Using static analysis to improve automatic test generation

Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu

Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Using Static Analysis To Improve Automatic Test Generation*

Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu

VERIMAG**, Centre Equation, 2 avenue de Vignate, F-38610 Gières
e-mail: Marius.Bozga@imag.fr, Lucian.Ghirvu@imag.fr, Jean-Claude.Fernandez@imag.fr

The date of receipt and acceptance will be inserted by the editor

Abstract. Conformance testing is still the main industrial validation technique for telecommunication protocols. In practice, the automatic construction of test cases based on finite-state models is hindered by the state explosion problem. We try to reduce its magnitude by using static analysis techniques in order to obtain smaller but equivalent models.

Key words: static analysis, slicing, conformance testing, asynchronous systems, 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. In the case of protocols, conformance testing has been completely formalized by [8,34,18] and is also standardized within [17]. Test cases can be automatically generated from formal specifications and several tools such as TGV [12], TVEDA-2 [25], TESTCOMPOSER [22], AUTOLINK [29] or TORX [2] are now fully operational.

Conformance test cases are automatically constructed by exploring a synchronous product between a model obtained from the specification and some *test purpose*, both represented as labeled transition systems. A test purpose is used to select *interesting* test cases within the specification model.

* This work was supported in part by the European Commission (IST project ADVANCE, contract No IST-1999-29082 and IST project AGEDIS, contract No IST-1999-20218) and by Région Rhône-Alpes, France

** VERIMAG is a joint laboratory of CNRS, UJF and INPG Grenoble

The central problem arising here is the well known state explosion. There are mainly two reasons: *concurrency*, which is usually flattened using an interleaving semantics and *data*, which are also evaluated to all possible, distinct values. Various solutions exist and have been implemented to deal with state explosion. For instance, *on-the-fly techniques* attempt to explore only a part of the model e.g, guided by the test purpose. *Symbolic techniques* rely on symbolic and compact representations of sets of states instead of individual states. In this context, we propose a solution relying on the use of static simplifications of specifications depending on test purposes, before model generation. This approach can be used in combination with the above reduction techniques. Moreover, it is not limited to test generation, but it can be applied to model-checking in general.

Our contribution

We propose to simplify specifications by means of different slicing methods with respect to information from test purposes. We consider specifications as being asynchronously communicating extended finite-state automata and test purposes as extended finite-state automata with constraints.

A first slicing method takes into account the set of interactions between specification components, starting from inputs enabled in the test purpose. We obtain a first reduction of the specification, consisting of the part which is statically *reachable*, given the inputs of the test purpose. A second slicing, computes variables and parameters which may be relevant to outputs observed by the test purpose. All other, *irrelevant* variables and associated actions are safely discarded. Finally, the specification is sliced with respect to constraints on data attached to the test purpose. The constraints we consider are either *constants* (i.e, variable equals some value at

a control point) or *intervals* (i.e, for numerical variables only, restriction to intervals of values).

All these analyses transform specifications without loss of information with respect to the test purpose. Moreover, they are independent and can be implemented separately. Furthermore, they can be applied iteratively, in any order, until no more reduction is obtained – a fixpoint is always reached given that specifications are statically finite.

We have implemented all these analysis methods in the context of the IF tool-set [6], a platform for the validation of distributed real-time systems developed at Verimag. We have already experimented them in several industrial case studies and have obtained very good results.

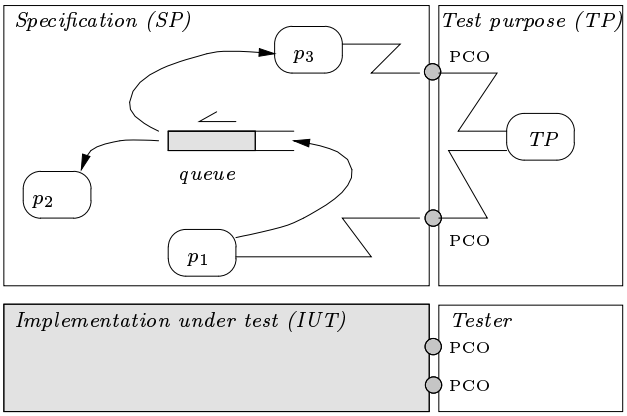

Fig. 1. Test architecture

Related work

Static program slicing has been introduced by Weiser [35] as a technique for analyzing program dependencies. Given a *slicing criterion* (s, V), where s is a control point and V a set of program variables, a program slice consists of a subset of statements that affect (*backward slicing*) or are affected by (*forward slicing*) the values of variables at s. Korel and Laski [24] introduced the notion of *dynamic slicing*. In this case, only the dependencies that occur in a specific execution of the program are taken into account. A *dynamic slicing criterion* specifies not only a a set of variables and a statement, but the different occurrences in the execution history. A survey of algorithms for program slicing can be found in [32]. In our work, we consider only static slicing.

In general, slicing is used to automate work on software debugging, testing and maintainance and we mention here just a few typical examples of its use. In [14] the authors present an approach for selective regression testing using slicing. Regression testing identifies the parts of the program that are affected by some change. In [28], the authors present a testing and debugging methodology for Spreadsheet languages based on program slicing. In [30], slicing is defined on interprocedural control flow in order to be applicable to large software. TVEDA-2 [25] produces test cases from SDL specifications by performing simple syntactic transformations on them. A different application is given in [15], where slicing is used for *verification* purposes, in order to extract finite-state machines from multi-threaded programs.

Outline

The remainder of the paper is structured as follows. Section 2 briefly reviews the notions of conformance testing and presents the underlying model. In section 3, we introduce and formalize the slicing techniques of the specification with respect to the test purpose. We give some experimental results in section 4 showing the interest of our simplifications on complex specifications. We conclude in section 5.

2 Conformance test case generation

According to the classification of test architectures from [17,27], we consider a *local single-layer test method* with synchronous communication between the tester and the *implementation under test* (IUT). It is *local* because all the events from the environment to the IUT are under the control of the tester. 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). The communication at the PCOs is synchronous (by rendez-vous). The architecture is shown in Figure 1.

2.1 Specification

We consider specifications consisting of concurrent processes, communicating through signal passing via a set of unbounded queues. We distinguish between *internal* queues (communications inside the system) and *external* queues (communications between the system and its environment). Processes are represented by finite-state automata extended with actions on queues and local variables. Without loss of generality, actions are simple guarded commands and signals carry a single parameter.

Definition 1 (syntax). *A specification SP is a tuple* (S, C, P) *where* S *is the set of signals,* $C = C^{\text{int}} \cup C^{\text{ext}}$ *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 control 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 control transitions. An action can be either a guarded assignment* $[\ b\ ]\ x := e$*, a guarded input* $[\ b\ ]\ c?s(x)$*, or a guarded output* $[\ b\ ]\ c!s(e)$*.* b *and* e *are expressions,* $x \in X_p$ *is a variable,* $c \in C$ *is a queue and*

$$\frac{-}{\hat{q}^0_{\rm sp} \in \widehat{Q}_{\rm sp}} \tag{1}$$

$$\frac{(\theta,\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad q_p \xrightarrow{[b]\ x:=e} q'_p \quad \sigma(b) = \mathbf{t} \quad \sigma(e) = v}{(\theta',\sigma',\rho') = (\theta[q'_p/q_p],\sigma[v/x],\rho) \in \widehat{Q}_{\rm sp} \quad (\theta,\sigma,\rho) \xrightarrow{\tau} (\theta',\sigma',\rho') \in \widehat{T}_{\rm sp}} \tag{2}$$

$$\frac{(\theta,\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad q_p \xrightarrow{[b]\ c!s(e)} q'_p \quad c \in C^{\rm ext} \quad \sigma(b) = \mathbf{t} \quad \sigma(e) = v}{(\theta',\sigma',\rho') = (\theta[q'_p/q_p],\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad (\theta,\sigma,\rho) \xrightarrow{c!s(v)} (\theta',\sigma',\rho') \in \widehat{T}_{\rm sp}} \tag{3}$$

$$\frac{(\theta,\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad q_p \xrightarrow{[b]\ c!s(e)} q'_p \quad c \in C^{\rm int} \quad \sigma(b) = \mathbf{t} \quad \sigma(e) = v \quad \rho(c) = w}{(\theta',\sigma',\rho') = (\theta[q'_p/q_p],\sigma,\rho[w.s(v)/c]) \in \widehat{Q}_{\rm sp} \quad (\theta,\sigma,\rho) \xrightarrow{\tau} (\theta',\sigma',\rho') \in \widehat{T}_{\rm sp}} \tag{4}$$

$$\frac{(\theta,\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad c \in C^{\rm ext} \quad \sigma(b) = \mathbf{t} \quad v \in D \setminus \{\perp\}}{(\theta',\sigma',\rho') = (\theta[q'_p/q_p],\sigma[v/x],\rho) \in \widehat{Q}_{\rm sp} \quad (\theta,\sigma,\rho) \xrightarrow{c?s(v)} (\theta',\sigma',\rho') \in \widehat{T}_{\rm sp}} \tag{5}$$

$$\frac{(\theta,\sigma,\rho) \in \widehat{Q}_{\rm sp} \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad c \in C^{\rm int} \quad \sigma(b) = \mathbf{t} \quad v \in D \quad \rho(c) = s(v).w}{(\theta',\sigma',\rho') = (\theta[q'_p/q_p],\sigma[v/x],\rho[w/c]) \in \widehat{Q}_{\rm sp} \quad (\theta,\sigma,\rho) \xrightarrow{\tau} (\theta',\sigma',\rho') \in \widehat{T}_{\rm sp}} \tag{6}$$

Table 1. Semantic rules for specifications.

$s \in S$ is a signal. We denote by $q_p \xrightarrow{\alpha} q'_p$, a transition $(q_p, \alpha, q'_p) \in T_p$.

We give the semantics of specifications in terms of basic labeled transition systems (without variables). We assume the existence of a universal domain D which contains the values of variables and signal parameters. The boolean values $\{\mathbf{t}, \mathbf{f}\}$ and the special *undefined* $\perp$ value are contained in D. We define variable contexts as total mappings $\sigma : \bigcup_{p \in P} X_p \to D$ from variables to values. We extend these mappings to expressions in the usual way (in the case of a variable x and an expression e such that $x \in vars(e)$ and $\sigma(x) = \perp$ we have $\sigma(e) = \perp$).

We define queue contexts as total mappings $\rho : C^{\rm int} \to (S \times D)^*$ which associate with each internal queue c a sequence of messages. Messages are pairs (s, v) denoted also by $s(v)$, where s is a signal and v the carried parameter value. The empty sequence is denoted as ϵ.

Definition 2 (semantics). *The semantics of a specification SP is given by a labeled transition system $\widehat{SP} = (\widehat{Q}_{\rm sp}, \widehat{T}_{\rm sp}, \hat{q}^0_{\rm sp})$. States $\hat{q}_{\rm sp}$ of this system are triples of the form (θ, σ, ρ), where σ is a variable context, ρ 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 (they correspond to assignments or internal communication) or visible and labeled with the corresponding action (on external communication). The set of transitions is the smallest set obtained by the rules given in table 1. In the initial state $\hat{q}^0_{\rm sp} \in \widehat{Q}_{\rm sp}$ internal queues are empty and processes are at initial control states, and all variables have some default initial values.*

2.2 Test purpose and consistency

A test purpose is an extended finite-state automaton which describes a pattern of interactions between the tester and the IUT. It is usually described from the perspective of the implementation i.e, inputs and outputs of the test purpose mean respectively inputs and outputs of the implementation.

In our setting, inputs of the test purpose have constraints attached to the parameter value. Constraints restrict the allowed values of the parameter to some subset of its domain, for example, to some *constant* (unique, fixed) value, or to an *interval* of values, etc. Contrarily to inputs, output parameters are always unconstrained.

This definition has been inspired by TTCN[1] and has the following intuition: it is possible to constrain values sent to the implementation (because the signal is sent by the tester and thus controlled) but one can only observe values received from the IUT (because the implementation cannot be controlled by the tester).

Definition 3 (test purpose). *A test purpose TP is a tuple $(Q_{\rm tp}, T_{\rm tp}, q^0_{\rm tp}, Q^{\rm acc}_{\rm tp})$ where $Q_{\rm tp}$ is a set of states, $T_{\rm tp} \subseteq Q_{\rm tp} \times \Sigma_{\rm tp} \times Q_{\rm tp}$ is a set of transitions and $Q^{\rm acc}_{\rm tp} \subseteq Q_{\rm tp}$ is a set of accepting states, without successors by $T_{\rm tp}$.*

[1] Tree and Tabular Combined Notation [17]

$\Sigma_{\rm tp}$ is the set of interactions α_{tp} which are controlled or observed by the test purpose. This set is partitioned into the set of feeds *$\Sigma_{\rm f}$ containing the constrained signal* inputs *$c?s(\mathcal{C})$ and the set of* observations *$\Sigma_{\rm o}$ containing the unconstrained signal outputs $c!s(*)$. We denote by $c \in C^{\rm ext}$ an external queue, $s \in S$ a signal, $\mathcal{C}$ a constraint and $*$ any value.*

In order to ensure the feasibility of the test generation method, we rely on a notion of *consistency* between the test purpose and the specification. Consistency ensures that the set of behaviors described by the test purpose is included in the set of behaviors described by the specification. More precisely, a test purpose TP is said to be consistent with respect to a specification SP if there exists a *consistency preorder* relating them.

Definition 4 (consistency preorder). *Let $\widehat{SP} = (\widehat{Q}_{\rm sp}, \widehat{T}_{\rm sp}, \hat{q}^0_{\rm sp})$ be the semantic model of the specification, and $TP = (Q_{\rm tp}, T_{\rm tp}, q^0_{\rm tp}, Q^{\rm acc}_{\rm tp})$ be the test purpose. A binary relation $\mathcal{R} \subseteq \widehat{Q}_{\rm sp} \times Q_{\rm tp}$ is a consistency preoder if and only if for all $(\hat{q}_{\rm sp}, q_{\rm tp}) \in \mathcal{R}$ whenever $q_{\rm tp} \xrightarrow{\alpha} q'_{\rm tp}$ then exists $\hat{q''}_{\rm sp}, \hat{q}'_{\rm sp}$, such that $\hat{q}'_{\rm sp}$ is reachable from $\hat{q}_{\rm sp}$ by a trace which does not contain α, and $\hat{q}''_{\rm sp} \xrightarrow{\alpha} \hat{q}'_{\rm sp}$ and $(\hat{q}'_{\rm sp}, q'_{\rm tp}) \in \mathcal{R}$.*

The notion of consistency of a test purpose with respect to a specification is different of the notion of *conformance relation* relating the implementation and its specification [33,25]. The conformance relation states that inputs which are not accepted by the specification may be accepted by the implementation but outputs produced by the implementation must also be produced by the specification.

2.3 Synchronous product

Test cases are automatically constructed by exploring the synchronous product between the model of the specification $\widehat{SP} = (\widehat{Q}_{\rm sp}, \widehat{T}_{\rm sp}, \hat{q}^0_{\rm sp})$ and the test purpose $TP = (Q_{\rm tp}, T_{\rm tp}, q^0_{\rm tp}, Q^{\rm acc}_{\rm tp})$. We formally define the synchronous product since all of the preservation results rely on it. However, we do not describe *how* tests are selected from the product, for this aspect see [21].

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

$$\frac{-}{(\hat{q}^0_{\rm sp}, q^0_{\rm tp}) \in Q_\pi} \tag{7}$$

$$\frac{(\hat{q}_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad \hat{q}_{\rm sp} \xrightarrow{\tau} \hat{q}'_{\rm sp} \quad q_{\rm tp} \notin Q^{\rm acc}_{\rm tp}}{(\hat{q}'_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad (\hat{q}_{\rm sp}, q_{\rm tp}) \xrightarrow{\tau} (\hat{q}'_{\rm sp}, q_{\rm tp}) \in T_\pi} \tag{8}$$

$$\frac{(\hat{q}_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad \hat{q}_{\rm sp} \xrightarrow{c?s(v)} \hat{q}'_{\rm sp} \quad q_{\rm tp} \xrightarrow{c?s(\mathcal{C})} q'_{\rm tp} \quad \mathcal{C}(v)}{(\hat{q}'_{\rm sp}, q'_{\rm tp}) \in Q_\pi \quad (\hat{q}_{\rm sp}, q_{\rm tp}) \xrightarrow{c?s(v)} (\hat{q}'_{\rm sp}, q'_{\rm tp}) \in T_\pi} \tag{9}$$

$$\frac{(\hat{q}_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad \hat{q}_{\rm sp} \xrightarrow{c?s(v)} \hat{q}'_{\rm sp} \quad c?s(\mathcal{C}) \in \Sigma_{\rm f} \quad \mathcal{C}(v)}{(\hat{q}'_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad (\hat{q}_{\rm sp}, q_{\rm tp}) \xrightarrow{c?s(v)} (\hat{q}'_{\rm sp}, q_{\rm tp}) \in T_\pi} \tag{10}$$

$$\frac{(\hat{q}_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad \hat{q}_{\rm sp} \xrightarrow{c!s(v)} \hat{q}'_{\rm sp} \quad q_{\rm tp} \xrightarrow{c!s(*)} q'_{\rm tp}}{(\hat{q}'_{\rm sp}, q'_{\rm tp}) \in Q_\pi \quad (\hat{q}_{\rm sp}, q_{\rm tp}) \xrightarrow{c!s(v)} (\hat{q}'_{\rm sp}, q'_{\rm tp}) \in T_\pi} \tag{11}$$

$$\frac{(\hat{q}_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad \hat{q}_{\rm sp} \xrightarrow{c!s(v)} \hat{q}'_{\rm sp}}{(\hat{q}'_{\rm sp}, q_{\rm tp}) \in Q_\pi \quad (\hat{q}_{\rm sp}, q_{\rm tp}) \xrightarrow{c!s(v)} (\hat{q}'_{\rm sp}, q_{\rm tp}) \in T_\pi} \tag{12}$$

3 Static analysis for testing

The purpose of static analysis is to compute the (minimal) part of the specification which is relevant for the test purpose. We present three distinct analyses:

1. the *relevant control analysis* restricts the processes to the sets of control states and control transitions which are reacheable via the transition relations T_p of SP, given the feeds of the test purpose; this analysis corresponds to a *forward static slicing* method based on control dependencies,
2. the *relevant variables analysis* computes and simplifies processes with respect to variables which are used to compute values needed for observations of the test purpose; this analysis corresponds to a *backward static slicing* method based on data dependecies,
3. the *constraint propagation* aims to further simplify processes, given the constraints attached to feeds of the test purpose.

Each analysis takes as input a specification and provides an transformed (smaller) one, but still equivalent with respect to the synchronous product operation. The three analyses are completely independent and can be applied in any order. Moreover, they can be applied iteratively because the reduction obtained by one analysis can be further exploited by another and so on, until no more reduction is possible.

Example 1. In order to illustrate all three analyses let us consider the toy example presented in Figure 2. It consists of two processes communicating through an internal queue $cl \in C^{\rm int}$. The external queues are $ci, co \in C^{\rm ext}$. We want to generate some test case containing the input $ci?sr(x)$, with $x \in [1, 10]$ followed by the output $co!sa$. Therefore, the set of feeds is $\Sigma_{\rm f} = \{ci?sr([1, 10])\}$ and the set of observations is $\Sigma_{\rm o} = \{co?sa(*)\}$.

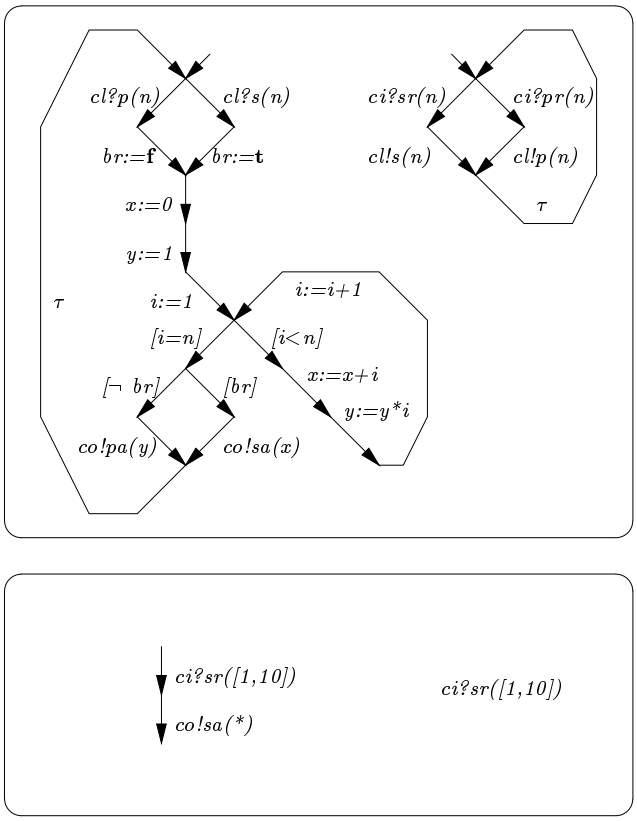

Fig. 2. Example

3.1 "Relevant control" analysis

This analysis restricts each process to the set of control states and control transitions that may be reached given the feeds. Intuitively, it 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. This analysis corresponds to a *forward static slicing* method based on control dependencies.

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} of a test purpose TP to be the specification $SP \setminus_{\mathrm{f}} \Sigma_{\mathrm{f}} = (S, C, P \setminus_{\mathrm{f}} \Sigma_{\mathrm{f}})$, where $P \setminus_{\mathrm{f}} \Sigma_{\mathrm{f}}$ 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 sets satisfying the following rules:*

$$\frac{-}{q_p^0 \in Q'_p} \tag{13}$$

$$\frac{q_p \in Q'_p \quad q_p \xrightarrow{[b]\ x:=e} q'_p}{q'_p \in Q'_p \quad q_p \xrightarrow{[b]\ x:=e} q'_p \in T'_p} \tag{14}$$

$$\frac{q_p \in Q'_p \quad q_p \xrightarrow{[b]\ c!s(e)} q'_p}{q'_p \in Q'_p \quad q_p \xrightarrow{[b]\ c!s(e)} q'_p \in T'_p} \tag{15}$$

$$\frac{q_p \in Q'_p \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad c?s(\mathcal{C}) \in \Sigma_{\mathrm{f}}}{q'_p \in Q'_p \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \in T'_p} \tag{16}$$

$$\frac{q_p \in Q'_p \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad c \in C^{\mathrm{int}} \ \exists r.q_r \xrightarrow{[b']\ c!s(e)} q'_r \in T'_p}{q'_p \in Q'_p \quad q_p \xrightarrow{[b]\ c?s(x)} q'_p \in T'_p} \tag{17}$$

We mention here the input/output propagation between processes. That is, an input transition is part of the slice if there exists at least one corresponding output transition which is already part of the slice.

The algorithm computing the sliced system proceeds in an iterative manner. It maintains the sets of control states and control transitions *reached* for each process. Initially, the sets of states contain the initial states of the processes, and the sets of transitions are empty. Then, at each step, one of the rules is applied and sets of reached states and transitions are updated accordingly. The algorithm stops when no rule is applicable any more. The complexity of this algorithms is $\mathcal{O}(\sum_p |Q_p| + |T_p|)$, linear with respect to the number of control states and control transitions in the specification.

Example 2. Slicing wrt feeds applied on the specification from Figure 2, results in the specification shown in Figure 3. The external input $ci?pr(n)$ is uncovered by the feeds and therefore it is eliminated; in turn, this 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 $br := \mathrm{f}$.

Slicing with respect to feeds preserves the synchronous product.

Theorem 1 (correctness of slicing wrt feeds). *Let $SP = (S, C, P)$ be a specification, TP a test purpose and Σ_{f} the set of feeds of TP. The synchronous products between the models of SP respectively $SP \setminus_{\mathrm{f}} \Sigma_{\mathrm{f}}$ and the test purpose TP are the same:*

$$\prod(\widehat{SP}, TP) = \prod(\widehat{SP \setminus_{\mathrm{f}} \Sigma_{\mathrm{f}}}, TP) \ .$$

Proof. In one direction, it is obvious that the initial product (between the specification and the test purpose) contains the sliced product (between the sliced specification and the same test purpose) because slicing may only eliminate some parts of the specification. In the other direction, the inclusion is proved by induction. By definition, both products contain the same initial state. If some state is present in both products, all its successors in the global product can be also found in the sliced product. More exactly, if some successor is reached (cf.

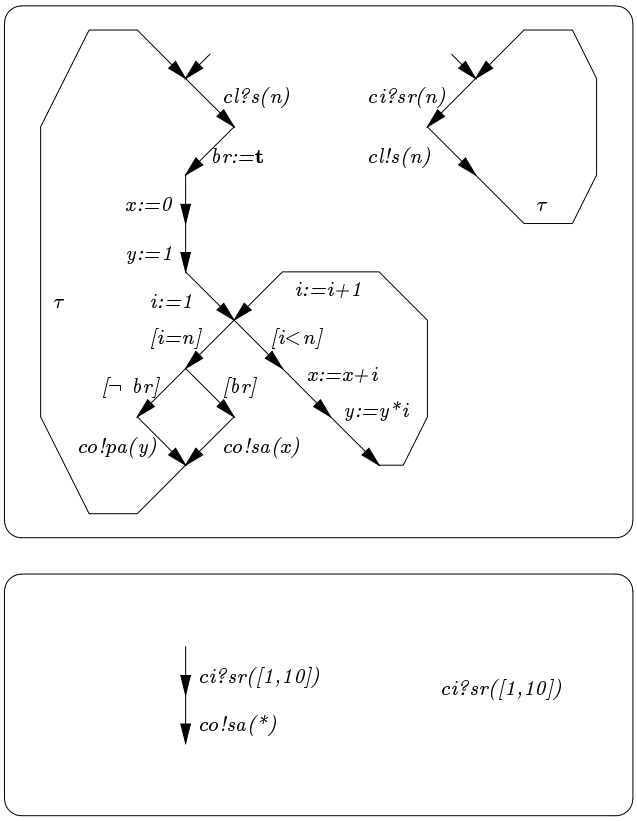

Fig. 3. Example slicing wrt feeds

some of the rules 7-12) in the global product, then the corresponding (control) state and transition are also in the sliced specification (cf. the equivalent rule in 13-17), and therefore it can also be reached in the sliced product. □

3.2 "Relevant variables" analysis

This analysis is an extension of live variable analysis [4]. It attempts to compute, for each process, the set of relevant variables in each control state. The relevance is defined with respect to test purpose observations: a variable is relevant in a control state if its value in that state might influence the parameter value of some signal output occurring in the test purpose. Or, similar to the definition of live variables, we consider a variable to be relevant in a control 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 here, we consider a variable to be *used* only when it is used in external outputs observed by the test purpose, or in assignments to relevant variables (possibly via internal inputs).

Definition 7 (variables relevant wrt observations). *Let $SP = (S, C, P)$ be a specification and $TP = (Q_{\rm tp}, T_{\rm tp}, q^0_{\rm tp}, Q^{\rm acc}_{\rm tp})$ be a 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

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

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

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 set of relevant variables for the source state is updated. The algorithms ends and the least fixed point is reached when no more change in the relevance sets occurs for any transition.

Also in this analysis, the relevance of variables is propagated between processes. In fact, variables used in expressions sent through internal communication channels are relevant at the sender side if there exists at least one possible destination in which their value is relevant.

Slicing with respect to observations attempts to reduce the number of variables used inside processes. Concretely, we eliminate assignements of irrelevant variables. Irrelevant variables used in inputs are replaced by some *don't care* variable $\top_p$. Finally, expressions occurring in unused outputs are replaced by the undefined value $\perp$. This transformation is formally described below.

Definition 8 (slicing wrt observations). *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 observations to be the specification $SP \setminus_{\rm o} \Sigma_{\rm o} = (S, C, P \setminus_{\rm o} \Sigma_{\rm o})$, where $P \setminus_{\rm o} \Sigma_{\rm o}$ 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 define irrelevant variables anymore:*

$$\frac{q_p \xrightarrow{[b]\ x:=e} q'_p \quad x \in Rlv_p(q'_p)}{q_p \xrightarrow{[b]\ x:=e} q'_p \in T'_p} \tag{18}$$

$$\frac{q_p \xrightarrow{[b]\ x:=e} q'_p \quad x \notin Rlv_p(q'_p)}{q_p \xrightarrow{[b]\ \tau} q'_p \in T'_p} \tag{19}$$

$$\frac{q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad x \in Rlv_p(q'_p)}{q_p \xrightarrow{[b]\ c?s(x)} q'_p \in T'_p} \tag{20}$$

$$\frac{q_p \xrightarrow{[b]\ c?s(x)} q'_p \quad x \notin Rlv_p(q'_p)}{q_p \xrightarrow{[b]\ c?s(\top_p)} q'_p \in T'_p} \tag{21}$$

$$\frac{q_p \xrightarrow{[b]\ c!s(e)} q'_p \quad Use(c!s)}{q_p \xrightarrow{[b]\ c!s(e)} q'_p \in T'_p} \tag{22}$$

$$\frac{q_p \xrightarrow{[b]\ c!s(e)} q'_p \quad \neg Use(c!s)}{q_p \xrightarrow{[b]\ c!s(\bot)} q'_p \in T'_p} \tag{23}$$

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

The complexity of slicing with respect to outputs is $\mathcal{O}(\sum_p |Q_p||X_p|)$, linear with respect to the product of the number of variables and the number of control states for each process.

Example 3. Slicing wrt observations applied to 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 by τ and the output $co!pa(y)$ becomes $co!pa(\bot)$ because $\neg Use(co!pa)$.

Intuitively, slicing wrt observations preserves the model of the specification up to the 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 output $c!s(v)$ which is not specified by the test purpose i.e., $c!s(*) \notin \Sigma_{\mathrm{o}}$, is renamed into $c!s(\bot)$. The other actions are left unchanged.

$$\frac{p \xrightarrow{c!s(v)} q \quad c!s(*) \notin \Sigma_{\mathrm{o}}}{p \xrightarrow{c!s(\bot)} q} \tag{24}$$

In this way, we abstract from the exact parameter values for outputs, other than those occurring in the test purpose. We note the renamed model by $\widehat{SP} \downarrow \Sigma_{\mathrm{o}}$. The following theorem holds.

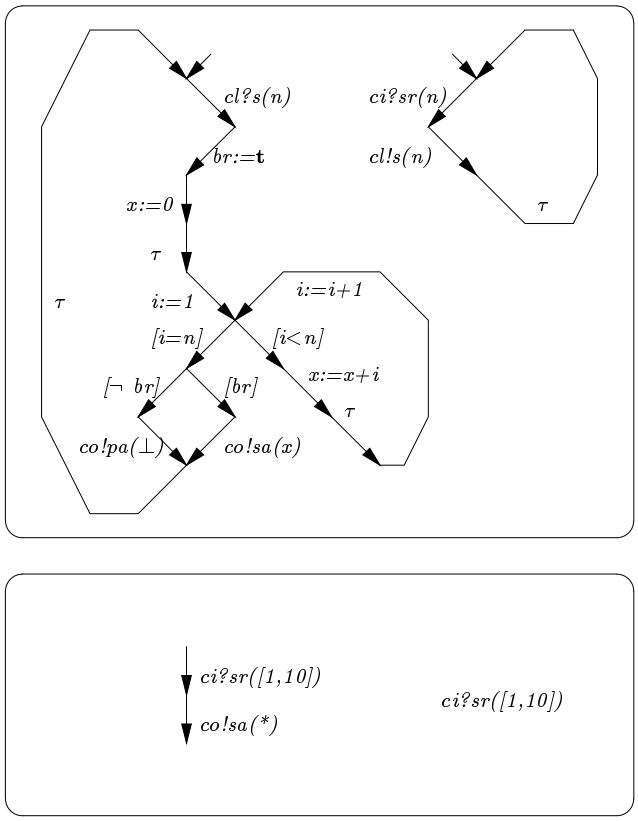

Fig. 4. Example slicing wrt observations

Theorem 2 (correcteness of slicing wrt observations). *Let* $SP = (S, C, P)$ *be a specification and* $TP = (Q_{\mathrm{tp}}, T_{\mathrm{tp}}, q^0_{\mathrm{tp}}, Q^{\mathrm{acc}}_{\mathrm{tp}})$. *The model of* SP *renamed with respect to the observable outputs* Σ_{o} *and the model of* $SP \setminus_{\mathrm{o}} \Sigma_{\mathrm{o}}$ *are strongly bisimilar. In turn, the synchronous product* $\prod(\widehat{SP}, TP)$ *and* $\prod(\widehat{SP \setminus_{\mathrm{o}} \Sigma_{\mathrm{o}}}, TP)$ *are strongly bisimilar.*

Proof. Let us consider the following relation between initial and sliced model states: a state (θ, σ, ρ) of SP is related to some state $(\theta', \sigma', \rho')$ of $SP \setminus_{\mathrm{o}} \Sigma_{\mathrm{o}}$ if and only if *for each process* p, *the control states are identical in both model states i.e* $\theta(p) = \theta'(p)$, *all relevant variables have the same values* $\sigma(x) = \sigma'(x)\ \forall x \in Rlv_p(\theta(p))$ *and all queue contents are identical up to values attached to irrelevant signals* $\rho(c) \sim \rho'(c) \quad \forall c \in C^{int}$ – a signal s is irrelevant if none of the processes uses the values of its parameter (that is $\neg Use(c!s)$). It is easy to see that the relation above is a bisimulation between these two models, that is, starting from related states, the same transitions are enabled in both models and they lead again to related states. In addition, given that the initial states of both models are also related it turns out that the two models are strongly bisimlar. □

Notice also the possibility of a more general application of relevant variables. In fact, we exploit them at

a purely syntactic level for example, by eliminating the irrelevant variables and their dependencies in the specification. However, it is possible to take them into account in a different manner: one can reinitialize them by some default value as soon as they become irrelevant and still obtain a bisimilar reduced model.

3.3 Constraint propagation

This section provides an approach to simplify the specification using the constraints imposed on the inputs of the test purpose. Currently, the constraints we consider have the form of constants ([23]) or integer intervals ([9]). First, these constraints can be attached to possible matching inputs. Then, using some intra/interprocesses data flow analysis algorithms, the constraints are propagated through the specification. 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 never firable.

We sketch the *constraint propagation* problem and a possible solution. Again, it is a data flow analysis problem whose basic components are:

1. the *flow graph* is composed of the (control) states and (control) transitions of each process and some auxiliary constructions in order to simulate the internal queues,
2. the *constant lattice* C_D of values of the domain D, the *interval lattice* I_D of intervals of values of the domain D and respectively the lattice L representing the disjoint union $C_D \oplus I_D$,
3. the class of *transfer* functions $Transfer_{t_p} : (X_p \to L) \to (X_p \to L)$, for each transition $t_p : q_p \xrightarrow{[b]\ \alpha} q'_p$:
   (a) if α is $x := e$ then
   $$Transfer_{t_p}(f)(x') = \begin{cases} f(x') & \text{if } x' \neq x \\ f(e) & \text{if } x' = x \end{cases}$$
   (b) if α is $c?s(x)$ then
   $$Transfer_{t_p}(f)(x') = \begin{cases} f(x') & \text{if } x' \neq x \\ \mathcal{C}(x') & \text{if } x' = x \ \wedge \\ & \quad c \in C^{\text{ext}} \ \wedge \\ & \quad c?s(\mathcal{C}) \in \Sigma_{\text{f}} \\ \top & \text{otherwise} \end{cases}$$
   (c) if α is $c!s(e)$ then $Transfer_{t_p}(f) = f$
   One can observe that in the definition of $Transfer_{t_p}$ functions guards are ignored. In fact, they are taken into account only later, in order to enable the application of a transfer function (see fixpoint equation 25).
4. the confluence functions $\bigsqcup$, one for each state.

By choosing the constraints to be the elements of L we ensure the possibility to test the emptiness of a constraint and to have a partial order among them. Also, 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. This requirement is fulfilled by defining the operations with set of values similarly as in *interval arithmetic* [26].

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

Definition 9 (constraint propagation). *Let $SP = (S, C, P)$ be a specification and Σ_{f} the set of feeds of the a test purpose TP. Constraints are represented, for each process by functions $Val_p : Q_p \to (X_p \to L)$ (extended to expressions, as usually). With the notations presented before, the constraint propagation problem is defined as finding the least fix point of the following equation system:*

$$Val_p(q'_p) = \bigsqcup_{t_p : q_p \xrightarrow{[b]\ \alpha} q'_p} Transfer_{t_p}(Val_p(q_p)) \quad \text{if } Val(q_p)(b) \neq false \tag{25}$$

The algorithm used for solving the constraint propagation problem in the case of the lattice of constants is the classical iterative algorithm from [23] with an interprocesses variant such as [11]. When we consider the lattice of intervals, which has an infinite height, we use for each process a widening technique as in [3] in order to guarantee convergence.

The results of the constraint propagation problem are used again to simplify the specification. 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 $SP \setminus_{\text{c}} \Sigma_{\text{f}} = (S, C, P \setminus_{\text{c}} \Sigma_{\text{f}})$, where $P \setminus_{\text{c}} \Sigma_{\text{f}}$ 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 sets satisfying the following rules:*

$$\frac{-}{q_p^0 \in Q'_p} \tag{26}$$

$$\frac{q_p \in Q'_p \quad q_p \xrightarrow{[b]\ \alpha} q'_p \in T_p \quad Val_p(q_p)(b) \neq \{\mathbf{f}\}}{q'_p \in Q'_p \quad q_p \xrightarrow{[b]\ \alpha} q'_p \in T'_p} \tag{27}$$

Example 4. Slicing wrt constraints applied to the specification and the test purpose of Figure 4, produces the specification shown in Figure 5. The value **t** for *br* is propagated to the source state for the transition with the guard $[\neg br]$ which guarantee that this transition and the

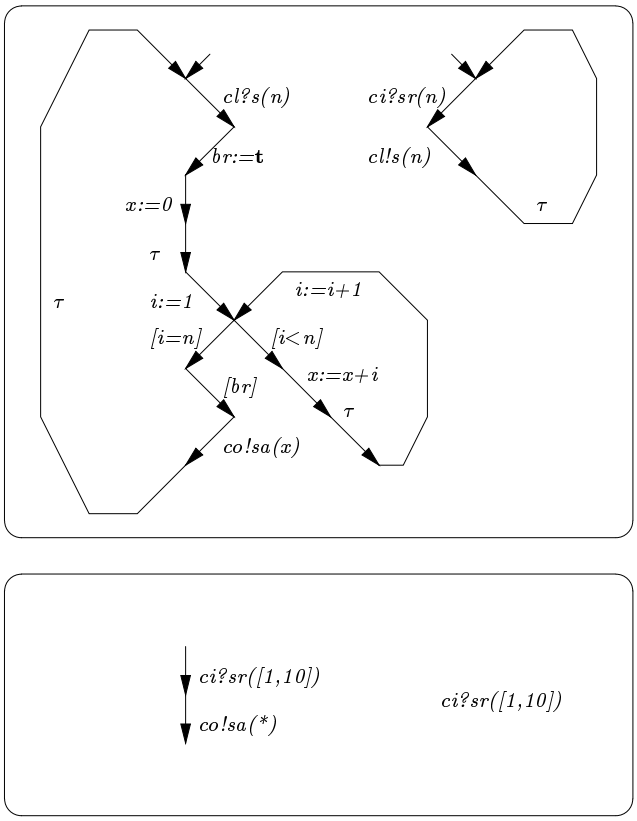

Fig. 5. Example slicing wrt constraints

following $co!pa(\bot)$ are never fired. These transitions are eliminated from the specification. The constraint propagation provides, given the feed $ci?sr([1,10])$, for the output parameter x in the transition $co!sa(x)$ the interval $[1,100]$.

We have the following correctness result:

Theorem 3 (correctness of slicing wrt constraints). *Let $SP = (S, C, P)$ be a specification, $TP = (Q_{\rm tp}, T_{\rm tp}, q^0_{\rm tp}, Q^{\rm acc}_{\rm tp})$ a test purpose and $\Sigma_{\rm f}$ the set of feeds of TP. The synchronous product between the models of SP and $SP \setminus_{\rm c} \Sigma_{\rm f}$ with the test purpose are the same, that is*

$$\prod(\widehat{SP}, TP) = \prod(\widehat{SP \setminus_{\rm c} \Sigma_{\rm f}}, TP) \ .$$

Proof. Constraint propagation and abstract interpretation in general, are a means to overapproximate the set of reachable values of variables at different control points. The slicing we consider, removes only *unreachable* (dead) code with respect to constraints.

Formally, we check inclusion in both directions. In one direction, it is obvious: slicing with respect to constraints removes states and transitions from the initial specification, and therefore the sliced product is included in the initial product. In the other direction, the proof is by induction: the initial state is the same in both products, then, each successor in the global product (cf. rules 7-12) has its equivalent in the sliced product (i.e, the corresponding control transition is preserved cf. rule 27). □

4 Experimentation

The three slicing techniques have been experimented in the context of the IF tool-set [6], an experimental platform for the validation of distributed real-time systems developed at Verimag. This platform is built upon the so-called IF intermediate language, an extended automata-based representation which allows:

- *to represent* significant subsets of specification formalisms for real-time systems, such as SDL [20], LOTOS [16], and currently UML/RT [31];
- to provide *an open validation framework*, able to interconnect verification and test-generation tools, both from the industry or the academia;
- an efficient implementation of *static analysis* techniques.

Currently, the IF platform has a number of components, including compilers, static analysers, code generators, model-checkers, and front-ends to several validation platforms. We focus below on the static analysis components, which implement several analysis ranging from the most classical ones used in compiler optimisation, to more ellaborate ones used in program verification:

- *live analysis* consists in finding live and dead variables in a specification, for each control state: explicit *resets* are added for dead variables in order to reduce redundancy at state-space generation time. This technique is described in [4] and extends the standard live analysis introduced by [1] to queue contents.
- *constant propagation* consists in finding locally constant variables in a specification, for each control state. This is a simple form of the constraint propagation problem presented in section 3.3;
- *clock reduction* is an application of constant propagation for finding constant clock differences in timed automata. It extends the algorithms of [10] and allows the reduction of the number of clocks and timers used in the program, given the functional dependencies which exist between them;
- *slicing* as presented in the previous sections.

In the remainder of this section we give a short overview on the concrete results obtained in several case studies.

4.1 SSCOP Protocol

The first case study is SSCOP (*Service Specific Connection Oriented Protocol*), a part of the ATM Adaption Layer normalized in [19]. We have considered an SDL

specification of the protocol developed by France Telecom R&D. It consists of about 2000 lines of SDL code and describes a single SDL process. It has been translated automatically into an IF specification, with 1075 control states, 1291 control transitions and 134 variables. We consider 10 test purposes concerning different phases of the protocol (connection establishment, disconnection, data transmission, error recovery, etc).

By iterated application of the three slicing techniques *the* SSCOP *specification is statically reduced by 50% to 80% for all considered test purposes*. Such an important reduction factor is typical for this kind of protocols. In fact, the SSCOP specification integrates multiple functionalities, in a single description, and testing usually focuses on a single functionality at a time. It is possible to slice away a large part of the specification, once the test purpose is fixed. For example, all the parts of the SSCOP specification (e.g, states, transitions, variables) involved in *data transfer* can be removed when the test purpose concerns the *connection establishment*.

4.2 MASCARA Protocol

The second case study is MASCARA (*Mobile Access Scheme based on Contention and Reservation for* ATM), a medium access control for wireless ATM. This protocol was proposed by the WAND Consortium (*Wireless* ATM *Network Demonstrator*) as a case study in the VIRES European Project. The MASCARA case study is reported in [13].

We start with an SDL specification of MASCARA which consists of many interacting processes. The complete description is huge: about 300 pages of SDL textual code. In the VIRES project, we focus on formal verification rather than test case generation. Nevertheless, most of the properties to be verified were reachability properties. In this particular case, both problems, verification and test generation, are equivalent: the satisfaction of a property amounts to the existence of a test case for it and vice-versa.

We were not able to check any of the properties without static simplifications. Any attempt to traverse the complete state space failed because of memory limitations. Using slicing, combined with partial order reductions and compositional verification, we manage to verify all the considered properties of the protocol. In this case too, each property concerns one functionality and significant part of the specification can be sliced away prior to verification. For example, in order to verify the *association establishment* property on *access points*, one needs to explore a model of 4,500 states and 12,000 transitions, if slicing is applied, instead of 7,000,000 states and 30,000,000 transitions, without slicing. All the results can be found in [13].

4.3 Ariane-5 Flight Program

The third case study concerns the Ariane-5 [2] flight program – the embedded software which controls the Ariane-5 launcher during its flight. This work has been initiated by the EADS Launchers company to evaluate the applicability of formal methods in softare validation, meaning both formal verification and testing.

The starting points is an SDL specification of the flight program, developed by *reverse engineering* from the actual code and a set of safety requirements the flight program must fulfill. Our aim was on one hand to verify these requirements on the formal specification and, on the other hand, to generate corresponding test cases to be executed on implementations. The main difficulties are the high degree of concurrency (the initial SDL specification contains not less than 31 parallel processes) and the timing constraints (the initial SDL specifications contains 141 timers).

In this case too, we benefit from the slicing techniques presented above, and from static analysis techniques in general. For example, using slicing with respect to feeds we detect (and remove) automatically *passive* processes i.e, not involved at all in the verification or test generation for some fixed requirement. Moreover, using an adapted implementation of constraint propagation for timers, we detect local dependencies between values of timers, for example, $t_1 = t_2 + k$ where t_1, t_2 are timers and k is a constant holds in some state of a process. Such a dependency is used to reduce the overall number of timers needed to express the timing constraints. On Ariane-5 flight program specification, this technique allows to divide the number of timers by 3 i.e, to reduce them from 141 at 55.

All these static simplifications combined with standard techniques like on-the-fly exploration and partial-order reductions allow to completely verify and to generate test cases for all requirements. The results obtained are completely described in [7].

5 Conclusion and future work

In this paper, we show how automatic test generation can take advantage of static analysis. The test generation method we consider is derived from on-the-fly model checking and consists in traversing a synchronous product defined between the specification and a test purpose. Simplifications on the specification may be applied before test generation, by exploiting information on test purposes.

We have considered specifications represented by extended finite-state automata, communicating asynchronously via message queues. Test purposes have been represented

[2] Ariane-5 is an European Space Agency Project delegated to CNES France.

by finite-state automata with constraints. We have proposed three static analyses methods which reduce specifications without loss of information with respect to a test purpose. The first one consists in eliminating from each automaton of the specification, the control states and control transitions which are not reachable given the set of inputs controlled by the test purpose. The second analysis computes the set of variables necessary to compute values observed by the test purpose. All other variables, are safely discarded. The last analysis is constraint propagation.

We have implemented these analyses in the context of IF toolset [6]. We experimented them on several industrial case studies such as SSCOP [5], MASCARA [13] and ARIANE-5 [7] and we obtained very good results.

We plan to extend the static analysis to 1) more general specifications and 2) more general test purposes. For instance, with slight modifications, the slicing methods still apply to specifications combining synchronous (rendez-vous) and asynchronous communication (message passing or shared variables). Moreover, they can be adapted to work on *timed* specifications. In practice, generated test cases depend on timers, which are set and tested against more or less arbitrary values in order to detect deadlocks or livelocks in the implementation. Nevertheless, a finer analysis can be applied to timed specifications to obtain relevant values to be used.

Another investigation point concerns the generation of symbolic tests. Here we have considered test cases without variables. We are currently studying an appropriate extension of the test purposes concept. For instance, the explicit use of variables in addition to constraints can make them much more expressive. Moreover, it may be interesting to reconsider the definition of the synchronous product at a symbolic level (i.e, of extended automata) such that it allows for the generation of symbolic test cases.

References

1. A. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Readings, MA, 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, *12th 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, 1993.
4. M. Bozga, J.Cl. Fernandez, and L. Ghirvu. State Space Reduction based on Live Variables Analysis. In A. Cortesi and G. Filé, editors, *Proceedings of SAS'99 (Venice, Italy)*, volume 1694 of *LNCS*, pages 164–178. Springer, September 1999.
5. M. Bozga, J.Cl. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, and L. Mounier. Verification and Test Generation for the SSCOP Protocol. *Journal of Science of Computer Programming, Special Isssue on Formal Methods in Industry*, 36(1):27–52, January 2000.
6. M. Bozga, S. Graf, and L. Mounier. Automated Validation of Distributed Software using the IF Environment. In *Workshop on Software Model-Checking*, volume 55. TCS, July 2001.
7. M. Bozga, D. Lesens, and L. Mounier. Model-Checking Ariane-5 Flight Program. In *Proceedings of FMICS'01 (Paris, France)*, pages 211–227. INRIA, 2001.
8. 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, *2nd International Workshop on Protocol Test Systems.* North Holland, 1990.
9. P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. Technical report, LIX, Ecole Polytechnique, may 1990.
10. C. Daws and S. Yovine. Reducing the Number of Clock Variables of Timed Automata. In *Proceedings of RTSS'96 (Washington, DC, USA)*, pages 73–82. IEEE Computer Society Press, December 1996.
11. M. Dwyer. Data Flow Analysis Frameworks for Concurrent Programs. Technical Report UM-CS-1995-062, University of Massachusetts at Amherst, 1995.
12. J.Cl. 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*, 29:123–146, 1997.
13. S. Graf and G. Jia. Verification Experiments on the Mascara Protocol. In *Proceedings of the SPIN'01 Workshop (Toronto, Canada)*, volume 2057 of *LNCS*. Springer, 2001. ISBN 3-540-42124-6.
14. R. Gupta, M.J. Harrold, and M.L. Soffa. Program Slicing-Based Regression Testing Techniques. *Journal of Software Testing, Verification and Reliability*, June 1996.
15. 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 *Proceedings of SAS'99 (Venezia, Italy)*, volume 1694 of *LNCS*. Springer-Verlag, 1999.
16. ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Technical Report 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, 1988.
17. ISO/IEC. Open Systems Interconnection Conformance Testing Methodology and Framework – Part 1: General Concept – Part 2 : Abstract Test Suite Specification – Part 3: The Tree and Tabular Combined Notation (TTCN). Technical Report 9646, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, 1992.
18. ISO/IEC. Information Retrieval, Transfer and Management for OSI; Framework: Formal Methods in Conformance Testing. Technical report, ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8., 1996.

19. ITU-T. Recommendation Q.2110. ATM Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP). Technical Report Q-2110, International Telecommunication Union – Standardization Sector, Genève, 1994.
20. ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union – Standardization Sector, Genève, November 1999.
21. T. Jéron and P. Morel. Test Generation Derived from Model Checking. In N. Halbwachs and D. Peled, editors, *Proceedings of CAV'99 (Trento, Italy)*, volume 1633 of *LNCS*, pages 108–122. Springer, July 1999.
22. A. Kerbrat, T. Jéron, and R. Groz. Methods and Methodology for Incremental Test Generation from SDL. In R. Dssouli, G. Bochmann, and Y. Lahav, editors, *Proceedings of SDL FORUM'99 (Montreal, Canada)*, pages 135–152. Elsevier, June 1999.
23. G. Kildall. An Unified Approach to Global Program Optimization. In *ACM Symposium on Principles of Programming Languages*, 1973.
24. B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(10):155–163, 1988.
25. M. Phalippou. Test Sequence Generation using Estelle or SDL Structure Information. In D. Hogrefe and S. Leue, editors, *Proceedings of FORTE/PSTV'94 (Berne ,Switzerland)*, pages 405–420, October 1994.
26. H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood, John Wiley, 1988.
27. D. Rayner. OSI Conformance Testing. *Computer Networks and ISDN Systems*, 1987.
28. J. Reichwein, G. Rothermel, and M. Burnett. Slicing spreadsheets: An integrated methodologu for spreadsheet testing and debugging. In *Proceedings of the 2nd Conference on Domain Specific languages*, Austin, Texas, October 1999.
29. 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 University of Lübeck, 1998.
30. S .Sinha, M.J. Harrold, and G. Rothermel. System dependance-graph-bases slicing of programs with arbitrary interprocedural control flow. In *ProceedingS of the 21st International Conference on Software Engineering*, Los Angeles, California, May 1999.
31. Real time UML Consortium. Response to the OMG RFP for Schedulability, Performance and Time, v. 1.0. OMG ducument ad/2000-08-04, August 2000.
32. F. Tip. A Survey of Program Slicing Techniques. Technical Report CS-R9438, CWI, Amsterdam, The Netherlands, 1994.
33. J. Tretmans. *A Formal Approcah to Conformance Testing.* PhD thesis, University of Twente, Twente, The Netherlands, 1992.
34. J. Tretmans. A Formal Approach to Conformance Testing. In *6th International Workshop on Protocols Test Systems*, 1994.
35. M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984.