INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The steam boiler controller problem in ESTEREL *and its verification by means of symbolic analysis*

Michel Bourdellès

N° 3285

Octobre 1997

THÈME 1

apport de recherche

ISSN 0249-6399

The steam boiler controller problem in ESTEREL and its verification by means of symbolic analysis

Michel Bourdellès

Thème 1 — Réseaux et systèmes
Projet Meije

Rapport de recherche n° 3285 — Octobre 1997 — 23 pages

Abstract:
We describe the use of the verification tools XEVE and FC2SYMBMIN on the ESTEREL encoding of a Steam Boiler controller proposed by J.R. Abial.

XEVE is a verification tool set dedicated to the analysis of synchronous reactive systems in the form of boolean equations, using the symbolic representation of BDDs for implicit state representation.

FC2SYMBMIN is a latter addition to this toolset, and allows verification after reduction and abstraction with respect to bisimulation of explicit finite state machines with a symbolic treatment (using BDDs again) of input event predicates. This specific representation triggers new algorithmic issues in the computation of bisimulation classes.

We demonstrate the power of FC2SYMBMIN in terms of reduction of states, but also in terms of reduction of transitions for which the gain is often dramatic.

Key-words: Synchronous reactive systems, symbolic bisimulation, verification by observer

(Résumé : tsvp)

Unité de recherche INRIA Sophia Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA ANTIPOLIS Cedex (France)
Téléphone : 04 93 65 77 77 – international : +33 4 93 65 77 77 – Fax : 04 93 65 77 65 – international : +33 4 93 65 77 65

Description en ESTEREL du contrôleur d'une turbine à vapeur et sa vérification par analyse symbolique

Résumé :

Nous décrivons l'utilisation des outils XEVE et FC2SYMBMIN sur la description en ESTEREL du contrôleur d'une turbine à vapeur proposé par J.R. Abrial.

XEVE est un ensemble d'outils de vérification basé sur l'analyse de systèmes réactifs synchrones à partir d'équations booléennes, utilisant une représentation implicite de l'espace d'états par des BDDs.

FC2SYMBMIN s'ajoute à cet ensemble d'outils, et permet la vérification après réduction et abstraction suivant une bisimulation à partir de la représentation explicite des machines d'états finis dans laquelle les prédicats d'événements d'entrée sont traités symboliquement (en utilisant encore des BDDs). Cette représentation amène à modifier l'algorithmique de calcul des classes d'équivalence.

Nous démontrons la puissance de FC2SYMBMIN en terme de réduction du nombre des états mais aussi et surtout de transitions.

Mots-clé : Systèmes réactifs synchrones, bisimulation symbolique, vérification par observateur

Introduction

Synchronous langages such as ESTEREL[4], LUSTRE [5] and SIGNAL [3] are convenient to express the behavior of system controllers, allowing structured programming including parallelism between subcomponents. They are based on Mealy finite state machine interpretation. The ESTEREL compiler can generate low level code based on two alternative representations for the underlying state space.

The first one is an implicit representation which uses registers to store state variables and boolean equation systems to compute the output and next state function. The second one is an explicit representation which enumerates the state space of the Mealy machine. On this explicit representation, called reactive automaton, a transition matches several transitions of the corresponding *classical* Mealy machine by allowing as input part of the label a predicate which represents symbolicaly a set of possible input events (with similar behavioral effects).

Due to parallelism, the reachable state space may grow very large. The ESTEREL verification environment provides two approaches to solve the state space explosion, corresponding to the alternative representations above.

The first one combines the reachable state space (symbolic) computation and verification by observer; it corresponds closely with the first form of ESTEREL compiled code.

The second one consists of a conjunction of compositional reductions and abstraction by signal hiding, and is applied to the second representation. We apply the strong bisimulation on explicit reactive automata (again: Mealy machines with symbolic input predicates). We have defined a new algorithmic on computation classes to adapt this bisimulation on the reactive automaton representation.

These verification techniques are implemented in the software tools XEVE and FC2SYMBMIN devoted respectively to the implicit and explicit finite state machine representation. XEVE and FC2SYMBMIN belong to the FC2TOOLS verification package [2]. XEVE uses BDDs to represent the implicit state space, and also FC2SYMBMIN to represent and operate on input transition label predicates.

We present an ESTEREL encoding of the celebrated Steam-Boiler problem [1] as a non trivial running example. In Section 1, we describe the verification techniques. In Section 2, we introduce the Steam Boiler Specification. In Section 3 we describe the ESTEREL encoding and in Section 4, we check properties with the help of our tools. We conclude in Section 5 with possible future work.

1 Verification techniques description

1.1 Definition of the implicit state space representation

In this model, a state of the finite state machine is defined as a valuation of boolean state variables (called also registers or latches). Two vectors of boolean functions compute the next state variables valuation and the output boolean variables valuation from the current state variables valuation and an input variables valuation. This model need not be aware of the reachable state space of the underlying Mealy finite state machine.

1.2 Verification by observer

The basic idea behind "observers" is to use the reactive paradigm itself for verification by building a composed system from the program and an auxiliary one aimed at checking a given property.

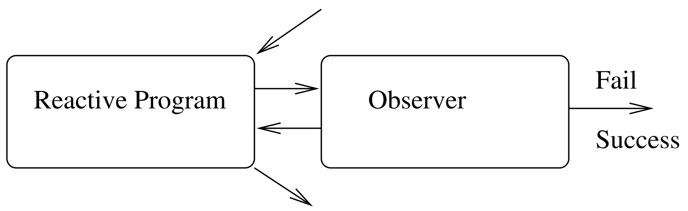

Figure 1: Verification by observer

Observers are reactive programs set in parallel with the program to be analysed, and interacting with it. They may emit additional success/failure signals for diagnostics. Observers may test signals from the program but also simulate the program environment by feeding signals to the program, guiding it to desired configurations. So the observer paradigm can be seen as a generalised "symbolic testing" procedures, where all possible tests will be conducted simultaneously in a reachable state space search.. Several observers can be set in parallel with the same main module (if they don't interfere). Input and output signals from the controller can be made internal to the global program to avoid interference from the potentially remaining environment.

Observers based verification reduces safety problems to reachability of a configuration in the product machine, and thus leads itself quite nicely to implicit representation treatment based on BDD symbolic methods. Some liveness properties can also be verified by detection of infinite sequence of behaviours, (and thus in a finite state system, loops) during which a signal is not eventually emitted.

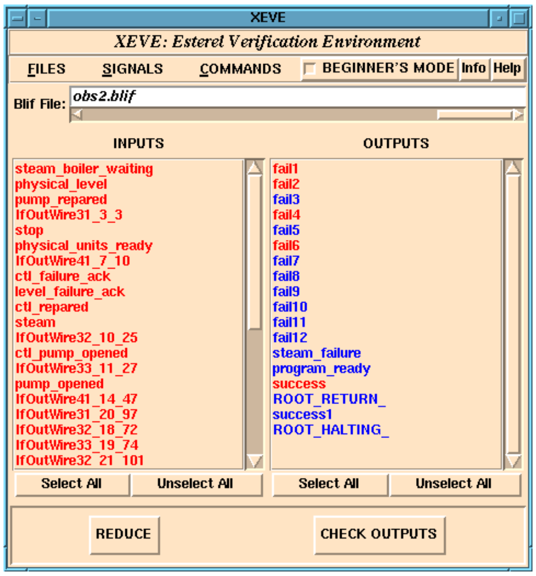

Figure 2: The XEVE graphical interface

Properties are expressed as modules in a programming operational style, and so, this verification technique is homogeneous, opposite to model checking which introduces extra temporal logic syntax. Programs can also be synthetized from specific temporal logics (based on past operators).

1.3 Definition of reactive automata

A Mealy machine $\mathcal{M}$ is defined from a finite state set $\mathcal{Q}$, an initial state i and two alphabets Σ, and Γ (the input and output alphabets), and computes two partial functions, the next state function $\mathcal{N} : \mathcal{Q} \times \Sigma^* \rightarrow \mathcal{Q}$, and the output function $\lambda : \mathcal{Q} \times \Sigma^* \rightarrow \Gamma^*$. For more information, refer to [6].

Synchronous reactive systems are based on deterministic and complete finite state Mealy machines interpretation. That is to say , for all $(q, (\sigma_1, .., \sigma_n)) \in (\mathcal{Q} \times \Sigma^*)$, it exists only one $(q', \gamma_1, .., \gamma_m) \in (\mathcal{Q} \times \Gamma^*)$, such that $\mathcal{N}(q, \sigma_1, .., \sigma_n) = q'$ and $\lambda(q, \sigma_1, .., \sigma_n) = (\gamma_1, .., \gamma_m)$

In our representation of the Mealy machine, called reactive automaton, a transition may stand for several transitions from the *classical* Mealy machine representation.

Definition (Reactive automata)

A *reactive automaton* is a 5-uple :

$$(Q, q_0, \Sigma, \Gamma, Trans)$$

Where

- Q is a finite set of states.
- q_0 is the initial state.
- Σ is a finite set of input signal names.
- Γ is a finite set of output signal names.
- $Trans$ is the transition relation, $Trans \subseteq Q \times Pred(\Sigma) \times Prod(\Gamma) \times Q$ with
  - $Pred(\Sigma)$ is the set of predicates on elements from the set Σ. For convenience we shall only deal with propositional boolean formulas represented as sums of products of positive and negative literals. Negative literals are identified by a # prefix instead the classical bar on top. Symmetrically positive literals are identified by a ? prefix. For instance figure 3 represents transitions from a Mealy machine and the respective transition from a corresponding reactive automaton.
  - $Prod(\Gamma)$ is the set of the subsets of Γ. $Prod(\Gamma)$ comprises the empty set noticed *tau* (no visible action).

  $Trans$ satisfies the following condition: $q \longrightarrow^{pred}_{prod} q' \in Trans \Rightarrow pred \wedge \bigwedge_{o \in prod} ?o \neq false$

A reactive automaton is complete if and only if for all states, the disjonction of input predicates of on outgoing transitions is equal to $True$.
A reactive automaton is deterministic if for all pair of transitions leaving the same state wiyh different output part, the input part of the transitions are disjoint.

Reactive automata generated from ESTEREL encodings are deterministic and complete.

1.4 Reduction

At First, We assume composition of automata reduced and/or with signals hidden stay always acyclic. This can be garantee in a causality analysis of the corresponding modules set in parallel, from an intermediate equational format of ESTEREL.
This verification technique takes advantage of the modular structure of ESTEREL encodings. It consists of the reduction of the reactive automaton associated to each ESTEREL parallel

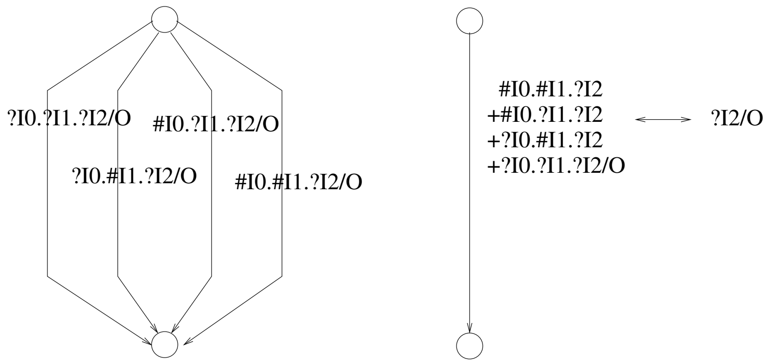

Figure 3: transitions from two states of a Mealy machine and from the corresponding reactive automaton states

submodule with the help of a particular reduction, followed by computation of the synchronous product of minimised automata. We can hide signals not relevant to the property to check. Afterwards, we can either visualise the resulting automaton with the help of the AUTOGRAPH tool, or applying further verification techniques or even observation on as described previously.
The equivalence relation we shall consider will be the strong bisimulation on Mealy machines. Due to the symbolic input parts on input signals, the definition becomes a bit more involved when applied to reactive automata: a transition recovers now a set of possible behaviours, and the behavioural matching may preserve sets of transitions.

Definition of the symbolic bisimulation relation [9]

Let $R = (\Sigma, \Gamma, Q, q_0, Trans)$ be a reactive automaton.

The symbolic bisimulation relation $\sim$ is defined on $Q \times Q$ like below :
$p \sim q$ if and only if

$\forall pred \in Pred(\Sigma)\ \forall prod \in Prod(\Gamma)\ \forall p' \in\ Q$, such as $p \longrightarrow^{pred}_{prod} p'$
$\exists q'_1 \in Q, ..., q'_n \in Q \exists pred1 \in Pred(\Sigma), ..., predn \in Pred(\Sigma)$, such as $q \longrightarrow^{predi}_{prod} q'_i$
Avec $\forall i, p' \sim q'_i$ et $Pred \Rightarrow \bigcup predi$

and $q \sim p$

In essence $p \sim q$ if and only if they both produce identical outputs leading to equivalent states under the same input predicates, possibly divided differently along subcases. We make this formal in the following lemma.

notation
Let $UnionIn(p, prod, p') = \bigcup pred_i$ such that$\exists p_i, p \longrightarrow^{pred_i}_{prod} p_i \; and \; p_i \sim p'$

Lemma
States p and q are bisimilar if and only if,

$\forall r \; \forall prod \in Prod(\Gamma), \; , UnionIn(p, prod, r) = UnionIn(q, prod, r).$

proof
The lemma can be deduced from the definition with the fact that symmetric implication certifies that, assume $p \sim q$, one product given, all the predicates of transitions from p (resp q) can be matched with union of predicates of transitions from q (resp p) to states bisimilar.

Informally, two states are considered as equivalent for this bisimulation if they access to equivalent states in emitting the same output signals for the same combination of input signals. In the example of Figure 4, states q_1 and q_2 are bisimilar iff q_1', q_1''' and q_2' are bisimilar, and also q_1'' and q_2''.
Important to note, bisimulation now needs to match several transitions to several others, instead of matching transitions pairwise.

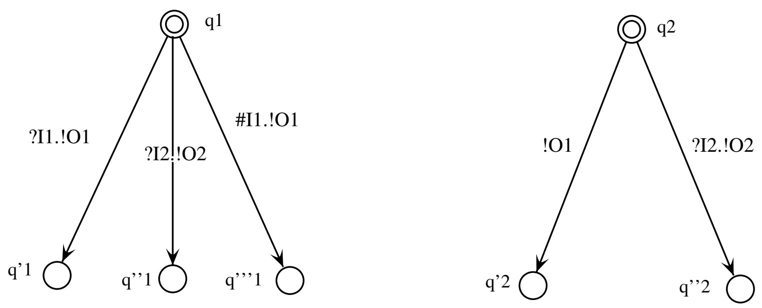

Figure 4: states equivalent for a special bisimulation relation

Partitioning algorithm

We exploit the lemma's result to propose an algorithmic adapted from the Kannelakis & Smolka partition refinement algorithm [7]. Consider blocks B and B' (we call block a class of states for an intermediate refinement equivalence) and a set of output signals *prod*. We split B by B' in computing for each state of the class B the union of all the input predicates from all the transitions emitting *prod* whose target states belong to the class B'. States with identical unions stay paired.
B can thus be divided in more than two sub-classes. Note that this coarsest partition refinement process does not require a choice among possible matching subsets of transitions, unlike forward on-the-fly (or local) methods would, leading to combinatorial explosion.

We have implemented the symbolic bisimulation algorithm in a tool called fc2symbmin In addition, FC2SYMBMIN computes also the product of automata, hiding of signals and the verification by observer. The product of automata implements the ESTEREL synchronous parallel operator. The hiding of signals reduces the size of behavior label. To hide input signals can create non-determinism, and is used to abstract an automata on signals interacting to a property in a verification by observer. Compositional linking is under way. We shall exam verification on the Steam Boiler in section 4. The UNIX command attached to the FC2SYMBMIN processor is detailed in Annex 1.

Experimental results

We present results obtained using symbolic bisimulation minimisation with the FC2SYMBMIN processor on a selection of automata produced from ESTEREL programs together with computing time on an Ultra Sparc, 320 Mbytes memory size.

Table 1 provides figures before reduction, and table 2 the corresponding number after reduction. Memory size is in bytes.

controller name	*initial automaton*		
	# nodes	*#transitions*	*mem. size*
Steam Boiler	1298	188515	8982788
One_Pump(v2)	139	3212	123424
One_Pump(v1)	70	2484	102058
Car	162	2968	146979
master	18	86	3795
lift	9	29	1576
submarine1	116	871	35932
submarine2	74	1926	87776
arbiter12	14	170	8163

Table 1: Size of few reactive automata

controller name	*reduced automaton*			
	# nodes	*#transitions*	*mem. size*	*time*
Steam Boiler	139	2346	242203	536 sec
One_Pump(v2)	65	299	10783	3 sec
One_Pump(v1)	11	46	2706	2 sec
Car	29	143	11113	2.5 sec
master	13	35	1882	0.3 sec
lift	9	19	1601	0.25 sec
submarine1	116	524	23143	4 sec
submarine2	64	372	21142	3 sec
arbiter12	13	13	6097	0.4 sec

Table 2: Results about the minimisation of ESTEREL controller encodings

The reduction in the number of states and even more in the number of the transitions is spectacular.

2 The case study description

This case study reports the behaviour of a Steam Boiler specification as described in [1]. It is composed of several elements: the steam-boiler, a device to measure the quantity of water, four pumps, four devices to supervise the pumps and a device to control the quantity of steam.
Five modes are available : Initial, Normal, Rescue, Degraded and Emergency.

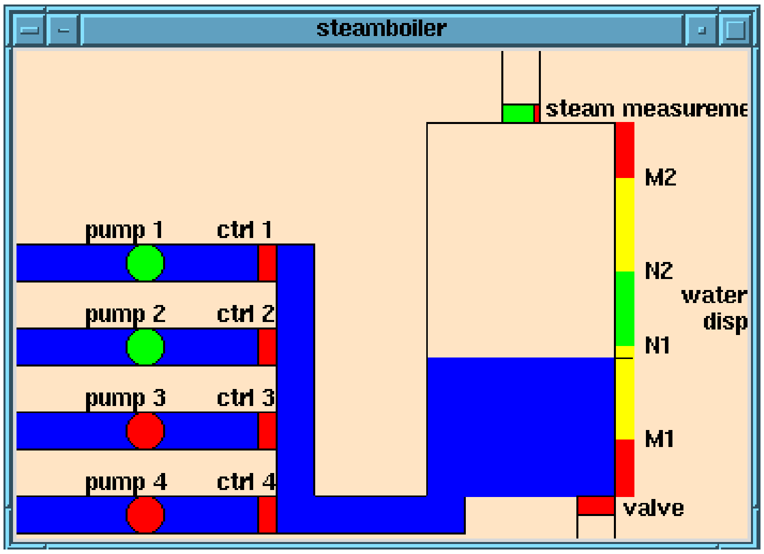

Figure 5: The FZI graphical interface linked with the controller

3 ESTEREL encoding of a Steam Boiler Specification

The ESTEREL modular architechture follows from the specification. It comprises :

1. A main **Steamboiler** module described below.
2. Modules **Normal**, **Initialization**, **Rescue** and **Degraded**; each implementing the corresponding mode.
   - Module **Initialisation** opens and closes pumps and valve until the water level becomes correct then we access the normal or degraded mode.
   - Modules **Normal**, **Degraded**, **Rescue** open or close pumps in response to the water level.
3. In the module **Choice_mode**, pump failure, pump controller failur or water level failure detection changes the mode.
4. The module **Level_management** controls the water level. It tests if the water level value agrees with the expected value.
5. module **Pump_manager** controls the pumps. It receives signals open/close(n) pumps from the mode modules and send open/close(m) pumps to the physical systems. m can be different from n because pumps can already be opened closed. It detects pump failure and pump controller failure, so as not to use the physical units until it is repared.

The main module is presented below while others can be found in Annex 2.

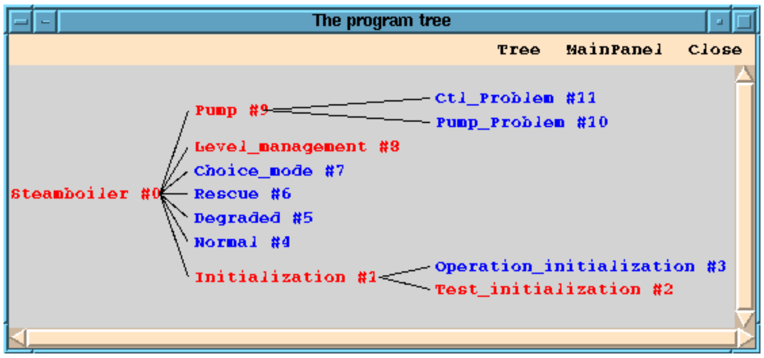

Figure 6: Modular architecture decomposition (viewed in XES, a simulation tool)

main module

```
module : Steamboiler
```

Declaration part

```
trap emergency_stop in
   await immediate steam_boiler_waiting;
   run Initialization ;
   loop
      present normal then
      run Normal
      else present degraded then
              run Degraded
            else present rescue then
                     run Rescue
                 else
                    exit emergency_stop
                 end present
            end present
      end present;
    await tick;
   run Choice_mode
   end loop

||
   await stop;
   exit emergency_stop
||
```

```
   run Level_management;
   exit emergency_stop
||
   run Pump
end trap %emergency_stop
```

4 Properties verification

4.1 Properties as Observers

We shall present a set of properties. Each property is introduced as informal specification in english then realized as an ESTEREL fragment. Properties are of the following kinds:

- Properties which certify coherence of the modes settings (**properties 1, 2**).
- Properties which certify coherence between failures and modes setting (**properties 3,5**).
- Property which certifies a correct behaviour of the valve (**property 4**).
- Property which certifies signals open_pump and close_pump can't be emitted at the same time (**property 6**).

property 1
Verification that if the controller is in normal or degraded or rescue mode, either it is still in one of these mode or the steam boiler stops in one of the possibility among emergency mode, or transmission failure or a stop message received

```
weak abort
      await [(normal or degraded or rescue)];
      await [not (normal or degraded or rescue)];
      emit fail1;
when immediate [stop or emergency or transmission_failure]
```

property 2
The system may be in one and only one mode

```
 every [(normal and (degraded or rescue))
           or (degraded and rescue)] do
 emit fail2
 end
```

property 3
The system can't be in normal mode in case of level_failure signal emission

```
every normal do
   present level_failure then emit fail4
   end present
end
```

property 4

The system can't be in normal, degraded or rescue mode with the valve opened

```
weak abort
 loop
   await valve; %ouverte
   abort
     await [normal or degraded or rescue];
     emit fail5;
   when valve %fermée
 end loop
when [normal or degraded or rescue or emergency]
```

property 5

In case of level_failure and not (steam_failure or pump_failure or ctl_failure) when the system is in normal mode, the system goes in rescue mode

```
weak abort
   loop
    await normal;
    await tick;
    present [ level_failure
                    and not (steam_failure or pump_failure or ctl_failure)] then
    present rescue then emit success else emit failure
    end
    end
    end
when [stop or emergency]
```

property 6

messages open_pump and close_pump can't be send to the physical unit in the same instant

```
every open_pump do
   present close_pump then emit fail8
   end
end
```

4.2 Verification from an implicit representation

These observer programs are put in parallel with the Steam_boiler program. Interface signals connecting them are declared as local. We checked the properties with a 2 pumps model from an implicit representation.

4.3 Verification by reduction and hiding

The verification techniques from an explicit representation of the ESTEREL encoding is leading to build from the global automaton an abstraction on which information about the property is kept. The abstracted automaton is obtained with application of minimisation on the global reactive automaton and in hiding signals which don't interfere to the verification of the property. Two methods can then be applied to check the property. The first one is to apply the verification by observer on this abstracted automaton. The second one consists to check the property in visualisation of the abstracted automaton with the AUTOGRAPH tool.

The six properties described in Subsection 4.1 have been verified with a one pump model from an explicit representation where irrelevant signals were hidden away. For each property we record in table 4 the size of the abstracted automaton. Memory size is in bytes.

automaton	*#nodes*	*#transitions*	*mem. size*
initial automaton	1298	188515	8982788
reduced automaton	139	2346	242203

Table 3: Recalling: size of the automaton and of its minimisation

Figure 7 shows the automaton obtained from the ESTEREL Steam Boiler encoding, after reduction and hiding of all signals safe **valve**, **degraded**, **normal** and **rescue**. Weak bisimulation [2] has been applied on the automaton of table 4 (property 4), to reduce *tau* transitions.
A node is in degraded, normal or rescue mode if it is the target of a transition including emission of signal degraded, normal or rescue. We just kept visible states that can access to states from which signals **degraded**, **normal** or **rescue** are emitted. `tau` expresses no action.

We check the signal valve is emitted an even number of times before beeing in normal, rescue or degraded mode.

Indeed, we can observe that an even number of emission of the valve is necessary to acess to vertice q_0, q_1, q_2, q_3, q_4 and q_7, and an odd number of emission of the valve is necessary to acess to vertice q_5 and q_6.

5 Conclusion

The FC2TOOLS XEVE and FC2SYMBMIN dedicated to ESTEREL verification can be succesfully used to verify properties from the Steam Boiler control system case study [1]. The imperative, structured and modular architecture of ESTEREL encodings is practical to write the specification. We have interfaced the program with the XES ESTEREL simulation tool. XES is helpful in playing action pathes recorded with XEVE if a special observer output signal has been emitted.
We are working on a syncCharts description of the case study. We extend the analysis techniques to the study of programs with data (counters, valued inputs and variables), foremost the reachable state space computation from an implicit representation of the finite state machine.

property	*External signals set visible*	*abstracted automaton*		
		#nodes	*#transitions*	*mem. size*
property 1	**normal degraded rescue emergency stop transmission_failure**	20	108	3344
property 2	**normal degraded rescue emergency**	16	73	2091
property 3	**normal level_failure**	29	194	4746
property 4	**normal degraded rescue valve**	23	122	3413
property 5	**normal rescue level_failure steam_failure pump_failure ctl_failure stop emergency**	113	1821	35248
property 6	**open_pump close_pump**	17	83	2631

Table 4: Results about the abstraction of ESTEREL encodings

References

[1] J-R. Abrial, *Steam-boiler control specification problem*, August, 10, 1994 *Formal Methods for Industrial Applications*, LNCS 1165

[2] A. Bouali, A. Ressouche, V. Roy and R. de Simone *The* FCTOOLS *User Manual (Version 1.0) Technical Report* INRIA, *n° 0191, April 1996*
See also the FC2TOOLS page at: http://www.inria.fr/meije/verification/

[3] A. Benveniste and P. LeGuernic, *Hybrid dynamical systems theory and the* SIGNAL *lanngage. IEEE Transactions of Automatic Control, 35(5):535-546, May 1990*

[4] G. Berry, G. Gonthier, *The* ESTEREL *Synchronous Programming Langage: Design,Semantics,Implementation. Science of Computer Programming vol. 19, n° 2, pp 87-152, 1992*
See also the ESTEREL web page at: http://www.inria.fr/meije/esterel/

[5] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud *The synchronous dataflow programming langage* LUSTRE. *Proceedings of the IEEE, 79(9):1305-1320, September 1991. Science of Computer Programming vol. 19, n° 2, pp 87-152, 1992*

[6] S. Eilenberg, *Automata, langages and Machines*, volume A, Editors *P. A Smith and S. Eilenberg*, Collection *Pure and Applied Mathematics*, ACADEMIC PRESS, 1974.

[7] P.C. Kanellakis and S.A. Smolka *CCS expressions, finite state processes and Three problems of equivalence*, *Information and Computation*, 86(1) May 1990.

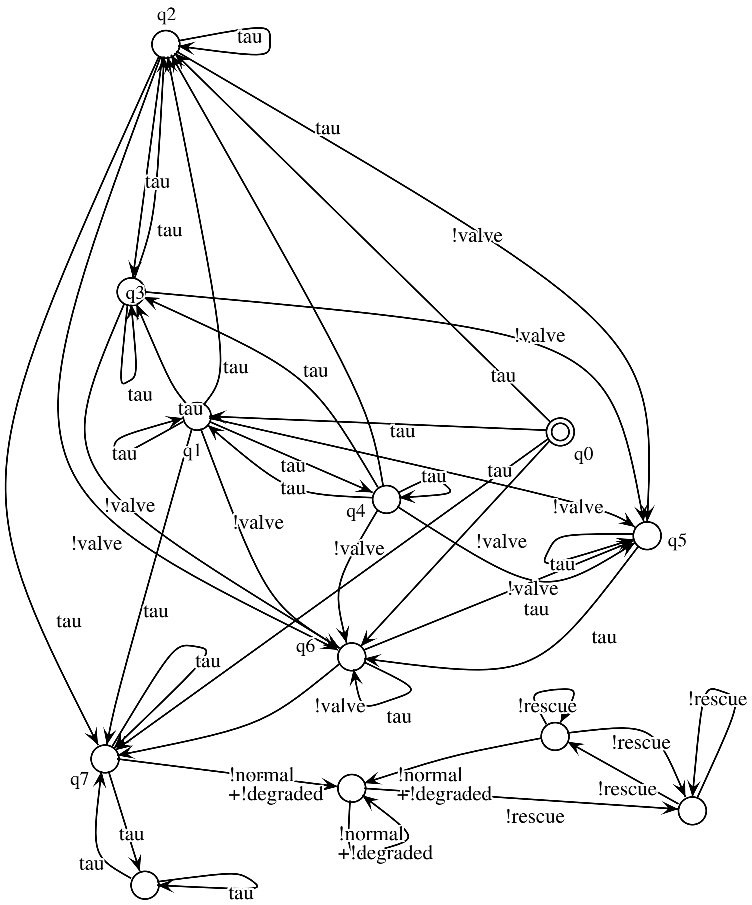

Figure 7: View from the AUTOGRAPH graphical editor

[8] R. Paige and R.E. Tarjan.*Three partition refiniment algorithm*, *SIAM journal of computation*, 973-989, 1987.

[9] R. de Simone and A. Ressouche, *Compositional Semantics of Esterel and Verification by Compositional Reductions*, *Computer Aided Verification*. Lecture Notes In Computer Science 818, pp 441-454, 1994.

Annex 1: Unix Command Line Description of FC2SYMBMIN

- `fc2symbmin`

SYNOPSIS: fc2symbmin analyses Mealy FSM under the form of fc2 automata produced from ESTEREL. typical analysis methods comprises minimisation, observation by ide programs, product, hiding and equivalence checking.

The minimisation is relative to bisimulation of reactive automata obtained from synchronous language ESTEREL. In this bisimulation the input event of the transition label is treated symbolically as a propositional formula (encoded as BDDs). For instance the two Esterel program fragments **`present I then emit O else emit O end present; halt`** and **`emit O; halt`** are equated.

Observers are programs set in parallel with the main program which interacts with it and may emit additional success/failure signals to indicate diagnostics. Observers may test signals from the main pogram but also send it signals to simmulate partly the ESTEREL environment.

The product of automata implements the ESTEREL synchronous parallel operator (disregarding causality issues) at the level of fc2 automata.

The hiding of signals reduces size of behavior label. To hide input signals can create non-determinism.

The equivalence checking computes if two automata can behave not the same with the same environment. It stops as possibly if this is assumed and produces a counter-example.

Fc2symbmin accepts as input fc2 files obtained from the intermediate formats `oc` and `blif` by the respective `ocfc2` and `bliffc2` processors.

USAGE: `fc2symbmin` *`options file[.fc2]`*

Minimisation :

`fc2symbmin -[il|ib] -[ob|n]` *`file[.fc2]`*

FORMAT OF *file.fc2* :

`fc2symbmin` accepts two alternative fc2 representations.

`-[il|ib]`:

- `-il` Here information on input signals is stored in a logical part of the transition label. It is obtained with the `ocfc2` process with the `-logic` option from the `oc` file deduced from the ESTEREL or LUSTRE encoding.
- `-ib`: Information on input signals is represented "*as classically*" in the `behav` field of the transitions.

RESULTS :

`-[ob|n]`:

- `-ob`: Information on the input signals is represented in the `behav` field of the transitions.
- `-n`: No automaton is printed.

In all the cases, the size of the reduced automaton is printed.

Product :

`fc2symbmin -prod` ***file[.fc2] file2[.fc2]***

Returns the automaton product of automata *file[.fc2]* and *file2[.fc2]*.

Verification by observer :

`fc2symbmin -[obs|[[may|must]` ***OutputObsSigname***`]` ***file[.fc2] obs[.fc2]***

`-obs` assumes the property's output signals from *obs[.fc2]*, may, must be emitted if *obs[.fc2]* interacts with *file[.fc2]*. The following restrictions are available.

`-may` ***OutputObsSigname***

Restricts attention to the property "may emit *OutputObsSigname*".

`-must` ***OutputObsSigname***

Restricts attention to the property "must emit *OutputObsSigname*".

Hiding :

`fc2symbmin [-ol] -[hide|see]` ***sigName1 ... sigNameN file[.fc2]***

`-hide` s_1 ... s_n

Returns the automaton with s_1 ... s_n hidden.

`-see` s_1 ... s_n

Returns the automaton with signals other than s_1 ... s_n hidden.

`-ol` In the automaton obtained, information concerning input signals is stored in a logical part of the transition label.

Equivalence checking :

`fc2symbmin -eq [-debug]` ***file[.fc2] file2[.fc2]***

Checks *file[.fc2]* and *file2[.fc2]* behave equivalent.

`-debug` If *file.fc2* and *file2.fc2* are not equivalent, returns a path going to a state where the two automata without equivalent in the other automaton.

DEFAULT : Options by default are `-ib -ob`

MORE INFORMATION :

Annexe 2: ESTEREL Steam boiler control system encoding

main module

Declaration part

```
trap emergency_stop in
   await immediate steam_boiler_waiting;
   run Initialization ;
   loop
      present normal then
      run Normal
      else present degraded then
              run Degraded
            else present rescue then
                     run Rescue
                 else
                    exit emergency_stop % eq to run Emergency module;
                 end present
            end present
      end present;
    await tick;
   run Choice_mode
   end loop

||
   await stop;
   exit emergency_stop
||
   run Level_management;
   exit emergency_stop
||
   run Pump
end trap %emergency_stop
```

Choice_mode module

Declaration part

```
present ok_level then
   present pb_pump  then
      emit degraded
   else
      emit normal
   end present;
else
   present pb_pump  then
```

```
      emit emergency
   else
      emit rescue
   end present;
end present;
```

Initialisation module

Declaration part

```
   run Test_initialization
||
   run Operation_initialization
```

Normal mode module

Declaration part

```
present level then
   if SUP(?level,N2) then
      emit close_pump;
      emit nb_close_2;
   else
      if INF(?level,N1) then
 emit open_pump;
         emit nb_open_2;
      end if
   end if
end present;
```

One Pump controler module

Declaration part

```
loop
   % check pump and controller closed until open_pump_command included
   weak abort
      every immediate [pump_opened] do
         emit pump_failure_found;
         await immediate [not problem];
         emit close_pump_command
      end
   ||
      every immediate [ctl_pump_opened] do
         emit ctl_failure_found;
         await immediate [not problem];
         emit open_pump_command
```

```
      end
   when [open_pump_command and not problem];
   emit open_pump;

   % check pump and controller opened until close_pump_command included
   weak abort
      every [not pump_opened] do
         emit pump_failure_found;
         await immediate [not problem];
         emit close_pump_command
      end
   ||
      await tick;
      every [not ctl_pump_opened] do
         emit ctl_failure_found;
         await immediate [not problem];
         emit close_pump_command
      end
   when [close_pump_command and not problem];
   emit close_pump;
   await tick;
end loop
||
loop
   await immediate pump_failure_found;
   %present [not ctl_ok] then
   run  Pump_Problem / Problem [ signal pump_failure / failure,
                                       pump_failure_ack / failure_ack,
                                       pump_repared / repared,
                                       ctl_ok / pb_not_real,
                                       no_pump_pb / no_problem ];
 %  end present;
   await tick;
end loop
||
loop
   await immediate ctl_failure_found;
   %present [not pump_ok] then
   run  Ctl_Problem / Problem [ signal ctl_failure / failure,
                                      ctl_failure_ack / failure_ack,
                                      ctl_repared / repared,
                                      pump_ok / pb_not_real,
                                      no_ctl_pb / no_problem ];
   % end present;
   await tick;
end loop
```

```
||
every immediate [pump_failure_found or ctl_failure_found] do
   abort
      await tick;
      sustain ctl_ok;
   when no_ctl_pb
   ||
   abort
      await tick;
      sustain pump_ok;
   when no_pump_pb
   end

module Problem :
```

Declaration part

```
trap problem in
   abort
      sustain problem
   when no_problem
||
   emit failure;
   await failure_ack;
   await repared;
   emit no_problem;
||
   await pb_not_real;
   exit problem
end trap
end module
```

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

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