INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Distributed On-the-Fly Model Checking and Test Case Generation

Christophe Joubert — Radu Mateescu

N° 5880

Avril 2006

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5880--FR+ENG

INRIA
RHÔNE-ALPES

Distributed On-the-Fly Model Checking and Test Case Generation

Christophe Joubert* , Radu Mateescu†

Thème COM — Systèmes communicants
Projet VASY

Rapport de recherche n° 5880 — Avril 2006 — 24 pages

Abstract: The explicit-state analysis of concurrent systems must handle large state spaces, which correspond to realistic systems containing many parallel processes and complex data structures. In this report, we combine the on-the-fly approach (incremental construction of the state space) and the distributed approach (state space exploration using several machines connected by a network) in order to increase the computing power of analysis tools. To achieve this, we propose MB-DSOLVE, a new algorithm for distributed on-the-fly resolution of multiple block, alternation-free boolean equation systems (BESs). First, we apply MB-DSOLVE to perform distributed on-the-fly model checking of alternation-free modal μ-calculus, using the standard encoding of the problem as a BES resolution. The speedup and memory consumption obtained on large state spaces improve over previously published approaches based on game graphs. Next, we propose an encoding of the conformance test case generation problem as a BES resolution from which a diagnostic representing the complete test graph is built. By applying MB-DSOLVE, we obtain a distributed on-the-fly test case generator whose capabilities scale up smoothly w.r.t. well-established existing sequential tools.

Key-words: boolean equation system, distributed algorithm, labeled transition system, model checking, μ-calculus, test case generation, verification

A short version of this report is also available as "Distributed On-the-Fly Model Checking and Test Case Generation", *Proceedings of the 13th International SPIN Workshop on Model Checking of Software (Vienna, Austria)*, March 3 – April 1, 2006. The first author's current affiliation is: GISUM (Software Engineering Group of the Malaga University), E.T.S.I. Telecomunicación, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain.

* current e-mail address: joubert@lcc.uma.es
† Radu.Mateescu@inria.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Vérification par logique temporelle et génération de tests distribuées et à la volée

Résumé : L'analyse des systèmes concurrents basée sur l'énumeration explicite des états nécessite la manipulation d'espaces d'états de grande taille, qui correspondent à des systèmes réalistes contenant de nombreux processus parallèles et des structures de données complexes. Dans ce rapport, nous combinons l'approche à la volée (construction incrémentale de l'espace d'états) et l'approche distribuée (exploration de l'espace d'états en utilisant plusieurs machines connectées en réseau) afin d'augmenter les capacités de calcul des outils d'analyse. Pour atteindre cet objectif, nous proposons MB-DSOLVE, un nouvel algorithme pour la résolution distribuée et à la volée des systèmes d'équations booléennes (SEBs) sans alternance contenant plusieurs blocs d'équations. Nous appliquons d'abord MB-DSOLVE pour effectuer la vérification distribuée et à la volée de formules du μ-calcul modal sans alternance, en utilisant la traduction standard du problème en termes de résolution de SEBs. L'accélération et la consommation mémoire obtenues sur des espaces d'états de grande taille sont ameliorées par rapport à des approches existantes basées sur les graphes de jeux. Ensuite, nous proposons une formulation du problème de la génération de cas de tests de conformité comme la résolution d'un SEB avec génération d'un diagnostic, à partir duquel est construit le graphe de test complet. En appliquant MB-DSOLVE, nous obtenons un générateur de tests distribué et à la volée dont les capacités passent bien à l'échelle par rapport à des outils séquentiels bien établis.

Mots-clés : algorithme distribué, génération de tests, logique temporelle, model checking, μ-calcul, système d'équations booléennes, système de transitions étiquetées, vérification

1 Introduction

The explicit-state verification of concurrent finite-state systems is confronted in practice with the *state explosion* problem (prohibitive size of the underlying state spaces), which occurs for realistic systems containing many parallel processes and complex data structures. Various approaches have been proposed for combating state explosion: *on-the-fly* verification constructs the state space in a demand-driven way, thus allowing the detection of errors without a priori building the entire state space, and *distributed* verification uses the computing resources of several machines connected by a network, thus allowing to scale up the capabilities of verification tools by one or two orders of magnitude [BLW02, JM04]. Practical experience suggests that combining these two techniques leads potentially to better results than using them separately [BBS01, HLL04, LS99].

Given that verification tools are complex pieces of software, their design should promote modular architectures and intermediate representations, in order to reuse existing achievements as much as possible. *Boolean Equation Systems* (BESs) [Mad97] are a useful intermediate representation for various verification problems, such as model checking of modal μ-calculus [And94, Mad97], equivalence checking [AV95, Mat03], and partial order reduction [PLM03]. Numerous sequential algorithms for on-the-fly BES resolution were proposed [And94, Mad97, MS03], some of them being subject to generic implementations, such as the CÆSAR_SOLVE library [Mat03, Mat05a], which serves as computing engine for the model checker EVALUATOR [MS03, Mat03], the equivalence checker BISIMULATOR [Mat03, BDJM05], and the reductor TAU_CONFLUENCE [PLM03, Mat05a], developed within the CADP toolbox [GLM02]. Due to their modular architecture, distributed versions of these tools can be obtained in a straightforward manner by developing distributed BES resolution algorithms, such as DSOLVE [JM05], which handles BESs with a single equation block and underlies the distributed version of BISIMULATOR [JM04].

In this report, we propose MB-DSOLVE, a new distributed on-the-fly resolution algorithm for multiple block, alternation-free BESs. The algorithm is based upon a distributed breadth-first exploration of the *boolean graph* [And94] representing the dependencies between boolean variables of a BES. Our first application of MB-DSOLVE was the distributed on-the-fly model checking of alternation-free μ-calculus formulas (as computing engine for EVALUATOR), using the standard translation of the problem into a BES resolution [Lar88, And94]. The only existing distributed on-the-fly algorithm for solving this problem was proposed in [BLW02] and is based on *game graphs*, stemming from a game-based formulation of the problem [SS98]. The latest version of this algorithm, called PTCL1 and implemented in the model checker UPPDMC [HLL04], has an extension, called PTCL2, which is also able to handle μ-calculus formulas of alternation depth 2 [LSW03] and exhibits good performance on large state spaces, such as those of the VLTS benchmark suite[1]. Although the two algorithms MB-DSOLVE and PTCL1 are graph-based and therefore similar in spirit, MB-DSOLVE allows all machines involved in the distributed computation to handle simultaneously all equation blocks of a BES, thus potentially reaching a higher degree of concurrency than PTCL1, which at a given moment synchronizes and employs all machines to solve a precise part, called *component*, of

[1] `http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html`

the game graph. This intuition is confirmed experimentally on large states spaces from the VLTS benchmark.

Our second application of MB-DSOLVE was the distributed on-the-fly generation of conformance test cases from specifications and test purposes (both given as state spaces), following the approach advocated in the TGV tool [JJ05]. To achieve this, we proposed an encoding of the test generation problem as a BES resolution from which a diagnostic representing the *Complete Test Graph* (CTG) is built, and we implemented it within CADP in a tool named EXTRACTOR. This led to sequential and distributed test case generation functionalities, obtained by applying the algorithms of CÆSAR_SOLVE optimized for disjunctive/conjunctive BESs [Mat05a] and MB-DSOLVE, respectively. The BES technology proved again its usefulness: the performance of the sequential version of EXTRACTOR exhibits comparable performances with the optimized algorithms of TGV, and the distributed version scales smoothly to larger systems. As far as we know, this is the first attempt of building a distributed on-the-fly conformance test generator.

Related work. Distributed model checking has also been investigated in the framework of Linear Temporal Logic (LTL) [MP92]. The first distributed model checking algorithm proposed for LTL [LS99] was based upon a non-nested DFS traversal of the state space, which allowed to check only safety properties. Although its complexity was not estimated precisely, an implementation of this algorithm on networks of workstations (NOWS) improved the capabilities of SPIN [Hol03] for the analysis of systems exceeding the memory of a single machine.

This work was continued in [BBS01], leading to an extended algorithm able to perform a distributed nested DFS and thus to check full LTL properties. The new algorithm, which has a cubic time complexity and a linear space complexity in the size of the state space, allowed to verify systems that could not be handled by the sequential version of SPIN. This algorithm could still be improved by allowing several DFS traversals to be performed concurrently.

Report outline. Section 2 recalls basic definitions of BESs and describes in detail the MB-DSOLVE resolution algorithm. Section 3 translates the problems of model checking alternation-free μ-calculus formulas and of conformance test case generation into BES resolutions. Section 4 shows experimental data comparing the performance of the distributed tools with their sequential versions and with other similar distributed tools. Finally, Section 5 gives some concluding remarks and directions for future work.

2 Distributed local resolution of alternation-free BESs

We first define the framework underlying the manipulation of alternation-free BESs, and then we present the MB-DSOLVE algorithm for distributed on-the-fly resolution.

2.1 Alternation-free BESs

A Boolean Equation System (BES) [And94, Mad97], defined over $\mathcal{X}$, a set of boolean variables, is a tuple $B = (x, M_1, ..., M_n)$, where $x \in \mathcal{X}$ is a boolean variable and M_i are equation blocks ($i \in [1, n]$). Each block $M_i = \{x_{ij} \stackrel{\sigma_i}{=} op_{ij} \boldsymbol{X}_{ij}\}_{j \in [1, m_i]}$ is a set of minimal (*resp.* maximal) fixed point equations of sign $\sigma_i = \mu$ (*resp.* $\sigma_i = \nu$). The right-hand side of each equation ij of block M_i is a pure disjunctive or conjunctive formula obtained by applying a boolean operator $op_{ij} \in \{\vee, \wedge\}$ to a set of variables $\boldsymbol{X}_{ij} \subseteq \mathcal{X}$. The boolean constants false and true abbreviate the empty disjunction $\vee\emptyset$ and the empty conjunction $\wedge\emptyset$. A variable x_{ij} depends upon a variable x_{kl} if $x_{kl} \in \boldsymbol{X}_{ij}$. A block M_i depends upon a block M_k if some variable of M_i depends upon a variable defined in M_k. A block is *closed* if it does not depend upon any other blocks. A BES is *alternation-free* if there are no cyclic dependencies between its blocks; in this case, the blocks are sorted topologically such that a block M_i only depends upon blocks M_k with $k > i$. The *main* variable x must be defined in block M_1.

The semantics $[\![op\{x_1, ..., x_k\}]\!]\delta$ of a formula $op\{x_1, ..., x_k\}$ w.r.t. $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$ and a context $\delta : \mathcal{X} \rightarrow \mathbb{B}$, which must initialize all variables x_1, ..., x_k, is the boolean value $\delta(x_1)$ *op* ... *op* $\delta(x_k)$. The semantics $[\![M_i]\!]\delta$ of a block M_i w.r.t. a context δ is the σ_i-fixed point of a vectorial functional $\Phi_{i\delta} : \mathbb{B}^{m_i} \rightarrow \mathbb{B}^{m_i}$ defined as $\Phi_{i\delta}(b_1, ..., b_{m_i}) = ([\![op_{ij} \boldsymbol{X}_{ij}]\!](\delta \oslash [b_1/x_{i1}, ..., b_{m_i}/x_{im_i}]))_{j \in [1, m_i]}$, where $\delta \oslash [b_1/x_{i1}, ..., b_{m_i}/x_{im_i}]$ denotes a context identical to δ except for variables $x_{i1}, ..., x_{im_i}$, which are assigned values $b_1, ..., b_{m_i}$, respectively. The semantics of an alternation-free BES is the value of its main variable x given by the solution of M_1, i.e., $\delta_1(x)$, where the contexts δ_i are calculated as follows: $\delta_n = [\![M_n]\!][]$ (the context is empty because M_n is closed), $\delta_i = ([\![M_i]\!]\delta_{i+1}) \oslash \delta_{i+1}$ for $i \in [1, n-1]$ (a block M_i is interpreted in the context of all blocks M_k with $k > i$).

The *local* (or *on-the-fly*) resolution of an alternation-free BES $B = (x, M_1, ..., M_n)$ consists in computing the value of x by exploring the right-hand sides of the equations in a demand-driven way, without explicitly constructing the blocks. Several sequential on-the-fly BES resolution algorithms are available [And94, Mad97, DSC99]; here we adopt the approach proposed in [And94], which formulates the resolution problem in terms of a *boolean graph* representing the dependencies between boolean variables. A boolean graph is a triple $G = (V, E, L)$, where $V = \{x_{ij} \mid i \in [1, n] \wedge j \in [1, m_i]\}$ is the set of *vertices* (boolean variables), $E : V \rightarrow 2^V$, $E = \{x_{ij} \rightarrow x_{kl} \mid x_{kl} \in \boldsymbol{X}_{ij}\}$ is the set of *edges* (dependencies between variables), and $L : V \rightarrow \{\vee, \wedge\}$, $L(x_{ij}) = op_{ij}$ is the *vertex labeling* (disjunctive or conjunctive). An example of BES with three blocks and its associated boolean graph is shown on Figure 2.

The resolution of variable x amounts to perform a forward exploration of the dependencies going out of x, intertwined with a backward propagation of stable variables (whose value is determined) along dependencies; the resolution terminates either when x becomes stable (after propagation of some stable successors) or when the portion of boolean graph reachable from x is completely explored.

2.2 Distributed local resolution algorithm

The algorithm we propose for distributed on-the-fly resolution of multiple block, alternation-free BESs is called MB-DSOLVE (*Multiple Block Distributed SOLVEr*). We consider a computing architecture consisting of P machines (called *nodes*), numbered from 1 to P, intercon-

nected via a network and communicating by message-passing. Examples of such architectures are NOWs and clusters of PCs. Here we focus on homogeneous architectures, in which all machines are equipped with the same processor, memory, and operating system. For simplicity, we assume that each node executes a single instance of MB-DSOLVE, called *worker*, although in practice there may be several worker instances running on the same node.

The resolution of an alternation-free BES $B = (x, M_1, ..., M_n)$ is done by means of two breadth-first traversals of the corresponding boolean graph, starting from x: a forward exploration of the dependencies of the variables being solved, and a backward propagation of stable variables. The traversals are done in a distributed manner, each worker node being responsible for solving a subset of the boolean variables defined in B, determined using a static hash function.

In addition to workers, a special process, called *supervisor*, usually executed on the end-user node (numbered 0), is responsible for initializing the distributed computation by copying files and launching workers on remote nodes, for collecting statistics about the BES resolution, and for detecting (normal and abnormal) termination. A description of the supervisor associated to the DSOLVE algorithm for solving single block BESs can be found in [JM05]. Its extension to multiple block BESs involves a multiplexing of the data structures for each equation block and of the distributed termination detection (DTD) algorithm in order to detect the partial termination of each block and the global termination of the resolution. For simplicity, we do not present here the full DTD algorithm, but rather we indicate how the DTD algorithm given in [JM05] can be extended to deal with multiple block BESs.

The function MB-DSOLVE, shown on Figure 1, describes the behavior of a worker node $i \in [1, P]$ participating to the distributed resolution on P nodes of the main variable $x \in V$ of an alternation-free BES $B = (x, M_1, ..., M_n)$ represented by its boolean graph $G = (V, E, L)$. The set of successors of a vertex x is noted $E(x)$. We assume that G is not entirely constructed, but is given implicitly by its successor function E, which allows to explore G on-the-fly. Boolean variables are distributed to worker nodes by means of a static hash function $h : V \rightarrow [1, P]$. The index of the block defining a variable is given by a function $b : V \rightarrow [1, n]$, $b(x_{ij}) = i$. Upon termination, the function MB-DSOLVE returns the boolean value computed for the main variable x.

To each block k are associated, locally to node i, several information: a set $S_k^i \subseteq V$ containing the visited vertices; a BFS queue W_k^i storing the vertices visited but not explored yet; a set $B_k^i \subseteq V$ containing stable variables, whose values must be back-propagated; a set $R_k^i \subseteq V$ containing unstable variables with interblock predecessor dependencies (i.e., variables defined in block k and occurring in the rhs of some equation of another block l); and a set $Q_k^i \subseteq E$ storing the interblock transitions going from block k and pending to be explored. The counter $exp\_req_k^i$, initialized to 0, gives the number of interblock transitions starting from variables in block k locally to node i, which needs to be eventually traversed by propagating the values of stable target variables. To each vertex y_k^j are associated four fields: a counter $c(y_k^j)$, which keeps the number of y_k^j's successors that must be stabilized in order to make the value of y_k^j stable, its boolean value $v(y_k^j)$, a set $d(y_k^j)$ containing the vertices that currently depend upon y_k^j, and a boolean $stable(y_k^j)$ indicating if y_k^j has a stable value (i.e., if $c(y_k^j) = 0$ or if y_k^j belongs to a completely explored and stabilized portion of block k). These fields are set up by $initialize(y_k^j)$ as follows: the counter $c(y_k^j)$ is set to $|E(y_k^j)|$ if

```
1  MB-DSOLVE(x,(V,E,L),n,P,h,i) → 𝔹 :
2    if h(x) = i then
3      S^i_{b(x)} := {x};  W^i_{b(x)} := put(x, nil);
4      initialize(x)
5    endif;  term^i_{b(x)} := false;
6    while ¬term^i_{b(x)} do
7      if IRECEIVE(msg_i, sender_i) then
8        READ(msg_i, sender_i)
9      elsif (l_i := HASSTABILITY) ≤ n then
10       STABILIZATION(l_i)
11     elsif (k_i := HASEXPANSION) ≥ 1 then
12       EXPANSION(k_i)
13     else
14       RECEIVE(msg_i, sender_i);
15       READ(msg_i, sender_i)
16     endif
17   endwhile;
18   returnv(x)

19 READ(m_i, s_i):
20   case m_i is
21     Exp(x^{s_i}_k, y^i_l) →  if k ≠ l then
22           Q^i_l ∪:= {(x^{s_i}_k, y^i_l)}
23           else EXPAND(x^{s_i}_k, y^i_l) endif
24     Evl(x^i_k, y^{s_i}_l) → if k ≠ l then
25           exp_req^i_k −:= 1 endif;
26           if ¬stable(x^i_k) then
27             STABILIZE(x^i_k, y^{s_i}_l) endif
28   endcase

29 STABILIZATION(l):
30   while B^i_l ≠ ∅ ∨ (term^i_l ∧ R^i_l ≠ ∅) do
31     if B^i_l ≠ ∅ then y^i_l := get(B^i_l);
32       B^i_l \:= {y^i_l} else y^i_l := get(R^i_l);
33     R^i_l \:= {y^i_l} endif;
34     forall w^j_k ∈ d(y^i_l) ∧ (B^i_l ≠ ∅ ∨ k ≠ l)
35       ∧¬term^i_{b(x)} ∧ ¬stable(w^j_k) do
36       if h(w^j_k) = i then
37         if k ≠ l then exp_req^i_k −:= 1
38         endif; STABILIZE(w^j_k, y^i_l)
39       else SENDING(Evl(w^j_k, y^i_l), h(w^j_k))
40       endif
41     endfor;  d(y^i_l) := ∅
42   endwhile

43 STABILIZE(w^i_k, y^j_l):
44   if ((L(w^i_k) = ∨) ∧ v(y^j_l))∨
45   ((L(w^i_k) = ∧) ∧ ¬v(y^j_l)) then
46     s(w^i_k) := y^j_l;  c(w^i_k) := 0;
47     stable(w^i_k) := true
48   else c(w^i_k) −:= 1 endif;
49   if stable(w^i_k) then B^i_k ∪:= {w^i_k};
50     if w^i_k ∈ R^i_k then R^i_k \:= {w^i_k}
51     endif;  term^i_{b(x)} := stable(x)
52   endif

53 EXPANSION(k):
54   if W^i_k = nil then
55     forall (x^j_l, y^i_k) ∈ (Q^i_k)
56         ∧¬term^i_{b(x)} do
57       if j ≠ i ∨ ¬stable(x^j_l)
58         then EXPAND(x^j_l, y^i_k)
59       elsif l ≠ k then
60         exp_req^i_l −:= 1 endif
61     endfor
62   else
63     x^i_k := head(W^i_k); W^i_k := tail(W^i_k);
64     forall y^j_l ∈ E(x^i_k) ∧ ¬term^i_{b(x)}
65         ∧¬stable(x^i_k) do
66       if k ≠ l then exp_req^i_k +:= 1
67       endif;
68       if h(y^j_l) = i then
69         if k ≠ l then
70           Q^i_l ∪:= {(x^i_k, y^j_l)}
71         else EXPAND(x^i_k, y^j_l) endif
72       else
73         SENDING(Exp(x^i_k, y^j_l), h(y^j_l))
74       endif
75     endfor
76   endif

77 EXPAND(x^j_k, y^i_l):
78   if y^i_l ∉ S^i_l then
79     S^i_l ∪:= {y^i_l};  initialize(y^i_l);
80     if c(y^i_l) ≠ 0 then
81       W^i_l := put(y^i_l, W^i_l)
82     else stable(y^i_l) := true endif
83   endif;
84   if k ≠ l ∧ y^i_l ∉ R^i_l then
85     R^i_l ∪:= {y^i_l} endif;
86   if stable(y^i_l) then
87     if y^i_l ∈ R^i_l then R^i_l \:= {y^i_l}
88     endif;
89     if h(x^j_k) = i then
90       if k ≠ l then exp_req^i_l −:= 1
91       endif; STABILIZE(x^j_k, y^i_l)
92     else B^i_l ∪:= {y^i_l};
93       d(y^i_l) ∪:= {x^j_k} endif
94   else d(y^i_l) ∪:= {x^j_k} endif
```

Figure 1: Distributed local resolution of a multiple block, alternation-free BES using its boolean graph

$\sigma_k = \mu$ and $L(y_k^j) = \wedge$ or $\sigma_k = \nu$ and $L(y_k^j) = \vee$, and to 1 otherwise; $v(y_k^j)$ is set to false if $\sigma_k = \mu$ and to true otherwise; $d(y_k^j)$ is initially empty; and $stable(y_k^j)$ is initially false.

At each iteration of the main while-loop (lines 6–17), received messages are processed first (lines 7–8). Then, the block with minimal index $l_i \in [1, n]$ that has stable variables not propagated yet (i.e., $B_l^i \neq \emptyset$) or that is completely explored but contains interblock predecessor dependencies not yet traversed by backward propagation of stable values (i.e., $term_l^i \wedge R_l^i$), is returned by HasStability and stabilized by Stabilization(l_i) (lines 9–10). If such block does not exist, the block k_i with maximal index that has a non-empty Bfs queue (i.e., $W_k^i \neq nil$) or that is completely explored and contains pending resolution requests on unvisited variables (i.e., $exp\_req_k^i = 0 \wedge B_k^i = R_k^i = \emptyset \wedge term_k^i \wedge Q_k^i \neq \emptyset$), is returned by HasExpansion and explored with Expansion(k_i) (lines 11–12). Finally, if there is no more work on any block, the worker i remains blocked on reception, waiting, e.g., for termination detection messages sent by the supervisor (lines 14–15).

This strategy of choosing the next block to be processed aims at improving the speed and memory consumption of the distributed resolution: stabilizing the block with the minimal index l_i, which is "closest" to the block containing the main variable, makes the resolution converge faster (e.g., when the main variable can be stabilized by some variable in block l_i), and expanding the block with the maximal index k_i, which currently has no "descendant" blocks being explored, prevents the Bfs queues of the other intermediate blocks from becoming large (which would increase, at a later stage, the amount of variables under exploration).

The boolean graph resolution begins with the successor generation (i.e., expansion) of main variable x (lines 63–75). Successors are then traversed in a breadth first (Bfs) manner, and each of the new visited successor variables is either added to the set of interblock transitions going from block k and pending to be explored (line 70), either added to Bfs heap (line 71) using primitive Expand (lines 77–94) locally to node i or sent to a remote node (lines 72–73) with a message Exp. The other novelty of Mb-DSolve compared to DSolve is that primitive Expansion(k) explores interblock transitions whose destination block is k (lines 54–62) when the current visited portion of block k has completely been explored and detected stable by distributed partial termination detection. It does so by treating all such interblock transitions (lines 55–56) waiting to be explored, in order to minimize the number of partial termination detections of block k which involve costly internode synchronization.

Concerning the stabilization of variables, whose operation has a higher priority w.r.t. expansion, it is composed of two parts: one being focused on detection of block portion stability (i.e., passive stabilization) part of the distributed termination detection algorithm, and the other one being focused on the propagation of stable variables (i.e., active stabilization) (lines 30–42) either extracted from same block l (B_l^i) or from remote block l (R_l^i). Primitive Stabilize (lines 43–52) is then invoked to update the value of variable w_k^i depending on the propagated value of y_l^j.

Asynchronous non-blocking communication. In order to enable a maximal overlapping of communications and computations, send and receive operations are made as much as possible in a *non-blocking* and *asynchronous* manner. Non-blocking means that a communication operation is not blocked until its termination. An example is the primitive

IRECEIVE, which tries to immediately receive a message, or exits otherwise (line 7). Asynchronous means that a communication operation does not depend on the synchronization with another remote communication attempt. Such asynchronism can be obtained by a buffering mechanism on the sending and receiving sides. Hence, communication becomes a task performed concurrently with the other global and local computation tasks. The bottleneck of model checking being memory consumption, the communication layer on which relies MB-DSOLVE is based on bounded buffers, managed explicitly in the distributed algorithm to allow a fine tuning of memory usage. An example is the primitive SENDING, which tries to send immediately a message, and keeps trying until successful emission, possibly interleaving emission attempts with reception attempts and passive waiting on communication events (lines 39 and 73).

Distributed generation of diagnostics. The result of the distributed BES resolution must be accompanied by a diagnostic (example or counterexample) which provides the minimal amount of information needed for understanding the value computed for the main variable x. MB-DSOLVE computes diagnostic information in the form of a boolean subgraph rooted at x, following the approach proposed in [Mat00]. The minimal information necessary for producing the diagnostic is stored as $s(w_k^i)$ (line 46), indicating the successor of variable w_k^i that stabilized it after a backward propagation (e.g., a true successor of an $\vee$-variable). This provides an implicit, distributed representation of the diagnostic, which can be explored on-the-fly once the resolution has finished. An application of this mechanism will be given in Section 3.2, where the construction of a positive diagnostic (example) explains the conformance of a specification w.r.t a test purpose.

Distributed termination detection for equation blocks. The variable $term_{b(x)}^i$ is set to true when distributed termination of the BES resolution is detected. Conditions of termination are: either the main variable x has been stabilized ($c(x) = 0$) during backward propagation (line 51), or the boolean graph has been completely explored, i.e., all local working sets of variables are empty ($\forall i \in [1, P], k \in [1, n] \cdot W_k^i = nil \wedge B_k^i = R_k^i = Q_k^i = \emptyset \wedge exp\_req_k^i = 0$) and no more messages are transiting through the network. MB-DSOLVE implements a mechanism for detecting the termination of the partial resolution of a block k on a node i ($term_k^i$). Contrary to the PTCL2 algorithm underlying the UPPDMC model checker, in which all nodes cooperate for solving a single block at a given time, our approach allows a fine-grain distribution of the BES resolution by allowing each node to work at the resolution of several blocks at the same time. The detection of the inactivity of nodes w.r.t. the resolution of a particular block improves the convergence of the distributed BES resolution by increasing the probability of finding a partially solved block from which stable values can be propagated backwards.

Our DTD is based on the four-counter method presented in [Mat87] on a star-shaped topology with a central agent (the supervisor) whose role is asymmetric to worker nodes [HK91]. Activity status of workers is regularly sent to the supervisor, which therefore has a global view of the computation and is able to initiate the DTD for an equation block with higher probability of success than traditional ring-based DTD algorithms.

Example of distributed local resolution. Figure 2 shows an alternation-free BES containing three equation blocks and its corresponding boolean graph, partitioned by the hash function onto three worker nodes P_1, P_2, P_3. Worker P_1 starts the exploration of the boolean graph by expanding the main variable $x_{1,1}$, whose successors $x_{2,1}$ and $x_{1,2}$ are solved locally by P_1 and sent to node P_3, respectively. Variables $x_{2,1}$ and $x_{1,2}$ can be expanded in parallel with the effect that $x_{3,1}$ is sent to node P_2, $x_{1,3}$ is sent to node P_3 (note that $x_{1,3}$ has an interblock predecessor dependency with $x_{2,1}$), and $x_{1,3}$, $x_{2,2}$ are solved locally by node P_3. Since $x_{1,3}$ is an $\vee$-vertex without successors (i.e., a constant false), its value is stable and can be propagated through backward dependencies to its predecessors; this stabilizes to false the $\wedge$-vertex $x_{2,1}$, but not the $\vee$-vertex $x_{1,2}$. The further propagation of $x_{2,1}$ stabilizes $x_{1,1}$ to false. To illustrate the fixed point operator, we can emphasize that the final value of $x_{3,1}$ is true since the variable is defined by a maximal fixed point boolean equation, and has an initial value true. Moreover, we should make clear that the block partitioning of the BES is specific to a problem resolution and totally independent from the hashing function used to distribute the BES onto remote computing nodes.

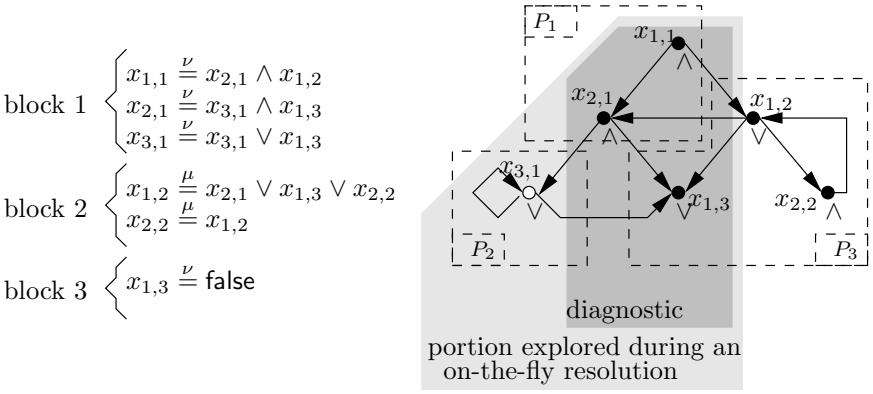


Figure 2: A multiple block, alternation-free BES, its partitioned boolean graph, and the result of a distributed on-the-fly resolution for $x_{1,1}$ on three nodes. Black and white vertices denote false and true variables, respectively.

The light grey area delimits the portion of the boolean graph that was explored in order to complete the resolution of $x_{1,1}$. The dark grey area delimits the diagnostic (counterexample) associated to $x_{1,1}$, which is obtained by choosing, for each $\wedge$-variable x, the successor $s(x)$ which stabilized it to false, computed by MB-DSOLVE during propagation.

Complexity in time, memory, and messages. MB-DSOLVE is based on the theory of boolean graphs underlying the sequential resolution algorithms for alternation-free BESs [And94]. It consists roughly of two intertwined traversals (forward and backward) of the boolean graph, with a worst-case time complexity $O(|V|+|E|)$. The same bound applies for memory complexity, because of the backward dependencies stored during resolution for eventual propagations of stable variables. The communication cost of MB-DSOLVE can also be estimated, assuming that messages (excluding those for DTD) are sent for each *cross-dependency* (i.e., edge $(x, y) \in E \mid h(x) \neq h(y)$). Since the hash function h shares variables

equally among nodes without *a priori* knowledge about locality, it also shares dependencies equally. Thus, the number of cross-dependencies can be evaluated to $((P-1)/P)\cdot|E|$, since statistically only $|E|/P$ edges will be local to a node. Hence, the message complexity is $O(|E|)$, the worst-case being obtained with two messages (expansion and stabilization) exchanged per cross-dependency, i.e., at most $2\cdot((P-1)/P)\cdot|E|$ messages. Our DTD algorithm has the same worst-case message complexity, but in practice it reveals to be very efficient, with only 0.01% of total exchanged messages used for DTD; this is due to the supervisor, which has an up-to-date global view of the computation.

3 Applications

We illustrate in this section the application of the MB-DSOLVE algorithm on two analysis problems defined on *Labeled Transition Systems* (LTSs): model checking of alternation-free μ-calculus formulas and generation of conformance test cases. An LTS is a tuple (S, A, T, s_0) containing a set of states S, a set of actions A, a transition relation $T \subseteq S \times A \times S$ and an initial state $s_0 \in S$. A transition $(s, a, s') \in T$, noted also $s \xrightarrow{a} s'$, states that the system can move from state s to state s' by executing action a (s' is an a-successor of s). Both problems can be formulated as the resolution of a multiple block, alternation-free BES, the second one essentially relying upon diagnostic generation for BESs. By applying MB-DSOLVE as BES resolution engine, we obtain distributed versions of the on-the-fly model checker EVALUATOR 3.5 [MS03, Mat03] and the on-the-fly test generator TGV [JJ05] of the CADP toolbox [GLM02].

3.1 Model checking for alternation-free mu-calculus

Modal μ-calculus [Koz83] is a powerful fixed point based logic for specifying temporal properties on LTSs. Its formulas are defined by the following grammar (where X is a propositional variable):

$$\phi ::= \mathsf{false} \mid \mathsf{true} \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle\, \phi \mid [a]\, \phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

Given an LTS (S, A, T, s_0), a formula ϕ denotes a set of states, defined as follows: boolean formulas have their usual set interpretation; modalities $\langle a \rangle\, \phi$ (*resp.* $[a]\, \phi$) denote the states having some (*resp.* all) a-successors satisfying ϕ; fixed point formulas $\mu X.\phi$ (*resp.* $\nu X.\phi$) denote the minimal (*resp.* maximal) solution of the equation $X = \phi$, interpreted over 2^S. The local model checking problem amounts to establish whether the initial state s_0 of an LTS satisfies a formula ϕ, i.e., belongs to the set of states denoted by ϕ.

The alternation-free fragment of the modal μ-calculus, noted L^1_μ [EL86], is obtained by forbidding mutual recursion between minimal and maximal fixed point variables. L^1_μ benefits from model checking algorithms whose complexity is linear in the size of the LTS (number of states and transitions) and the formula (number of operators), while still allowing to express useful properties, since it subsumes CTL [CES86] and PDL [FL79]. The model checking of L^1_μ formulas on LTSs can be encoded as the resolution of an alternation-free BES [And94]. We illustrate the encoding by considering the following formula, which states that the emission

snd of a message is eventually followed by its reception *rcv* ('$-$' stands for 'any action' and '$\overline{a}$' stands for 'any action different from a'):

$$\nu X.([snd]\,\mu Y.(\langle - \rangle\,\mathsf{true} \wedge [\overline{rcv}]\,Y) \wedge [-]\,X)$$

The formula is translated first into a specification in HML with recursion [Lar88], which contains two blocks of modal equations:

$$\{X \stackrel{\nu}{=} [snd]\,Y \wedge [-]\,X\}, \{Y \stackrel{\mu}{=} \langle - \rangle\,\mathsf{true} \wedge [\overline{rcv}]\,Y\}$$

Then, each modal equation block is converted into a boolean equation block by 'projecting' it on each state of the LTS:

$$\{X_s \stackrel{\nu}{=} \bigwedge_{s \stackrel{snd}{\rightarrow} s'} Y_{s'} \wedge \bigwedge_{s \rightarrow s'} X_{s'}\}_{s \in S}, \{Y_s \stackrel{\mu}{=} \bigvee_{s \rightarrow s'} \mathsf{true} \wedge \bigwedge_{s \stackrel{rcv}{\not\rightarrow} s'} Y_{s'}\}_{s \in S}$$

A boolean variable X_s (*resp.* Y_s) is true iff state s satisfies the propositional variable X (*resp.* Y). Thus, the local model checking of the initial formula amounts to compute the value of variable X_{s_0} by applying a local BES resolution algorithm. This method underlies the on-the-fly model checker EVALUATOR 3.5 [MS03, Mat03] of CADP [GLM02], which handles formulas of L^1_μ extended with PDL-like modalities containing regular expressions over transition sequences. As verification engine, EVALUATOR 3.5 uses the BES resolution library CÆSAR_SOLVE [Mat03, Mat05a], which offers four algorithms implementing different strategies of exploring boolean variable dependencies (depth-first, breadth-first, etc.). Since MB-DSOLVE was designed to be compliant with the CÆSAR_SOLVE application programming interface, it allowed to immediately obtain a distributed version of EVALUATOR 3.5.

3.2 Conformance test case generation

Conformance testing aims at establishing that the implementation under test (IUT) of a system is correct w.r.t. a specification. We consider here the conformance test approach advocated in the pioneering work underlying the TGV tool [JJ05]. We give only a brief overview of the theory used by TGV and focus on the algorithmic aspects of test selection, with the objective of reformulating them in terms of BES resolution and diagnostic generation.

The IUT and the specification are modelled as Input-output LTSs (IOLTSs), which distinguish between inputs and outputs: e.g., the actions of the IOLTS of the specification $M = (S^M, A^M, T^M, s_0^M)$ are partitioned into $A^M = A_I^M \cup A_O^M \cup \{\tau\}$, where A_I^M (*resp.* A_O^M) are input (*resp.* output) actions and τ is the internal (unobservable) action. Intuitively, input actions of the IUT are controllable by the environment, whereas output actions are only observable. In practice, tests observe the execution traces consisting of observable actions of the IUT, but can also detect *quiescence*, which can be of three kinds: deadlock (states without successors), outputlock (states without outgoing output actions), and livelock (cycles of internal actions). For an IOLTS M, quiescence is modelled by a *suspension automaton* $\Delta(M)$, an IOLTS which marks quiescent states by adding self-looping transitions labeled by a new output action δ. The traces of $\Delta(M)$ are called suspension traces of M. The conformance relation **ioco** [Tre96] between the IUT and the specification M states that after

executing each suspension trace of M, the (suspension automaton of the) IUT exhibits only those outputs and quiescences that are allowed by M.

Test generation requires a determinization of M, since two sequences with the same traces of observable actions cannot be distinguished. Since quiescence must be preserved, determinization must take place after the construction of the suspension automaton $\Delta(M)$.

A *test case* is an IOLTS $TC = (S^{TC}, A^{TC}, T^{TC}, s_0^{TC})$ equipped with three sets of trap states $\mathbf{Pass} \cup \mathbf{Fail} \cup \mathbf{Inconc} \subseteq S^{TC}$ denoting verdicts. The actions of TC are partitioned into $A^{TC} = A_I^{TC} \cup A_O^{TC}$, where $A_O^{TC} \subseteq A_I^M$ (TC emits only inputs of M) and $A_I^{TC} \subseteq A_O^{IUT} \cup \{\delta\}$ (TC captures outputs and quiescences of the IUT). A test case must satisfy several structural properties: **Fail** and **Inconc** states are only directly reachable by inputs; from each state a verdict is reachable; in every state, no choice is allowed between two outputs or an input and output (controllability); each state, where an input is possible, is input complete. The execution of a test case TC against an IUT is modelled by a parallel composition $TC||IUT$ with synchronization on common observable actions, verdicts being determined by the trap states reached by a maximal trace of $TC||IUT$.

The test generation technique of TGV is based upon *test purposes*, which allow to guide the test case selection. A test purpose is a deterministic and complete IOLTS $TP = (S^{TP}, A^{TP}, T^{TP}, s_0^{TP})$, with the same actions as the specification $A^{TP} = A^M$, and equipped with two sets of trap states $Accept^{TP}$ and $Reject^{TP}$, which are used to select targeted behaviours and to cut the exploration of M, respectively. TGV decomposes the test generation process into several steps, most of which are performed on-the-fly during forward traversals of the IOLTS M modelling the specification. Here we focus on the computation of the *complete test graph* (CTG), which contains all test cases corresponding to a specification and a test purpose, and therefore can serve as a criterion for comparison and performance measures. Controllable test cases can be produced from a CTG by applying specific algorithms [JJ05].

The CTG is produced by TGV as the result of three operations, all performed on-the-fly: (a) computation of the synchronous product $SP = M \times TP$ between the IOLTSs of the specification and the test purpose, in order to mark accepting and refusal states; (b) suspension and determinization of SP, leading to $SP^{vis} = det(\Delta(SP))$, which keeps only visible behaviours and quiescence; (c) selection of the test cases denoting the behaviours of SP^{vis} accepted by TP, which form the CTG. The main operation (c) roughly consists in computing $L2A$ (*lead to accept*), the subset of the states of SP^{vis} from which an accepting state is reachable, checking whether the initial state of SP^{vis} belongs to $L2A$ (which indicates the existence of a test case), and defining, based upon $L2A$, the sets **Pass**, **Fail**, and **Inconc** corresponding to the verdicts. This is the operation we chose to encode as a BES resolution with diagnostic.

Assuming that the accepting states of SP^{vis} are marked by self-looping transitions labeled by an action acc (as it is done in practice), the reachability of an accepting state is denoted by the following μ-calculus formula:

$$\phi_{acc} = \mu Y.(\langle acc \rangle \, \mathsf{true} \vee \langle - \rangle \, Y)$$

The set $L2A$ contains all states satisfying ϕ_{acc}. It can be computed in a backwards manner by using a fixed point iteration to evaluate ϕ_{acc} on SP^{vis}. Since this requires the prior com-

putation of SP^{vis}, we seek another solution suitable for on-the-fly exploration, by considering the formula below:

$$\phi_{l2a} = \nu X.(\phi_{acc} \wedge [-](\phi_{acc} \Rightarrow X))$$

Formula ϕ_{l2a} has the same interpretation as ϕ_{acc}, meaning that its satisfaction by the initial state of SP^{vis} denotes the existence of a test case. Moreover, the on-the-fly evaluation of ϕ_{l2a} on a state s satisfying ϕ_{acc} requires the inspection of every successor s' of s and, if it satisfies ϕ_{acc}, the recursive evaluation of ϕ_{l2a} on s', etc., until all states in $L2A$ have been explored.

The CTG could be obtained as the diagnostic produced by an on-the-fly model checker (such as EVALUATOR) for the formula ϕ_{l2a}. However, to annotate the CTG with verdict information and to avoid redundancies caused by the two occurrences of ϕ_{acc} present in ϕ_{l2a}, a finer-grained encoding of the problem in terms of a BES resolution with diagnostic is preferred. The corresponding BES denotes the model checking problem of ϕ_{l2a} on SP^{vis}, by applying the translation given in Section 3.1 (s, s' are states of SP^{vis}):

$$\{X_s \stackrel{\nu}{=} Y_s \wedge \bigwedge_{s \to s'}(Z_{s'} \vee X_{s'})\}, \{Y_s \stackrel{\mu}{=} \bigvee_{s \stackrel{acc}{\to} s'} \mathsf{true} \vee \bigvee_{s \to s'} Y_{s'}\},$$

$$\{Z_s \stackrel{\nu}{=} \bigwedge_{s \stackrel{acc}{\to} s'} \mathsf{false} \wedge \bigwedge_{s \to s'} Z_{s'}\}$$

If $X_{s_0^{vis}}$ is true, then a positive diagnostic (example) can be exhibited in the form of a boolean subgraph [Mat00] containing, for each conjunctive variable (such as X_s and Z_s), all its successor variables, and for each disjunctive variable (such as Y_s) only one successor which evaluates to true. This diagnostic can be obtained by another forward traversal of the boolean graph portion already explored for evaluating $X_{s_0^{vis}}$. We turn the diagnostic into a CTG in the following manner: we associate a state of the CTG to each variable X_s; we produce an accepting transition going out of X_s only if the first subformula in the right-hand side of the equation defining Y_s is true (i.e., s has an *acc*-successor); we produce a transition $X_s \stackrel{a}{\to} X_{s'}$ for each state s' such that $Z_{s'}$ is false. Note that the diagnostic for variables Z_s does not need to be explored. Additional verdict information in the form of refuse and inconclusive transitions is produced in a similar way during diagnostic generation, since the information needed for verdicts in the CTG is local w.r.t. states of $L2A$ [JJ05].

In the discussion above, formula ϕ_{l2a} was evaluated on the IOLTS SP^{vis} obtained after suspension and determinization of SP; however, these two operations can also be performed *after* test case selection, by taking care to handle the inconclusive verdicts as described in [JJ05]. In other words, the BES based generation procedure sketched above can be applied directly on the synchronous product SP between the specification and the test purpose, producing a 'raw' CTG, which is subsequently suspended and determinized to yield the final CTG. This procedure underlies the prototype EXTRACTOR tool we developed within CADP for producing raw CTGs, which are then processed by the DETERMINATOR tool [HJ03], resulting in CTGs strongly bisimilar to those produced by TGV. Although this ordering of operations is not the most efficient one for sequential on-the-fly test case generation (since the IOLTS of the specification can contain large amounts of τ-transitions), it appears to be suitable in the distributed setting, because it leads to large BESs, which are solved efficiently by using MB-DSOLVE.

4 Implementation and experiments

The model checker Evaluator 3.5 [MS03, Mat03] and the prototype test case generator Extractor (see Figure 3) have been developed within Cadp [GLM02] by using the generic Open/Cæsar environment [Gar98] for on-the-fly exploration of Ltss.

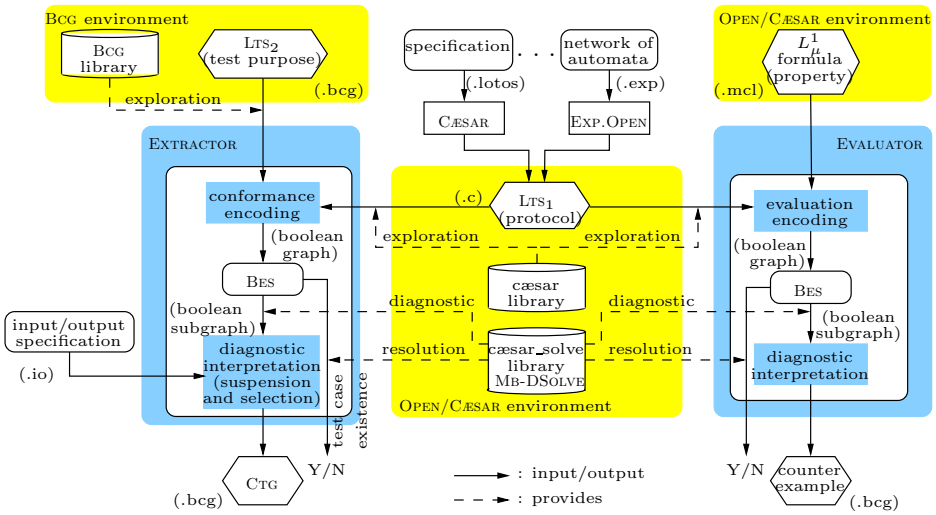


Figure 3: The distributed on-the-fly tools Evaluator and Extractor

Evaluator (*resp.* Extractor) consists of two parts: a front-end, responsible for encoding the verification of the L^1_μ formula (*resp.* the test selection guided by the test purpose LTS_2) on LTS_1 as a Bes resolution, and for producing a counterexample (*resp.* a Ctg) by interpreting the diagnostic provided by the Bes resolution; and a back-end, responsible of Bes resolution, playing the role of verification engine.

Sequential and distributed versions of Evaluator and Extractor are obtained by using as back-end either the sequential algorithms of the Cæsar_Solve library [Mat03, Mat05a], or the Mb-DSolve algorithm, respectively.

Mb-DSolve (15 000 lines of C code) is a conservative extension of the distributed resolution algorithm DSolve [JM05] for single block Bess and was implemented by using the Open/Cæsar environment. The size of the worker and supervisor processes is roughly the double in Mb-DSolve w.r.t. DSolve. For communication, Mb-DSolve is based on the Cæsar_Network library of Cadp, which offers a set of 40 primitives finely-tuned for verification problems, such as non-blocking asynchronous emission/reception of messages through Tcp/Ip sockets and explicit memory management by means of bounded communication buffers.

We present in this section experimental measures comparing the distributed versions of Evaluator and Extractor with their sequential counterparts, as well as with the UppDmc distributed model checker and the Tgv test case generator, respectively.

4.1 Performance of distributed model checking

We begun our experiments by checking two simple properties expressed in modal μ-calculus, namely absence of deadlock ($\nu X.\,(\langle - \rangle \,\mathsf{true} \wedge [-]\,X)$) and presence of livelock ($\mu X.\,(\nu Y.\,(\langle \tau \rangle\, Y) \vee \langle - \rangle\, X)$), on a cluster of 21 XEON 2.4 GHz PCs, with 1.5 GB of RAM, running LINUX, and interconnected by a 1 Gigabit ETHERNET network. These properties were checked on the nine largest LTSs of the VLTS benchmark. Figure 4 shows the speedup (a) and memory ratio (b) between distributed EVALUATOR and its sequential optimized version based on the resolution algorithms for disjunctive/conjunctive BESs present in CÆSAR_SOLVE. Each point on each curve represents the average of ten experiments excluding the minimum and extremum values. The sequential version is very fast in finding counterexamples (e.g., for *vasy_2581_11442*, *vasy_4220_13944*, *vasy_4338_15666*, *vasy_6120_11031*, *vasy_11026_24660*, and *vasy_12323_27667* which contain deadlocks), as can be observed with the six lower speedup curves close to x-axis on Figure 4 (a). However, when it is necessary to explore the underlying BES entirely (e.g., for *cwi_7838_59101* and *vasy_6020_19353* which do not contain deadlocks), the distributed version becomes interesting, allowing close to linear speedups and a good scalability as the number of workers increases.

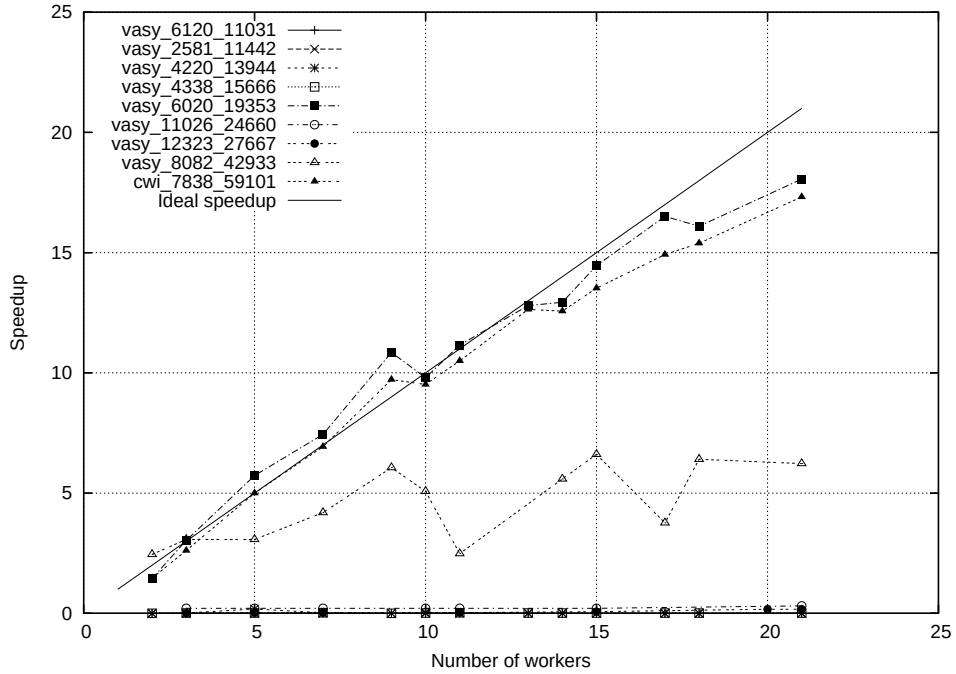
The (slightly) super linear speedups are due to the use of hash tables for storing sets of boolean variables. Since the balancing of these tables is not perfect, some collision lists tend to become large; the effect of this phenomenon is stronger on the sequential version of the tool (which uses a single large hash table) than on its distributed version (which uses P smaller tables).

The memory overhead (see Figure 4 (b)) of the distributed version is not really affected by an increasing number of workers and remains low, with a memory consumption averaged over all nodes around 5 times bigger than the sequential one. Moreover, we observed almost no idle time, the distributed computation using systematically around 99% of the CPUs capacity on each worker node. This is partly a consequence of the well-balanced data partitioning induced by the static hash function, and indicates a good overlapping between communications and computations.

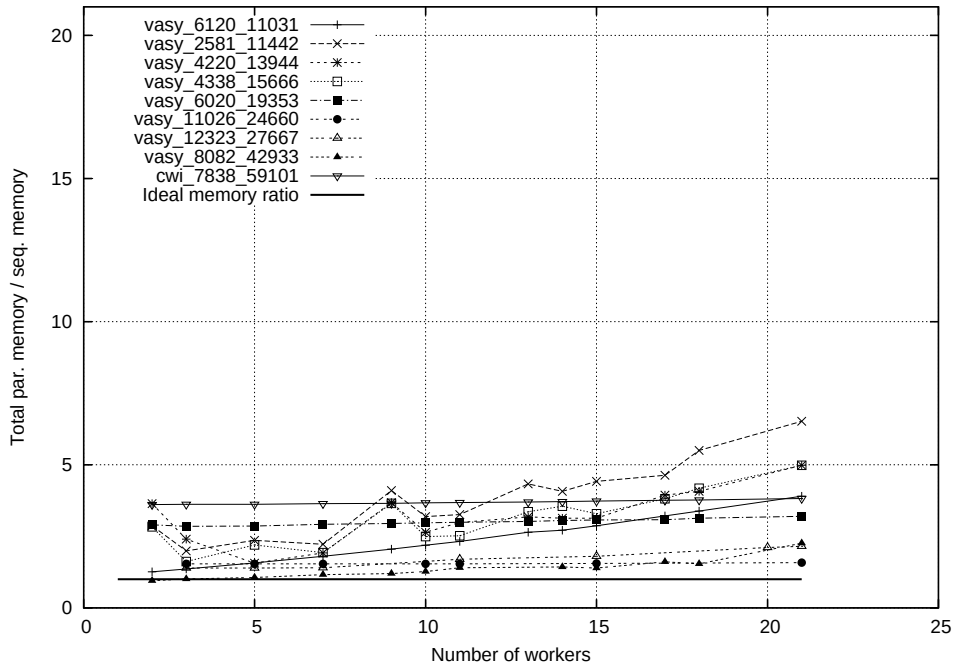
We have also compared time and memory performances of distributed EVALUATOR against the UPPDMC model checker based on game graphs. Results are given in Table 1, where each of the seven VLTS examples considered (e.g., *vasy_2581_11442*, an LTS with $2\,581 \cdot 10^3$ states and $11\,442 \cdot 10^3$ transitions) is checked for deadlock and livelock. Distributed EVALUATOR is very fast in detecting counterexamples, which explains most of the improvements in time and memory compared to UPPDMC running two threads on 25 bi-PENTIUM III 500 MHz, with 512 MB of RAM. When the whole BES (*resp.* game graph) has to be explored (e.g., for *vasy_6020_19353*), the execution time is closer to that of UPPDMC, but always remains between 2 and 8 times lower. In this case, the memory consumption of distributed EVALUATOR is slightly greater w.r.t. UPPDMC; this is due to the simple data structures used for storing backward dependencies (linked lists) and could be reduced by using more compact data structures (e.g., packet lists).

We further experimented the scalability of distributed EVALUATOR by considering LTSs of increasing size and more complex properties taken from the CADP demos[2]. For instance,

[2] `http://www.inrialpes.fr/vasy/cadp/demos.html`



(a)



(b)

Figure 4: Speedup (a) and memory overhead (b) of distributed w.r.t. sequential EVALUATOR when checking absence of deadlock

Example	absence of deadlock					presence of livelock				
	truth	U (s)	U (MB)	E (s)	E (MB)	truth	U (s)	U (MB)	E (s)	E (MB)
vasy_2581_11442	false	44	461	2	272	false	47	n.c.	7	844
vasy_4220_13944	false	56	726	21	294	false	67	n.c.	622	1 149
vasy_4338_15666	false	64	745	2	313	false	64	n.c.	11	1 203
vasy_6020_19353	true	59	1 085	24	1 239	true	125	n.c.	8	1 442
vasy_6120_11031	false	95	947	1	170	false	108	n.c.	13	1 092
cwi_7838_59101	true	149	1 531	46	2 298	true	314	n.c.	16	2 793
vasy_8082_42933	false	162	1 374	2	268	false	134	n.c.	24	2 401

Table 1: Execution time (in seconds) and memory consumption (in MB) of two distributed on-the-fly model checkers: UppDmc (U) with 25 bi-Pentium III nodes and Evaluator (E) with 21 Xeon nodes

we used the following formula, stating that after a *put* action, either a livelock, or a *get* action will be eventually reached:

$$\nu X.\,([put]\,\mu Y.\,((\nu Z.\,\langle\tau\rangle\,Z \vee (\langle -\rangle\,\mathsf{true} \wedge [\overline{get}]\,\mathsf{false})) \vee (\langle -\rangle\,\mathsf{true} \wedge [-]\,Y)) \wedge [-]\,X)$$

We checked that this property is satisfied by an Lts named *b*256 with 6 067 712 states and 19 505 146 transitions, modelling the behaviour of a communication protocol that exchanges 256 different messages. The sequential version of Evaluator (based on the Dfs resolution algorithm of Cæsar_Solve) took around 15 minutes to perform the check, whereas the distributed version running on 10 nodes converged in 90 seconds, thus achieving a speedup close to 10.

4.2 Performance of distributed conformance test case generation

We experimented the generation of conformance test cases by using a generic test purpose, which states that an accepting state must be reachable after 10 visible actions. Table 2 shows the performance (in time and memory) of Tgv and sequential Extractor for generating Ctgs from this test purpose and five Ltss from the Vlts benchmark, together with the Lts *b*256 previously used for model checking. The table also gives the size of the resulting Ctgs; note that the raw Ctgs generated by Extractor contain τ-transitions, which explains their difference in size w.r.t. the Ctgs produced by Tgv.

	Tgv				(sequential) Extractor					
Example	time	MB	states	trans.	time	%	MB	%	states	trans.
vasy_164_1619	15'8s	242	100 319	231 266	3'47s	75	210	13	438 861	2 982 696
vasy_166_651	20'23s	242	170 657	586 602	1'41s	92	113	53	444 542	1 504 985
cwi_371_641	6'5s	1600	125 894	597 445	5'20s	12	310	81	1 912 260	3 163 177
vasy_386_1171	9s	11	3 319	3 892	7s	22	10	9	5 561	6 324
vasy_1112_5290	23s	33	10 827	20 888	13s	44	28	15	15 008	41 225
b256	597'4s	2322	264 194	854 786	139'22s	77	2772	-2	12 139 232	39 020 231

Table 2: Performance analysis of Tgv and sequential Extractor on six Ltss with a generic test purpose

Table 3 gives time and memory measures obtained with distributed EXTRACTOR on the same six LTSs. The raw CTGs generated by this tool are exactly the same as its sequential counterpart, since both tools share the same front-end encoding the problem of test case generation. The raw CTGs are subsequently suspended and determinized using DETERMINATOR [HJ03], yielding final CTGs strongly equivalent to those produced by TGV (this was checked using the BISIMULATOR [Mat03, BDJM05] tool). The table also gives time and memory measures, as well as final CTG sizes obtained by applying DETERMINATOR.

	(distributed) EXTRACTOR		DETERMINATOR			
EXAMPLE	time	MB	time	MB	states (final)	transitions (final)
vasy_164_1619	4'39s	470	4'40s	55	103 658	975 594
vasy_166_651	2'59s	335	2'27s	50	173 259	801 675
cwi_371_641	12'4s	880	25'8s	185	127 218	777 278
vasy_386_1171	16s	104	15s	6	2 452	3 894
vasy_1112_5290	27s	228	17s	7	8 369	41 225
b256	180'	6127	19'	459	527 875	1 709 058

Table 3: Performance analysis of (distributed) EXTRACTOR (7 nodes) and final CTG construction by DETERMINATOR

From the measures shown in these two tables, we obtain a speedup of 1.82 of sequential EXTRACTOR combined with DETERMINATOR w.r.t. TGV. However, TGV compares favourably as regards the size of the final CTGs, which is between 30% and 50% smaller. This limitation, although not very significant (the CTGs produced by EXTRACTOR and DETERMINATOR can be subsequently reduced modulo strong equivalence), can be partially overcome by adding on-the-fly reductions, such as compression of τ-cycles [Mat05b], during the computation of the synchronous product between the LTS and the test purpose.

Distributed EXTRACTOR exhibits significant speedups w.r.t. its sequential version as regards the resolution of the underlying BES, and also a good scalability: the resolution on 16 machines of the BES corresponding to the generic test purpose and the LTS *b*256 was done in less than 372 seconds, whereas sequential EXTRACTOR took around 30 minutes (five times slower). This example was handled by TGV in more than 597 minutes, which is four times slower than sequential EXTRACTOR combined with DETERMINATOR.

EXAMPLE	M states	M trans.	EXTRACTOR + DETERMINATOR
cwi_214_684	214	684	8 s., 19 MB, no test case
cwi_566_3984	566	3 984	1195 s., 145 MB, (32 states, 49 trans.)

Table 4: Specification examples on which TGV fails due to memory shortage

Finally, some experiments were performed successfully by EXTRACTOR and DETERMINATOR, but not by TGV. Table 4 shows two LTSs from the VLTS benchmark, on which the CTG generation for the generic test purpose using TGV leads to memory shortage (consumption of more than 3 GB of memory). On the contrary, EXTRACTOR concluded in 8 seconds that no test case was present in *cwi*_214_684 and, together with DETERMINATOR, computed the final (very small) CTG contained in *cwi*_566_3984.

5 Conclusion and future work

Building efficient and generic verification components is crucial for facilitating the development of robust explicit-state analysis tools. Our MB-DSOLVE algorithm for distributed on-the-fly resolution of multiple block, alternation-free BESs, goes towards this objective. MB-DSOLVE was designed to be compliant with the interface of the BES resolution library CÆSAR_SOLVE [Mat03, Mat05a], thus being directly available as verification back-end for all analysis tools based on CÆSAR_SOLVE. Here we illustrated its application for alternation-free μ-calculus model checking and conformance test generation, as distributed computing engine for the tools EVALUATOR [MS03, Mat03] and EXTRACTOR, developed within CADP [GLM02] using the generic OPEN/CÆSAR environment [Gar98] for LTS exploration.

The modular architecture of these tools does not penalize their performance. Our experiments using large state spaces from the VLTS benchmarks have shown that distributed EVALUATOR compares favourably in terms of speed and memory with UPPDMC, the other existing distributed on-the-fly model checker for μ-calculus based on game graphs [HLL04]. Moreover, distributed EVALUATOR exhibits a good speedup and scalability w.r.t. its sequential version, relying on the optimized algorithms of CÆSAR_SOLVE for disjunctive/conjunctive BESs. Distributed EXTRACTOR, to our knowledge the first tool performing distributed on-the-fly conformance test generation, allows to scale up the capabilities of well-established dedicated tools, such as TGV [JJ05].

We plan to continue our work along several directions. First, we will study other distributed resolution strategies, aiming at reducing memory consumption for disjunctive/conjunctive BESs, which occur frequently in practice [Mat03]: one such strategy could combine a distributed breadth-first and a local depth-first exploration of the boolean graph. Next, we will seek distributed solutions to other problems, such as discrete controller synthesis and Horn clause resolution, by investigating their translation in terms of BESs resolution with diagnostic, following, e.g., the approaches proposed in [ZS05] and [LS98].

References

[And94] Henrik R. Andersen. Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[AV95] Henrik R. Andersen and Bart Vergauwen. Efficient Checking of Behavioural Relations and Modal Assertions using Fixed-Point Inversion. In Pierre Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification CAV'95 (Liege, Belgium)*, volume 939 of *Lecture Notes in Computer Science*, pages 142–154. Springer Verlag, July 1995.

[BBS01] Jiri Barnat, Lubos Brim, and Jitka Stribrna. Distributed LTL Model-Checking in SPIN. In Matthew B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software SPIN'2001 (Toronto, Canada)*, volume 2057 of *Lecture Notes in Computer Science*, pages 200–216. Springer Verlag, May 2001.

[BDJM05] Damien Bergamini, Nicolas Descoubes, Christophe Joubert, and Radu Mateescu. BISIMULATOR: A Modular Tool for On-the-Fly Equivalence Checking. In Nicolas Halbwachs and Lenore Zuck, editors, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2005 (Edinburgh, Scotland, UK)*, volume 3440 of *Lecture Notes in Computer Science*, pages 581–585. Springer Verlag, April 2005.

[BLW02] Benedikt Bollig, Martin Leucker, and Michael Weber. Local Parallel Model Checking for the Alternation Free Mu-Calculus. In Dragan Bosnacki and Stefan Leue, editors, *Proceedings of the 9th International SPIN Workshop on Model Checking of Software SPIN'2002 (Grenoble, France)*, volume 2318 of *Lecture Notes in Computer Science*, pages 128–147. Springer Verlag, April 2002.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[DSC99] Xiaoqun Du, Scott A. Smolka, and Rance Cleaveland. Local Model Checking and Protocol Analysis. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(3):219–241, 1999.

[EL86] E. A. Emerson and C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of the 1st LICS*, pages 267–278, 1986.

[FL79] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, (18):194–211, 1979.

[Gar98] Hubert Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84, Berlin, March 1998. Springer Verlag. Full version available as INRIA Research Report RR-3352.

[GLM02] Hubert Garavel, Frédéric Lang, and Radu Mateescu. Compiler Construction using LOTOS NT. In Nigel Horspool, editor, *Proceedings of the 11th International Conference on Compiler Construction CC 2002 (Grenoble, France)*, volume 2304 of *Lecture Notes in Computer Science*, pages 9–13. Springer Verlag, April 2002.

[HJ03] Holger Hermanns and Christophe Joubert. A Set of Performance and Dependability Analysis Components for CADP. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, volume 2619 of *Lecture Notes in Computer Science*, pages 425–430. Springer Verlag, April 2003.

[HK91] S. T. Huang and P. W. Kao. Detecting Termination of Distributed Computations by External Agents. *Journal of Information Science and Engineering*, 7(2):187–201, 1991.

[HLL04] Fredrik Holmen, Martin Leucker, and Marcus Lindstrom. UppDMC – A Distributed Model Checker for Fragments of the Mu-Calculus. In Lubos Brim and Martin Leucker, editors, *Proceedings of the 3rd International Workshop on Parallel and Distributed Methods in Verification PDMC'2004 (London, UK)*, volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 91–105. Elsevier, 2004.

[Hol03] Gerard Holzmann. *The SPIN Model Checker – Primer and Reference Manual*. Addison-Wesley, 2003.

[JJ05] Claude Jard and Thierry Jéron. TGV: Theory, Principles and Algorithms — A Tool for the Automatic Synthesis of Conformance Test Cases for Non-Deterministic Reactive Systems. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 7(4):297–315, August 2005.

[JM04] Christophe Joubert and Radu Mateescu. Distributed On-the-Fly Equivalence Checking. In Lubos Brim and Martin Leucker, editors, *Proceedings of the 3rd International Workshop on Parallel and Distributed Methods in Verification PDMC'2004 (London, UK)*, volume 128 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2004.

[JM05] Christophe Joubert and Radu Mateescu. Distributed Local Resolution of Boolean Equation Systems. In Francisco Tirado and Manuel Prieto, editors, *Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing PDP'2005 (Lugano, Switzerland)*, pages 264–271. IEEE Computer Society, February 2005.

[Koz83] D. Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lar88] K. G. Larsen. Proof Systems for Hennessy-Milner logic with Recursion. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming CAAP '88 (Nancy, France)*, volume 299 of *Lecture Notes in Computer Science*, pages 215–230, Berlin, March 1988. Springer Verlag.

[LS98] X. Liu and S. A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming ICALP'98 (Aalborg, Denmark)*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer Verlag, July 1998.

[LS99] F. Lerda and R. Sisto. Distributed-Memory Model Checking with SPIN. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Proceedings of the 5th and*

6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking SPIN'99, volume 1680 of *Lecture Notes in Computer Science*, pages 22–39. Springer Verlag, July 1999.

[LSW03] Martin Leucker, Rafal Somla, and Michael Weber. Parallel Model Checking for LTL, CTL* and L^2_μ. In Lubos Brim and Orna Grumberg, editors, *Proceedings of the 2nd International Workshop on Parallel and Distributed Methods in Verification PDMC'2003 (Boulder, Colorado, USA)*, volume 89 of *Electronic Notes in Theoretical Computer Science*, pages 4–16. Elsevier, 2003.

[Mad97] Angelika Mader. *Verification of Modal Properties Using Boolean Equation Systems*. VERSAL 8, Bertz Verlag, Berlin, 1997.

[Mat87] F. Mattern. Algorithms for Distributed Termination Detection. *Distributed Computing*, 2:161–175, 1987.

[Mat00] Radu Mateescu. Efficient Diagnostic Generation for Boolean Equation Systems. In Susanne Graf and Michael Schwartzbach, editors, *Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000 (Berlin, Germany)*, volume 1785 of *Lecture Notes in Computer Science*, pages 251–265. Springer Verlag, March 2000. Full version available as INRIA Research Report RR-3861.

[Mat03] Radu Mateescu. A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, volume 2619 of *Lecture Notes in Computer Science*, pages 81–96. Springer Verlag, April 2003. Full version available as INRIA Research Report RR-4711.

[Mat05a] Radu Mateescu. CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 8(1):37–56, February 2005.

[Mat05b] Radu Mateescu. On-the-fly State Space Reductions for Weak Equivalences. In Tiziana Margaria and Mieke Massink, editors, *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems FMICS'05 (Lisbon, Portugal)*, pages 80–89. ERCIM, ACM Computer Society Press, September 2005.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume I (Specification). Springer Verlag, 1992.

[MS03] Radu Mateescu and Mihaela Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Science of Computer Programming*, 46(3):255–281, March 2003.

[PLM03] Gordon Pace, Frédéric Lang, and Radu Mateescu. Calculating τ-Confluence Compositionally. In Jr Warren A. Hunt and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification CAV'2003 (Boulder, Colorado, USA)*, volume 2725 of *Lecture Notes in Computer Science*, pages 446–459. Springer Verlag, July 2003. Full version available as INRIA Research Report RR-4918.

[SS98] Perdita Stevens and Colin Stirling. Practical Model-Checking using Games. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *Lecture Notes in Computer Science*, pages 85–101, Berlin, March 1998. Springer Verlag.

[Tre96] Jan Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

[ZS05] Roberto Ziller and Klaus Schneider. Combining Supervisor Synthesis and Model Checking. *ACM Transactions on Embedded Computing Systems*, 4(2):331–362, May 2005.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

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