. WONHAM. *The Control of Discrete Event Systems*, in "Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems", 1989, vol. 77, n^{o} 1, p. 81-98.

[29] V. RUSU, L. DU BOUSQUET, T. JÉRON. *An approach to symbolic test generation*, in "International Conference on Integrating Formal Methods (IFM'00)", Lecture Notes in Computer Science, 2000, vol. 1945, p. 338-357.

[30] J. TRETMANS. *Test Generation with Inputs, Outputs and Repetitive Quiescence*, in "Software - Concepts and Tools", 1996, vol. 17, n^{o} 3, p. 103-120.