Using Static Analysis To Improve Automatic Test Generation*

Marius Bozga[1], Jean-Claude Fernandez[2], and Lucian Ghirvu[1]

[1] VERIMAG, Centre Equation, 2 avenue de Vignate, F-38610 Gières
Marius.Bozga@imag.fr, Lucian.Ghirvu@imag.fr
[2] LSR/IMAG, BP 82, F-38402 Saint Martin d'Hères Cedex
Jean-Claude.Fernandez@imag.fr
tel: +(33) (0)4 76 82 72 14 fax: +(33) (0)4 76 82 72 87

Abstract. Conformance testing is still the main industrial validation technique for telecommunication protocols. The automatic construction of test cases based on the model approach is hindered by the state explosion problem. Our method reduces its magnitude by reconsidering the test case generation at a higher level and by taking advantage of some static analysis techniques, in particular the slicing techniques. The specification is simplified by pipelining a set of three modules, each one implementing a different slicing technique.

Keywords: conformance testing, asynchronous systems, static analysis, slicing, bisimulation

1 Introduction

Conformance testing is a well-established technique for the validation of telecommunication protocols. Currently, it is still the main validation technique used at an industrial scale, given the inherent complexity of more ambitious techniques such as formal verification. Moreover, in the case of protocols, the conformance testing was completely formalized by [22, 7, 15] and is also standardized within [12]. Test cases can be automatically generated from formal specifications and tools such as TGV [11], TVEDA [18], AUTOLINK [21] or TORX [2] concretely implement this activity.

In the model-based approach, test cases are usually constructed by exploring a synchronous product between the model of the specification and some test purpose, both represented as labeled transition systems. The central problem arising here is the well known state explosion problem. To deal with it we propose to reconsider the test generation at a higher level i.e., to work with specifications and test purposes represented by some kind of extended automata and to perform relevant static simplifications before generating test cases. In this paper, we consider specifications as asynchronously communicating extended automata

* This work was supported in part by the European Commission FET project ADVANCE, contract No IST-1999-29082.

and test purposes as acyclic automata with constraints. We want to generate tests describing a finite interaction between the tester and the implementation under test. Moreover, by the fact that the test purpose is an automaton with constraints, it is possible to generate symbolic tests.

We propose to simplify specifications by means of *slicing* [23]. A first slicing consists in taking into account the set of interactions between specification components, starting from inputs enabled in the test purpose, and regardless the signal parameters. We obtain a first reduction of the specification, consisting of the part of it which is reachable given the enabled inputs. Then, for the second slicing, we look at the set of variables and parameters which are relevant to outputs observed by the test purpose and safely discard the others. Finally, the specification is sliced with respect to constraints attached to the test purpose. These analyses transform specifications without loss of information with respect to the test purpose. They are independent of each other and can be implemented separately. Each of them inputs a specification, performs a slicing on it, and then outputs a new equivalent one. They can be applied in any order, until no more reduction can be obtained. Concerning the overall simplification on the initial specification, our experiments showed very good results.

The idea of using static analysis to improve model checking and test generation was already being investigated in different contexts. For instance, TVEDA [18] produces test cases from SDL specifications by performing simple syntactic transformations on them. Slicing is used in the context of automatic generation of test data for sequential programs. In [13], the authors present an approach to selective regression testing using slicing. Selective regression approach identifies the parts of the program that are affected by a change. In [14], slicing is used for verification purposes, to extract finite-state machines from multi-threaded programs.

The paper is structured as follows. Section 2 briefly remember the notions of conformance testing and presents the underlying model. In section 3 we propose and formalize three new slicing techniques of the specification with respect to the test purpose. Finally, we give some results in section 4 and we conclude in section 5.

2 Conformance test case generation

This section reviews some notions of conformance testing and presents the underlying model, which is parallel asynchronous processes communicating via queues.

Conformance testing is a black-box testing method, which aims at validating that the implementations of protocols conform to their specifications. Conformance testing activity is standardized in [12] and work has been done to formalize it [22]. In this context, our purpose is to generate automatically conformance test cases for telecommunication protocols.

In the classification of testing architectures from [12, 20], our method is a *local single-layer test method* with synchronous communication between the tester and the *implementation under test* (IUT). It is *local* because in the interactions between the tester and the IUT no event caused by the surrounding environment appears. It is *single-layer* because we test implementations of specifications organized in one layer. The tester interacts with the IUT via some *points of control and observation* (PCOs), which, in our case, are seen as external queues of the specification (see below). The communication at the PCOs is synchronous. This architecture is pictured in Figure 1.

In order to assure the feasibility of our method (correctness, compatibility with the industrial practice) we require that the tester, the IUT and the specification satisfy some conditions :

1. *controllability condition* : the tester always controls its outputs and can feed the specification only at one PCO at the time (therefore, for each state of the test purpose, whenever an input is enabled, it is the only transition starting in this state),
2. *consistency relation* : between the test purpose and the specification (which ensures that the set of behaviors described by the test purpose is included in the set of behaviors described by the specification)[1],
3. *conformance relation* : ensures that the outputs of the implementation must be produced also by the specification.

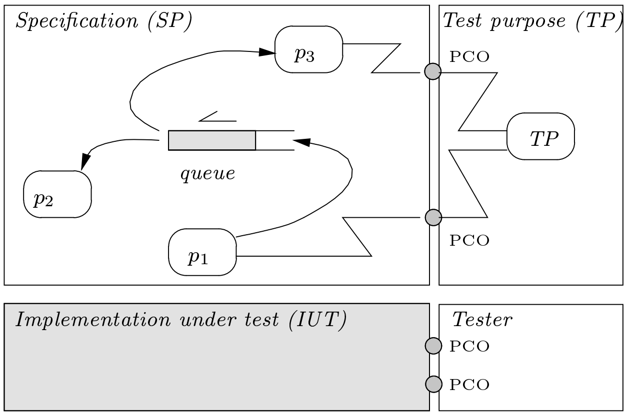

Fig. 1. Test architecture

2.1 The specification

We consider specifications consisting of asynchronous parallel composition of a number of processes that communicate through parameterized signals passing via a set of unbounded fifo queues. We distinguish between internal queues (closed inside the specification) and external queues (opened to the environment). In the context of conformance testing with local tester, external queues contents are controlled by the tester. Then, we make explicit the assumption of synchronicity between the tester and the IUT. Processes are extended finite-state automata. They perform actions on queues and local variables. For the sake of simplicity, the actions are simple guarded commands.

[1] This assumption is strong, however it can be verified during the test generation process (as in TGV [10]).

Definition 1 (specification syntax). *A specification SP is a tuple (S, C, P) where S is the set of signals, $C = C^{int} \cup C^{ext}$ is the set of queues (internal and external ones) and P is the set of processes. A process $p \in P$ is a tuple (X_p, Q_p, T_p, q_p^0) where X_p is a set of local variables, Q_p is a set of states, Σ_p is a set of actions which can be performed by p, and $T_p \subseteq Q_p \times \Sigma_p \times Q_p$ is a set of transitions. An action can be either a guarded assignment $[\ b\ ]\ x := e$, a guarded input $[\ b\ ]\ c?s(x)$, or a guarded output $[\ b\ ]\ c!s(e)$. Above, b and e are expressions, $x \in X_p$ is a variable, $c \in C$ is a queue and $s \in S$ is a signal.*

We give the semantics of specifications in terms of labeled transition systems. We assume the existence of the universal domain D which contains the values of variables and signal parameters. We suppose that the boolean values $\{\mathbf{t}, \mathbf{f}\}$ and also the special undefined $\perp$ value are contained in D. We define variable contexts as being total mappings $\rho : \bigcup_{p \in P} X_p \to D$ which associate to each variable x a value v from the domain. We extend these mappings to expressions in the usual way. We define internal queue contexts as being also total mappings $\delta : C^{int} \to (S \times D)^*$ which associates to each internal queue c a sequence $(s_1, v_1), ..., (s_k, v_k)$ of messages, that is pairs (s, v) noted also by $s(v)$, where s is a signal and v is the carried parameter value. We assume also the existence of some special undefined message σ. The empty sequence is noted with ϵ.

Definition 2 (specification semantics). *The semantics of a specification SP is given by a labeled transition system $\widehat{SP} = (\widehat{Q}_{sp}, \widehat{T}_{sp}, \hat{q}_{sp}^0)$. States $\hat{q}_{sp}$ of this system are triples of the form (ρ, δ, θ), where ρ is a variable context, δ is a queue context and $\theta = \langle q_1, ...q_n \rangle \in \times_{p \in P} Q_p$ is a global control state. Transitions are either internal and labeled with τ, when derived from assignments or internal communication, either visible and labeled with the concrete action when derived from external communication. Transitions are constructed by the following rules:*

$$\frac{q_p \xrightarrow{[b]\ x:=e} q_p' \quad \rho(b) = \mathbf{t} \quad \rho(e) = v}{(\rho, \delta, \theta) \xrightarrow{\tau} (\rho[v/x], \delta, \theta')} \qquad \frac{q_p \xrightarrow{[b]\ c!s(e)} q_p' \quad \rho(b) = \mathbf{t} \quad \rho(e) = v \quad c \in C^{ext}}{(\rho, \delta, \theta) \xrightarrow{c!s(v)} (\rho, \delta, \theta')}$$

$$\frac{q_p \xrightarrow{[b]\ c!s(e)} q_p' \quad \rho(b) = \mathbf{t} \quad \rho(e) = v \quad c \in C^{int} \quad \delta(c) = w}{(\rho, \delta, \theta) \xrightarrow{\tau} (\rho, \delta[w.s(v)/c], \theta')}$$

$$\frac{q_p \xrightarrow{[b]\ c?s(x)} q_p' \quad \rho(b) = \mathbf{t} \quad c \in C^{ext}}{(\rho, \delta, \theta) \xrightarrow{c?s(v)} (\rho[v/x], \delta, \theta')} \qquad \frac{q_p \xrightarrow{[b]\ c?s(x)} q_p' \quad \rho(b) = \mathbf{t} \quad c \in C^{int} \quad \delta(c) = s(v).w}{(\rho, \delta, \theta) \xrightarrow{\tau} (\rho[v/x], \delta[w/c], \theta')}$$

where θ' was obtained from θ by considering one step in process p from q_p to q_p', and the initial state $\hat{q}_{sp}^0$ is obtained considering the default value of the variables, empty queues and processes initial states.

2.2 Test purpose

The test purpose is an acyclic finite state automaton which describe a pattern of interactions between the tester and the IUT. It is described from the implementation side i.e., inputs and outputs in the test purpose means respectively inputs and outputs in the implementation. It contains both constrained signal inputs and unconstrained signal outputs.

A *constraint* $\mathcal{C}$ is a boolean combination of atoms, each of them being a particular restriction on the used value. For example, we can test the containment of an element to an interval or to a set of values. The notation $v \models \mathcal{C}$ stands for the value v satisfies the constraint $\mathcal{C}$. For a given input of the test purpose, there is a constraint related to the signal parameter [12]. There are no relational dependencies between constraints attached to different inputs.

This test purpose definition was inspired by TTCN and has the following intuition : if the tester provide a signal to the implementation with the value of its parameter satisfying a constraint then we would like to approximate the value of outputs parameters.

Definition 3 (test purpose). *A test purpose TP is a tuple $(Q_{tp}, T_{tp}, q^0_{tp}, Q^{acc}_{tp})$ where Q_{tp} is a set of states, $T_{tp} \subseteq Q_{tp} \times \Sigma_{tp} \times Q_{tp}$ is a set of transitions and $Q^{acc}_{tp} \subseteq Q_{tp}$ is a set of accepting states, without successors by T_{tp}. Σ_{tp} is the set of interactions α_{tp} which can be fixed within the test purpose. This set contains both constrained signal inputs of the form $c?s(\mathcal{C})$ and unconstrained signal outputs of the form $c!s(*)$, where $c \in C^{ext}$ is an external queue and $s \in S$ is a signal. $\mathcal{C}$ denotes a generic constraint e.g., interval constraint, on the received value and $*$ denotes any value.*

The feeds are the set of inputs we intend to supply to the IUT during the test. Feeds are a parameter completely controlled by the tester. They will be taken into account during the test case generation process. That is, any external input in the specification will be enabled if and only if it is contained in the set of feeds. Intuitively, the set of feeds must cover the set of inputs given in the test purpose.

Definition 4 (feeds). *The feeds Σ_f are a set of constrained signal inputs $\{c?s(\mathcal{C}) \mid c \in C^{ext}, s \in S\}$.*

2.3 Synchronous product

The tests are automatically derived by exploring a kind of synchronous product between the model of the specification $(\widehat{Q}_{sp}, \widehat{T}_{sp}, \hat{q}^0_{sp})$, the test purpose $(Q_{tp}, T_{tp}, q^0_{tp}, Q^{acc}_{tp})$, and taking into account the set of feeds $\Sigma_f = \{c?s(\mathcal{C}) \mid c \in C^{ext}, s \in S\}$.

Definition 5 (synchronous product). *We define the synchronous product $\prod(\widehat{SP}, TP, \Sigma_f)$ as the labeled transition system (Q_π, T_π, q^0_π), with $Q_\pi \subseteq \widehat{Q}_{sp} \times Q_{tp}$, where Q_π and T_π are the smallest sets obtained by the application of the following rules:*

$$\frac{-}{(\hat{q}^0_{sp}, q^0_{tp}) \in Q_\pi} \qquad \frac{(\hat{q}_{sp}, q_{tp}) \in Q_\pi \quad \hat{q}_{sp} \xrightarrow{\tau} \hat{q}'_{sp} \quad q_{tp} \notin Q^{acc}_{tp}}{(\hat{q}'_{sp}, q_{tp}) \in Q_\pi \quad (\hat{q}_{sp}, q_{tp}) \xrightarrow{\tau} (\hat{q}'_{sp}, q_{tp})}$$

$$\frac{(\hat{q}_{sp}, q_{tp}) \in Q_\pi \;\; \hat{q}_{sp} \xrightarrow{c?s(v)} \hat{q}'_{sp} \;\; q_{tp} \xrightarrow{c?s(\mathcal{C})} q'_{tp} \;\; v \models \mathcal{C}}{(\hat{q}'_{sp}, q'_{tp}) \in Q_\pi \;\; (\hat{q}_{sp}, q_{tp}) \xrightarrow{c?s(v)} (\hat{q}'_{sp}, q'_{tp})}$$

$$\frac{(\hat{q}_{sp}, q_{tp}) \in Q_\pi \;\; \hat{q}_{sp} \xrightarrow{c?s(v)} \hat{q}'_{sp} \;\; c?s(\mathcal{C}) \in \Sigma_f \;\; v \models \mathcal{C}}{(\hat{q}'_{sp}, q_{tp}) \in Q_\pi \;\; (\hat{q}_{sp}, q_{tp}) \xrightarrow{c?s(v)} (\hat{q}'_{sp}, q_{tp})}$$

$$\frac{(\hat{q}_{sp}, q_{tp}) \in Q_\pi \;\; \hat{q}_{sp} \xrightarrow{c!s(v)} \hat{q}'_{sp} \;\; q_{tp} \xrightarrow{c!s(*)} q'_{tp}}{(\hat{q}'_{sp}, q'_{tp}) \in Q_\pi \;\; (\hat{q}_{sp}, q_{tp}) \xrightarrow{c!s(v)} (\hat{q}'_{sp}, q'_{tp})} \qquad \frac{(\hat{q}_{sp}, q_{tp}) \in Q_\pi \;\; \hat{q}_{sp} \xrightarrow{c!s(v)} \hat{q}'_{sp}}{(\hat{q}'_{sp}, q_{tp}) \in Q_\pi \;\; (\hat{q}_{sp}, q_{tp}) \xrightarrow{c!s(v)} (\hat{q}'_{sp}, q_{tp})}$$

Example 1. The previous definitions are exemplified in Figure 2. The specification is composed of two processes which communicate through an internal queue $cl \in C^{int}$. The external queues of the specification are $ci, co \in C^{ext}$. The set of feeds are $\Sigma_f = \{ci?sr([1, 10])\}$. This example will be used throughout the paper in order to picture the changes of the specification induced by the following slicing algorithms.

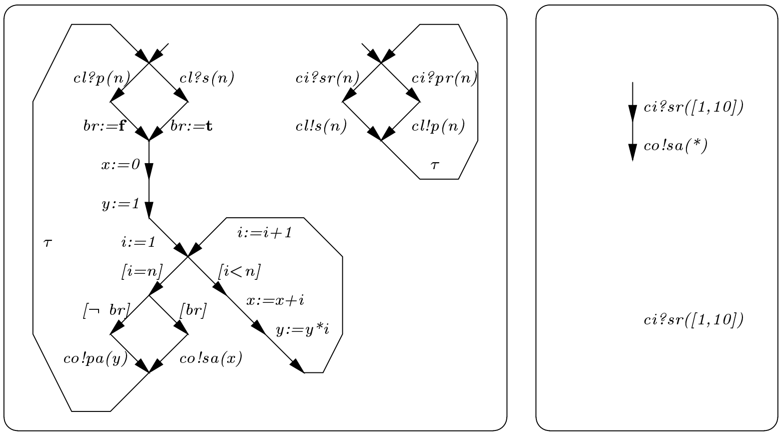

Fig. 2. Example

3 Static analysis for testing

The purpose of static analysis is to compute, given the test purpose and the feeds, the part of the specification which is relevant to them. We present three kinds of analyses :

1. *the relevant control analysis*: restricts the processes contained in the specification, to the sets of states and transitions which might be statically reached, given the set of feeds,
2. *the relevant variables analysis*: computes and simplifies processes, with respect to variables which might be used to compute values needed for outputs mentioned in the test purpose and

3. *the constraint propagation*: aims to further simplify processes, given the concrete constraints attached to feeds.

Each one of the analysis takes as input a specification and provides an equivalent one with respect to the test generation method presented below. They are completely independent and can be applied in any order. Furthermore, they could be applied iteratively as the code optimization techniques ([17]). The reduction obtained by one can be further exploited by another and so on, until no more reductions are possible.

We detail each one of these analysis below and illustrate them on the example presented before.

3.1 Relevant control analysis

A conservative approximation for the specification is computed, by taking into account the set of feeds. We restrict each process to the set of states and transitions that might be reached given the feeds. Intuitively, this analysis can be seen as building the largest sub-processes, after the removal of external inputs uncovered by feeds, and subsequently the internal inputs uncovered by internal outputs.

Definition 6 (slicing wrt feeds). *We define the slice of a specification $SP = (S, C, P)$ with respect to a set of feeds Σ_f to be the specification $SP \setminus_f \Sigma_f = (S, C, P \setminus_f \Sigma_f)$, where $P \setminus_f \Sigma_f$ contains a sliced process p' for each process $p \in P$. The slice for a process $p = (X_p, Q_p, T_p, q_p^0) \in P$ is defined as the process $p' = (X_p, Q'_p, T'_p, q_p^0)$, with the same sets of variables. The sets of states $Q'_p \subseteq Q_p$ and transitions $T'_p \subseteq T_p$ are defined as the smallest ones which satisfy the following rules:*

$$\frac{-}{q_p^0 \in Q'_p} \qquad \frac{q_p \in Q'_p \;\; q_p \xrightarrow{[b]x:=e} q'_p}{q'_p \in Q'_p \;\; q_p \xrightarrow{[b]x:=e} q'_p \in T'_p} \qquad \frac{q_p \in Q'_p \;\; q_p \xrightarrow{[b]c!s(e)} q'_p}{q'_p \in Q'_p \;\; q_p \xrightarrow{[b]c!s(e)} q'_p \in T'_p}$$

$$\frac{q_p \in Q'_p \;\; q_p \xrightarrow{[b]c?s(x)} q'_p \;\; c \in C^{ext} \;\; c?s(\mathcal{C}) \in \Sigma_f}{q'_p \in Q'_p \;\; q_p \xrightarrow{[b]c?s(x)} q'_p \in T'_p}$$

$$\frac{q_p \in Q'_p \;\; q_p \xrightarrow{[b]c?s(x)} q'_p \;\; c \in C^{int} \;\; \exists r.q_r \xrightarrow{[b']c!s(e)} q'_r \in T'_r}{q'_p \in Q'_p \;\; q_p \xrightarrow{[b]c?s(x)} q'_p \in T'_p}$$

We must notice here the input/output propagation between processes. That is, we keep an input inside some process p if and only if there exists some dual output enabled in some other process r.

The algorithm computing the sliced system proceeds in an iterative manner. It maintains the sets of states and transitions reached for each process. Initially, the sets of states contain the initial state of the processes, and the sets of transitions are empty. Then, at each step, one of the rules before is applied until the least fixed point is reached and no more rule is applicable.

This algorithm is similar to reachability analysis but it is performed at the control level.

Example 2. If one applies the previous algorithm for the specification and the feeds from Figure 2, one obtains the specification shown in Figure 3. The external input $ci?pr(n)$ is uncovered by the feeds so its elimination induces the elimination of $cl!p(n)$ and thus $cl?p(n)$ is no more covered by an internal output so it is eliminated together with $br := \mathrm{f}$.

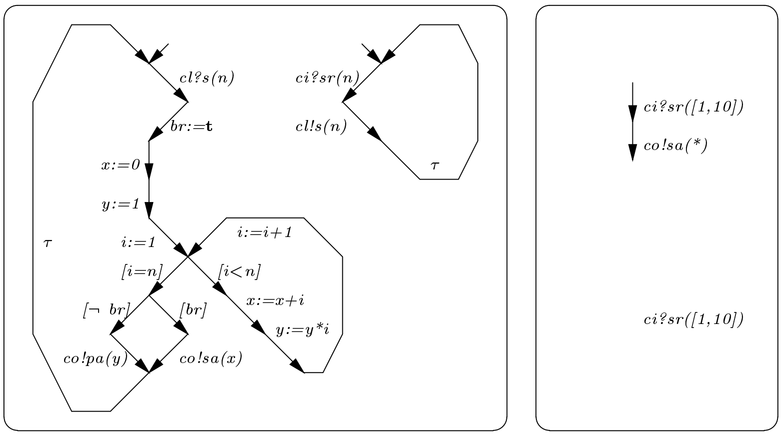

Fig. 3. Example slicing wrt feeds

The slicing with respect to feeds preserves the synchronous product, that is, the following proposition holds.

Proposition 1 (slicing wrt feeds correctness). *Let $SP = (S, C, P)$ be a specification, TP a test purpose and Σ_f a set of feeds which covers TP. The synchronous products between the models of SP and respectively $SP \setminus_f \Sigma_f$ with the test purpose TP, given the feeds Σ_f are strongly bisimilar. That is, $\prod(\widehat{SP}, TP, \Sigma_f) \sim \prod(\widehat{SP \setminus_f \Sigma_f}, TP, \Sigma_f)$.*

3.2 Relevant variables analysis

This calculus is an extension of live variable analysis [4]. It attempts to compute, for each process, the set of relevant variables in each state. The relevance is defined with respect to test purpose outputs: a variable will be relevant in a state if its value at that state might be used to compute the parameter value of some signal output occurring in the test purpose. Or, similar to the live variables definition, we consider a variable to be relevant in a state if and only if there exists a path starting at that state such that the variable is used before being redefined on the path. But, in our case, we consider a variable to be used only in external outputs mentioned by the test purpose, or in assignments (eventually via internal inputs) to relevant variables.

Definition 7 (relevant variables wrt outputs). *Let $SP = (S, C, P)$ be a specification and $TP = (Q_{tp}, T_{tp}, q^0_{tp}, Q^{acc}_{tp})$ be a test purpose. Let Σ_o be the set of signal outputs mentioned in the test purpose. The relevant variables are defined for each process $p = (X_p, Q_p, T_p, q^0_p) \in P$ by a function $Rlv_p : Q_p \to 2^{X_p}$ mapping states to subsets of variables. The sets $Rlv_p(q_p)$ for states $q_p \in Q_p$ are defined as the least fixed point of the following equation system:*

$$Rlv_p(q_p) = \bigcup_{t_p : q_p \xrightarrow{\alpha} q'_p} Rlv(q'_p) \setminus Def_p(t_p) \cup Use_p(t_p)$$

where

$$Def_p(t_p) = \begin{cases} \{x\} & \\ \quad \text{if } t_p = q_p \xrightarrow{[b]x:=e} q'_p & \\ \quad \text{or } t_p = q_p \xrightarrow{[b]c?s(x)} q'_p & \\ \emptyset \text{ otherwise} & \end{cases}$$

$$Use_p(t_p) = \begin{cases} vars(b) \cup vars(e) \\ \quad \text{if } t_p = q_p \xrightarrow{[b]x:=e} q'_p \text{ and } x \in Rlv_p(q'_p) \\ \quad \text{or } t_p = q_p \xrightarrow{[b]c!s(e)} q'_p \text{ and} \\ \qquad c \in C^{ext} \text{ and } \exists q_{tp} \xrightarrow{c!s(*)} q'_{tp} \in T_{tp} \text{ or} \\ \qquad c \in C^{int} \text{ and } \exists r.q_r \xrightarrow{c?s(x)} q'_r \in T_r \text{ and} \\ \qquad\quad x \in Rlv_r(q'_r) \\ vars(b) \text{ otherwise} \end{cases}$$

The relevant variables are computed simultaneously for all processes. The algorithm operates in a backward manner on the control graphs. It starts with empty sets of variables for each state, and at each step one transition is analyzed: the set of used variables is recomputed in the current context and then, the relevant variables set for the source state is updated. The algorithms ends when the least fixed point is reached and no more change in the relevance sets occurs for any of the transitions.

For this analysis too, we notice that the relevance of variables is propagated interprocesses. In fact, variables used in expressions sent through internal channels will become relevant only if, at the destination side their value is further relevant.

The slicing with respect to relevant variables attempts to reduce the number of variables used inside processes. Concretely, we cut off all the definitions assigning irrelevant variables. Irrelevant variables used in inputs are replaced by some special, don't care, variable $\top$. Finally, expressions occurring in unused outputs are replaced by the undefined value $\perp$. This transformation is formally described below.

Definition 8 (slicing wrt outputs). *Let $SP = (S, C, P)$ be a specification and TP be a test purpose. We define the slice of the specification SP given the relevant variables computed wrt outputs to be the specification $SP \setminus_o \Sigma_o = (S, C, P \setminus_o \Sigma_o)$, where $P \setminus_o \Sigma_o$ contains a sliced process p' for each process $p \in P$. The slice for a process $p = (X_p, Q_p, T_p, q^0_p)$ is defined as a process $p' = (X'_p, Q_p, T'_p, q^0_p)$ which has the same set of states and the same initial state,* *but operates only on relevant variables. We put* $X'_p = \bigcup_{q_p \in Q_p} Rlv_p(q_p)$ *and transitions* T'_p *are constructed from* T_p *such that they do not more define irrelevant variables :*

$$\frac{q_p \xrightarrow{[b]x:=e} q'_p \quad x \in Rlv_p(q'_p)}{q_p \xrightarrow{[b]x:=e} q'_p \in T'_p} \qquad \frac{q_p \xrightarrow{[b]x:=e} q'_p \quad x \notin Rlv_p(q'_p)}{q_p \xrightarrow{[b]\tau} q'_p \in T'_p}$$

$$\frac{q_p \xrightarrow{[b]c?s(x)} q'_p \quad x \in Rlv_p(q'_p)}{q_p \xrightarrow{[b]c?s(x)} q'_p \in T'_p} \qquad \frac{q_p \xrightarrow{[b]c?s(x)} q'_p \quad x \notin Rlv_p(q'_p)}{q_p \xrightarrow{[b]c?s(\top)} q'_p \in T'_p}$$

$$\frac{q_p \xrightarrow{[b]c!s(e)} q'_p \quad Use(c!s)}{q_p \xrightarrow{[b]c!s(e)} q'_p \in T'_p} \qquad \frac{q_p \xrightarrow{[b]c!s(e)} q'_p \quad \neg Use(c!s)}{q_p \xrightarrow{[b]c!s(\bot)} q'_p \in T'_p}$$

where $Use(c!s) = \exists q_{tp} \xrightarrow{c!s(*)} q'_{tp} \in T_{tp}$ *or* $\exists r.q_r \xrightarrow{c?s(x)} q'_r \in T_r$ *and* $x \in Rlv_r(q'_r)$ *denote the global utility of outputs of the form* $c!s$*.*

Example 3. The *slicing wrt outputs* algorithm, applied for the specification and the test purpose from Figure 3, produces the specification shown in Figure 4. The transitions labeled $y := 1$ and $y := y * i$ are relabeled with τ and the output $co!pa(y)$ become $co!pa(\bot)$ because $\neg Use(co!pa)$.

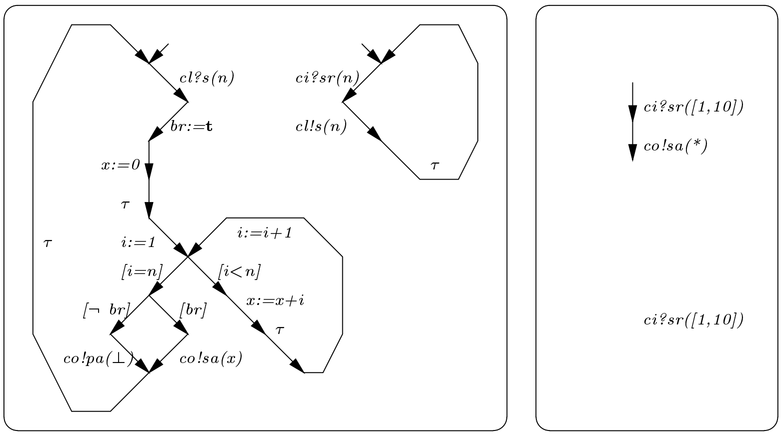

Fig. 4. Example slicing wrt outputs

Intuitively, the slicing wrt relevant outputs preserves the model of the specification up to concrete values carried by signals not observed in the test purpose. We define the renaming of the specification model $\widehat{SP}$ with respect to the set of output actions Σ_o in the following way: each visible output action $c!s(v)$ which is not specified by the test purpose i.e., $c!s(*) \notin \Sigma_o$, is renamed into $c!s(\bot)$. The other actions are left unchanged. In this way, we left out the exact parameter values for outputs, other than ones occurring in the test purpose. We note the renamed model with $\widehat{SP} \downarrow \Sigma_o$. The following proposition holds.

Proposition 2 (slicing wrt outputs correctness). *Let $SP = (S, C, P)$ be a specification and $TP = (Q_{tp}, T_{tp}, q_{tp}^0, Q_{tp}^{acc})$. The model of SP renamed with respect to the observable outputs Σ_o and respectively the model of $SP \setminus_o \Sigma_o$ are strongly bisimilar, that is, $\widehat{SP} \downarrow \Sigma_o \sim \widehat{SP \setminus_o \Sigma_o}$.*

A final remark concerns a more general utility of relevant variables. In fact, we tried here to exploit them at a purely syntactic level e.g., by eliminating the irrelevant ones and their dependencies in the specification. However, it is possible to take them into account in a deeper manner. For instance, using a technique similar to [4], one can reinitialize them with a default value as soon as they become irrelevant, thereby achieving a bisimilar reduced model.

3.3 Constraint propagation

This section provides an approach to simplify the specification, using the constraints imposed on the feeds and the inputs of the test purpose. First, these constraints will be attached to possible matching inputs. Then, by using some intra/interprocesses data flow analysis algorithms, the constraints are propagated in the specification. Thus, for each control state, a conservative approximation of the set of possible values for each variable is computed. Finally, this information is used to evaluate the transitions guards and to eliminate those ones never firable.

In the following we will sketch the *constraint propagation* problem and how to solve it. It is a data flow analysis problem whose basic components are :

1. the flow graph is composed of the states and the transitions of each process and some auxiliary constructions in order to simulate the internal queues,
2. the complete powerset lattice of D, the constraints being some of its elements,
3. the class of *transfer* functions $Transfer_{t_p}$, for each transition t_p.
4. the confluence functions $\bigsqcup$, one for each state.

Let us observe that by choosing the constraints to be the elements of 2^D we have ensured the possibility of testing the emptiness of a constraint and also the possibility of having a partial order among them.

In order to define the transfer functions for transitions, one has to ensure that the actions of transitions (assignments and arithmetic operations) can be realized with constraints (that is, with set of values of D instead of only one value of D). This requirement is fulfilled by defining the operations with set of values similarly as in the *interval arithmetic* [19].

Having seen what are the basic requirements and an approach to fulfill them, the definition of constraint propagation problem follows below.

Definition 9 (constraint propagation). *Let $SP = (S, C, P)$ be the specification and Σ_f the set of feeds. Constraints are represented, for each process, as a function $Val : Q_p \rightarrow 2^D$. With the notations presented before, the constraint propagation problem is defined as finding the least fix point solution of the following equation system:*

$$Val(q_p') = \bigsqcup_{t_p : q_p \xrightarrow{\alpha} q_p'} Transfer_{t_p}(Val(q_p))$$

In order to solve this problem we have considered the cases where constraints are expressed by means of constants and by integer intervals. This is due to the fact that in TTCN [12] the constraints have this kind of simple forms. The formal framework defined above is applicable in these cases, using the Galois connection, a classical abstract interpretation technique [8].

The algorithm used for solving the constraint propagation problem in the case of the lattice of constants is the classical iterative algorithm from [16] with an interprocesses variant such as [9]. In the case of the integer intervals lattice, due to the fact that it has infinite height, we use for each process a widening technique as in [3].

The results of the constraint propagation problem are used in simplifying the specification by means of slicing. However, they also allow, for the outgoing output transitions of a control state, to have a conservative approximation of the parameters of the signals, thereby enabling generation of symbolic test cases.

Definition 10 (slicing wrt constraints). *Let $SP = (S, C, P)$ be a specification and Σ_f a set of feeds. We define the slice of the specification SP given the constraints computed wrt feeds to be the specification $SP \setminus_c \Sigma_f = (S, C, P \setminus_c \Sigma_f)$, where $P \setminus_c \Sigma_f$ contains a sliced process p' for each process $p \in P$. The slice for a process $p = (X_p, Q_p, T_p, q_p^0)$ is defined as a process $p' = (X_p, Q_p', T_p', q_p^0)$, which operates on the same set of variables X_p. The sets of states $Q_p' \subseteq Q_p$ and transitions $T_p' \subseteq T_p$ are the smallest ones which satisfy the following rules:*

$$\frac{-}{q_p^0 \in Q_p'} \qquad\qquad \frac{q_p \in Q_p' \;\; t_p = q_p \xrightarrow{\alpha} q_p' \;\; Transfer_{t_p}(Val_p(q_p)) \neq \emptyset}{q_p' \in Q_p' \;\; t_p = q_p \xrightarrow{\alpha} q_p' \in T_p'}$$

Example 4. The *slicing wrt constraints* algorithm, applied for the specification and the test purpose from Figure 4, produces the specification shown in Figure 5. The value $\mathbf{t}$ for br is propagated to the source state of the transition with the guard $[\neg br]$ and thus determining that this transition and the following $co!pa(\bot)$ will be never fired. These transitions are detached from the specification. The constraint propagation problem, in this case, given the feed $ci?sr([1, 10])$, provides for the parameter x in the transition $co!sa(x)$ the interval $[1, 100]$.

We have the following preservation result.

Proposition 3 (slicing wrt constraints correctness). *Let $SP = (S, C, P)$ be a specification, $TP = (Q_{tp}, T_{tp}, q_{tp}^0, Q_{tp}^{acc})$ and Σ_f a set of feeds which cover TP. The synchronous product between the models of SP and respectively $SP \setminus_c \Sigma_f$ with the test purpose given the feeds Σ_f are strongly bisimilar, that is $\prod(\widehat{SP}, TP, \Sigma_f) \sim \prod(\widehat{SP \setminus_c \Sigma_f}, TP, \Sigma_f)$*

4 Experimentation

These techniques were implemented and currently we are experimenting with them on some case studies. We use the IF [5] framework, which is an interme-

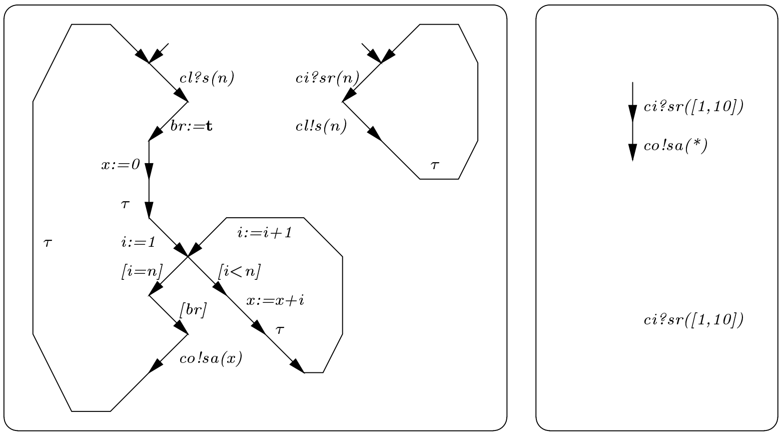

Fig. 5. Example slicing wrt constraints

diate program representation for protocols, based on *asynchronously communicating timed automata*. IF was designed right from the beginning to support the application of static analysis techniques used in compiler optimization [1, 17].

The techniques presented before were applied to improve test case generation for the SSCOP protocol [6], a layer from the ATM protocols stack. Previous work was already done on it in the context of the FORMA research action [6] and despite its success, it shows also the limitations of the existing test generation technology. We were interested to see what are the concrete benefits of our add-ons.

We started with an SDL version of the protocol provided by CNET France Telecom. It is about 2000 lines of code which describes the whole protocol as a single SDL process. It was translated into an IF extended automaton, with 1075 states, 1291 transitions and 134 variables. We considered 10 test purposes conceived for different phases of the protocol (connection, data transmission). The results obtained using previous analysis are summarized below:

slicing wrt feeds: By carefully choosing the appropriate set of feeds for each test purpose, we obtained reductions up to 80% of the specification. That is, we started usually with the smallest set of feeds covering the test purpose inputs. This choice is often too restrictive i.e., test cases cannot be constructed from the model of sliced specification. Thus, we iteratively added other inputs to the feeds until the model became sufficiently large to cover the test purpose behavior. In this way, we are able to work on some minimal version of the specification, still allowing the generation of test cases.

slicing wrt outputs: This analysis gives very good results too. When applied on the sliced specification wrt to feeds, it reduces the number of variables up to 40%. More generally, for the SSCOP protocol we obtained, in average, that, from total number of variables, 30% are relevant with respect to outputs, while the maximum reaches 60%. Also, when used at the model generation time, the relevant variables still allow important reductions on the number of model states and transitions.

constraint propagation: The constraint propagation is still under development. At this time, we experimented only the constant propagation algorithm. The results obtained are good, mainly when the test purpose and the feeds contains punctual constraints i.e., the values provided in test purpose inputs are fixed at some constants. We work currently to adapt the interval propagation algorithm.

These results are very encouraging. However, given the particular nature of the SSCOP protocol, we need further experimentations to clearly set up our techniques. We will consider experimentations for interprocesses slicing and the interaction between the three slicing techniques.

5 Conclusion and future work

In this paper, we show that automatic test generation can take advantages of static analysis. Our test generation method, derived from the so-called on the fly model checking, consists in traversing a product defined between the specification and the test purpose. Before test generation, simplifications may be made on the specification, by collecting informations on the test purposes.

Our general approach to define static analysis is based on the following considerations and remarks. The specification is a set of extended automata, asynchronously communicating via a set of queues. The test purpose is also an extended automaton with constraints. In the context of test generation, we distinguish between inputs and outputs. The static analysis we define transform specifications into others, smaller ones, without loss of information with respect to test purpose. This approach is compatible with the standard definition of conformance testing.

In this work, we proposed three kinds of slicing, based on different analysis. The first one consists in restricting each automaton, starting from the set of feeds. It includes the propagation through the dependence relation between the input of a process and the outputs of the others process. The second analysis computes the set of variables, necessary to determine values occurring into the outputs, and safely discards the others. The last analysis is the constraint propagation.

We have shown that the combination of these three interprocesses analyses may reduce the specification. We have implemented these analysis in the context of IF tools and we obtain very good results on the SSCOP protocol.

Our results can be further extended in several directions. Firstly, we aim at experimenting more systematically the static analysis in the context of test case generation for industrial protocols. The results on SSCOP were very encouraging but other experimentations are further needed to conclude the practical use of our techniques.

At short term too, we plan to extend these analysis techniques to work on timed specifications. In fact, the generated test cases usually uses timers, which are set and test to more or less arbitrarily values in order to observe deadlocks or livelocks in the implementation. However, a more fine analysis can be done on timed specifications to obtain relevant values to be used in this context.

A more speculative direction concerns the generation of symbolic tests. In this respect, we are currently thinking about the appropriate extension for the test purposes concept. For instance, the explicit use of variables in addition to constraints can make them much more expressive. Furthermore, it may be interesting to reconsider the definition of the synchronous product i.e., to be done at a higher level in such a way that it will allow the derivation of symbolic test cases.

References

1. A. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.
2. A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal Test Automation : a Simple Experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *12^{th} International Workshop on Testing of Communicating Systems*. Kluwer Academic Publishers, 1999.
3. F. Bourdoncle. Efficient Chaotic Iteration Strategies with Widenings. In *International Conference on Formal Methods in Programming and their Applications*, volume 735 of *LNCS*. Springer-Verlag, 1993.
4. M. Bozga, J.-C. Fernandez, and L. Ghirvu. State Space Reduction based on Live Variables Analysis. In *SAS*, volume 1694 of *LNCS*, Venezia, IT, 1999.
5. M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J.-P. Krimm, L. Mounier, and J. Sifakis. IF : An Intermediate Representation for SDL and its Applications. In *SDL Forum Proceedings*, Montreal, CA, June 1999.
6. M. Bozga, J.-C. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, and L. Mounier. Verification and Test Generation for the SSCOP Protocol. *Science of Computer Programming*, 1999.
7. E. Brinksma, R. Alderden, R. Langerak, J. Van de Lagemaat, and J. Tretmans. A Formal Approach to Conformance Testing. In J. De Meer, L. Mackert, and W. Effelsberg, editors, *2^{nd} International Workshop on Protocol Test Systems*. North Holland, 1990.
8. P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. Technical report, LIX, Ecole Polytechnique, 91128 Palaiseau Cedex FR, May 1990.
9. M. Dwyer. Data Flow Analysis Frameworks for Concurrent Programs. Technical Report UM-CS-1995-062, University of Massachusetts at Amherst, US, 1995.
10. J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *Proceedings of the 8th International Conference on Computer Aided Verification*, number 1102 in LNCS. Springer-Verlag, 1996.
11. J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Science of Computer Programming*, 29, 1997.
12. International Organization for Standardization. OSI-Open Systems Interconnection, Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework, 1992. Standard ISO/IEC 9646-1/2/3.
13. R. Gupta, M.J. Harrold, and M.L. Soffa. Program Slicing-Based Regression Testing Techniques. *Journal of Software Testing, Verification and Reliability,*, June 1996.

14. J. Hatcliff, J. Corbett, M. Dwyer, S. Sokolowski, and H. Zheng. A Formal Study of Slicing for Multi-Threaded Programs with JVM Concurrency Primitives. In *SAS*, volume 1694 of *LNCS*, Venezia, IT, 1999. Springer-Verlag.
15. ITU-T SG 10/Q.8. ISO/IEC JTC1/SC21 WG7. Information Retrieval, Transfer and Management for OSI; Framework: Formal Methods in Conformance Testing. Technical report, International Organization for Standardization - ISO, 1996.
16. G. Kildall. A Unified Approach to Global Program Optimization. In *ACM Symposium on Principles of Programming Languages*, 1973.
17. S. Muchnick. *Advanced Compiler Design Implementation.* Morgan Kaufmann Publishers, 1997.
18. M. Phalippou. Test Sequence using Estelle or SDL Structure Information. In *FORTE*, Berne, CH, October 1994.
19. H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood, John Wiley, 1988.
20. D. Rayner. OSI Conformance Testing. *Computer Networks and ISDN Systems*, 14, 1987.
21. M. Schmitt, B. Koch, J. Grabowski, and D. Hogrefe. Autolink - A Tool for Automatic and Semi-automatic Test Generation from SDL Specifications. Technical Report A-98-05, Medical Univ. of Lübeck, DE, 1998.
22. J. Tretmans. A Formal Approach to Conformance Testing. In *6th International Workshop on Protocols Test Systems*, number C-19 in IFIP Transactions, pages 257–276, 1994.
23. M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, SE-10(4), July 1984.