Generation of safe operation sequences using iterative refinements and abstractions of timed automata

Thomas Cochard, David Gouyon, Jean-François Pétin

Generation of safe operation sequences using iterative refinements and abstractions of timed automata

Thomas Cochard*,†, David Gouyon*,† and Jean-François Pétin*,†
* Université de Lorraine, CRAN, UMR 7039,
Campus Sciences, BP 70239
54506 Vandœuvre-lès-Nancy cedex, France
† CNRS, CRAN, UMR 7039, France

***Abstract*—The main objective of operation procedure engineering for complex and critical systems is to provide action sequences satisfying safety requirements specifications. A classical limit of the use of formal generation approaches for this purpose is the combinatorial explosion due to the size and the number of required models. This article addresses this issue by proposing an iterative approach for the generation of safe operation sequences, using timed automata, and based on reachability analysis. The originality of this approach is to combine a bottom-up framework to build progressively system models by abstraction, and a top-down iterative action sequence generation.**

***Index Terms*—Action sequences generation, Reachability analysis, Iterative approach, Model checking, Timed automata, Iterative refinement.**

I. Introduction

Industrial processes are complex systems to control, involving field devices (transmitters and actuators), that may be manually controlled and monitored by human field or control room operators, automated reflex control system, and plant operation control. In the case of safety critical systems, to operate while ensuring the satisfaction of safety requirements, operation is often based on predefined and qualified procedures. Operating procedures are ordered sequences of control and monitoring tasks, aiming to modify the process state and to control its evolution. They are mainly implemented in Supervisory Control And Data Acquisition (SCADA) systems and/or Manufacturing Execution Systems (MES).

The qualification of procedure needs the demonstration of safety requirements satisfaction. This may be aided using formal generation or verification processes. Focusing on action sequences, which are part of operating procedures, [1] showed, on the one hand, the feasibility of an automatic approach for the generation of safe sequences based on reachability analysis, but, one the other hand, put also in evidence some limitations related to scalability, mainly due to combinatorial explosion. The considered action sequences consist in sets of ordered actions. These actions may be performed manually by an operator (locally by a field operator or remotely by a control room operator) or automatically by control devices. They induce a change in the state or the status of a device. This brings an evolution on the process physical variables, leading the system from an initial situation to a goal situation. A situation is characterized by the state (such as functioning features) and status (such as availability) of the system components and by a set of physical values.

This article addresses the scalability issue highlighted in [1]. It proposes an iterative modelling and generation approach of safe action sequences. This approach is based on:

- a bottom-up iterative system modelling framework, which architecture is based on ISA88 standard [2] hierarchical levels, and in which the behaviour of system elements is formalised using timed automata [3]
- a top-down iterative action sequence generation, based on a ”two-by-two” decomposition with detailed and abstract models, using reachability analysis techniques (supported by model-checking tools) to generate execution traces that model possible action sequences.

This article is organized as follows. Section II

presents existing approaches for procedure modelling and generation. Section III presents the iterative approach proposed for system modelling and action sequence generation, which is applied on a lab case study in section IV.

II. Existing approaches for iterative procedure modelling and generation

Complex systems control has been formalized in ISA88 [2] and IEC 61512 [4] standards, and combines two sides: the first one considers automated control remotely operated from a control room, and the second one considers manual control, locally operated on the process *via* human actions. If a critical process is considered, the execution of such actions primarily depends on the qualification of operating procedures.

In the operational domain, a procedure is composed of two main types of operations: observations, to verify that operator process view is consistent with the process actual state, and actions, to operate on devices in order to control the global process. The field of procedure modelling [5], [6], generation [7] and verification [8] is an active research domain since the work of J. Rivas in [9] which highlighted these problems in process control.

Among the languages used in operation modelling and synthesis ([10], [5], [7], [11]), timed automata, introduced in [3], have been chosen by [8] and [1] for their formal definition. Timed automata can be defined by a 9-uplet $A = (S, V, X, I, L, T, S_m, s_0, v_0)$ such that:

- S is a finite set of locations;
- V is a finite set of variables;
- X is a finite set of clocks;
- I is a mapping that label each location $s \in S$ with some clock constraints
- L is a set of events, decomposed in three disjoints subsets L_i, L_o and L_l, where:
  - L_i is the set of reception labels;
  - L_o is the set or emission labels.
  - L_l is the set or local labels.
- T is a set of transitions $(s, l, g, m, s') \in S \times L \times G \times M \times S$, where:
  - G is a set of guards (constraints on the clocks of X and the variables of V);
  - M is the set of the updates on the valuation of variables and clocks.
- $S_m \subseteq S$ is a set of marked locations;
- $s_0 \in S$ is the initial location;
- v_0 represents the initial values of variables of V.

Using timed automata, [1] recently proposed to evaluate the feasibility of an approach to automatically generate safe *action sequences*. An action sequence, included in a procedure, is a set of actions to perform on the process, whether automatic controls or manual operations by field agents. The proposed approach was based, on the one hand, on multilevel models using timed automata (the operated system is then considered with three hierarchical levels: Device, Function, Recipe), and, on the other hand, on model checking to analyse the reachability of a target state. Even if the feasibility has been shown, the main problem which has been highlighted is the combinatorial explosion [12].

A first possible way to reduce the size of the explored state space is to reduce the size of the considered models. The use of abstraction techniques [13] seems to be efficient to this end [14]. One technique is based on data abstraction, using a mapping function from one set of variable to another [15]. In other works, [16] proposed a methodology applied to Markov chains to reduce state space, by generating partial Markov chains from high level implicit descriptions, while assessing that the lost of precision relies on acceptable approximations.

A second possible way is to reduce the number of considered models, by using a structured approach. As an example, [17] proposed an iterative approach for the modular control synthesis and implementation. This approach combines the iterative way of thinking proposed by the automation object-oriented methods, and the modular formal synthesis techniques to obtain iteratively hierarchical controllers. The criteria used to structure the control system were based on the structure of the physical process itself, leading to a bottom-up design starting from the field device models.

III. Proposition of an iterative approach for system modelling and action sequence generation

A. Overview of the approach

As presented in section II, the feasibility of an approach based on timed automata and reachability analysis to generate action sequences has already been shown, and some limits were highlighted, mainly regarding the combinatorial explosion problem.

In this article, the limits due to the combinatorial explosion are addressed. The main originality of the proposed approach is the combined use of a bottom-up modelling framework, based on hierarchically organized models of the system, and a top-down

iterative action sequence generation, based on a "two-by-two" decomposition with detailed and abstract models (Figure 1).

The entry points, which are highlighted with a black triangle in the lower right corner in Figure 1, are threefold, and detailed later in this section[1]:

- Model of a sequencer at the higher hierarchical level;
- Detailed models of the system elements at every hierarchical level;
- Model of the system elements at the lower hierarchical level.

The main principle of the proposed approach is the following: at the various hierarchical levels, system models are influenced by a sequencer automata, which models the authorized behaviour of the system. First, the model checking is performed on these models at the higher hierachical level, to check the reachability of a goal situation. If the goal situation can be reached, an execution trace is then generated, corresponding to an admissible actions sequence. This actions sequence is used to automatically build a new sequencer automata. This sequencer, used as a refined model of the authorized behaviour, influences system models of lower hierarchical level, and enables the generation of a refined sequencer. This is done iteratively from the higher hierarchical level, down to the lower level, which corresponds to the device level.

From the entry points and using the proposed approach, all other sequencers and abstract models can be automatically deduced, as explained in the next sections.

B. Bottom-up system modelling using timed automata

In this article, the considered hierarchical structure of the system is similar to the one used in [1], and limited to three hierarchical levels which are linked by causal relations:

- Level 1 - Devices: the lower hierarchical level, consists mostly in valves and pumps, which are elements on which whether operator or control can operate to change state;
- Level 2 - Functions: from a functional point of view, devices are grouped into "functions" enabling the evolution of physical variables. The configuration of a function depends on the state of devices;
- Level 3 - Recipe: the higher hierarchical level, which describes the set of possible situations, is specific to each process. The evolution of the values of physical variables, which characterize the situations of the system recipe, depends on the configuration of functions.

In order to guide system modelling, generic models are proposed in this article. Based on the actinomy [18] *To Prepare*, *To Do* and *To Close*, a sequence patterns is proposed: each sequence starts with a *Request* reception, and ends with a *Response* emission. The core of the sequence is more or less detailed, depending on the type of model, and on its hierarchical level.

1) Lower level models: Devices, at the lower hierarchical level, are mostly valves and pumps. They are the elements on which an operator or the control operates to change its state. Each device is characterized by a couple (*State, Status*) defined as:

- *State* characterises a set of discrete values to represent a device state (e.g. for a valve: open/closed),
- *Status* characterises a device operational configuration (e.g. padlocked, condemned...). The *Status* can be represented by a discrete variable which value is to be fixed at the initialisation of the model (in v_0), and will not change during the state space exploration. This hypothesis is realistic because a constraint of the approach is to propose an action sequence to reach a given situation while taking into account the current devices *Status*.

An example of a generic device behaviour is given in Figure 2, concerning more especially valves. In this model, the locations on the left and the right expresses that the device is in a stable state: for valves, it corresponds respectively to closed (corresponding to *state=0*), and open (corresponding to *state=1*). Two sequences enable to go from one of these location to another:

- starting with the reception of a request (*Open?* or *Close?*), conditioned by safety and status constraints (guard *[!Contraints]*);
- changing the state of the device (*state=1* or *state=0*);

[1]In the timed automata models presented in this article, the following graphical conventions are used: location names are given in bold, initial location is given by a transition with no source, guards on transitions or state invariants are between brackets [], events are in italic and followed by "!" or "?", to represent respectively emission or reception, events sets are between braces {}, and variables updates are underlined.

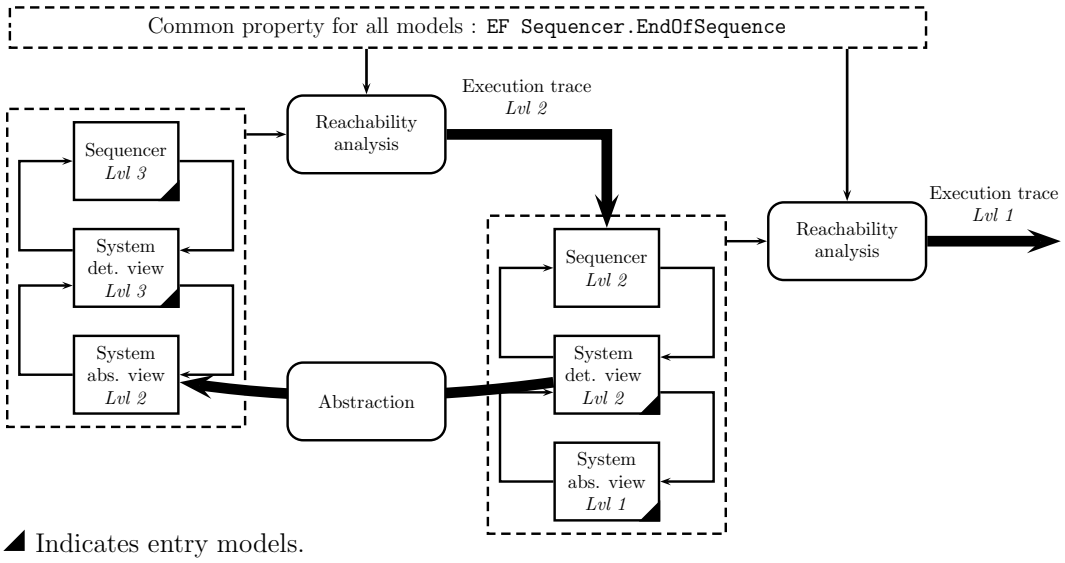

Figure 1. Overview of the action sequences generation approach

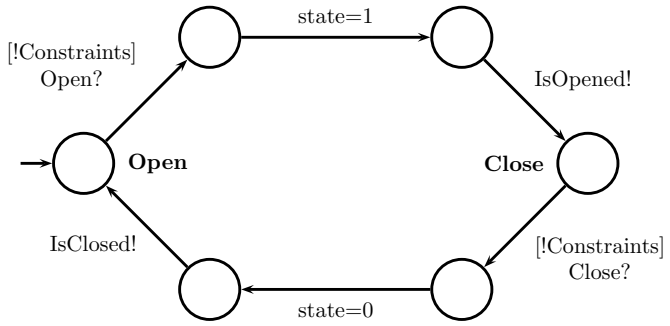

Figure 2. Generic model of a device (valve)

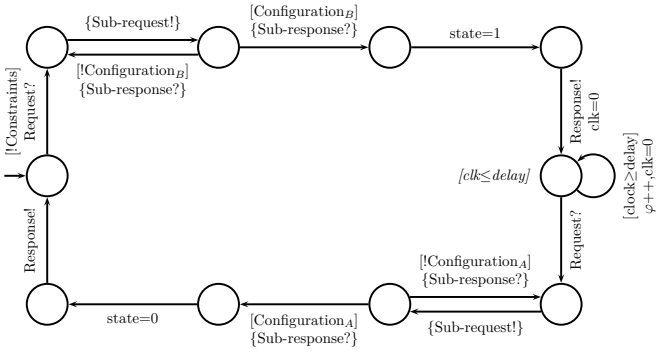

Figure 3. Detailed generic model proposed

- ending with the emission of a report (*IsOpen!* or *IsClosed!*).

It is assumed that, from a temporal point of view, the time needed to operate a device can be neglected in comparison to process timing constraints. This hypothesis is realistic because a device operation lasts only some seconds, while process evolutions often need some hours.

2) Function detailed models: In this section, a generic model of functions is detailed (Figure 3).

Initially, the function is in a configuration in which no physical variable can evolve, and is waiting for a *Request* from the sequencer. The reception of the request, conditioned by constraints to be fulfilled (such as mutual exclusion between functions), is the beginning of a sequence. The core of the sequence changes the configuration of the function. It consists in a set of possible interactions with device models *via* a set of *sub-requests* and a set of *sub-responses* with lower level models (devices models). Each sub-response reception is coupled with the evaluation of the configuration reached *via* the action performed on a device. Once the function is in the required configuration, a *state* variable is then updated, and the function model close the sequence by reporting on the reached configuration *via* the emission of a *Response* event.

When a function is configured, some physical variables may evolve, in function of time. In the model, their evolution (φ++) is dependant on clock values (*clock*$\geq$*delay*), used both to model the dynamics of the

process, and to synchronise all evolutions (in the case where multiple functions are configured).

To come back to the initial configuration of a function, a similar sequence is modelled, starting by the reception of a request, exchanging with lower level models, and, once the initial configuration is reached, ending by the emission of a report.

3) Recipe model: In recipe models, locations represents the possible situations of the process. Transitions between locations are bind to a guard modelling a set of constraints on physical variables and on devices status which characterizes the situation reached by the transition. As recipes are specific to each considered system, it is difficult to propose a generic recipe model. An example is proposed in the case study presented in section IV.

4) Models abstraction : As this article proposes a top-down iterative approach of action sequence generation, only a sub-set of all events is considered at each level. For this reason, it is possible to build an "abstraction" of a "detailed" model by a projection which keeps only the events shared at the considered level. The advantage is that it reduces the size of the explored state space during the reachability analysis. For example, it is possible to generate at the recipe level some sequences of functions without considering the events in relation with the devices. These events will be in fact included in the device action sequence generation.

Let us consider the generic function model of Figure 3. *Request* and *Response* are two events of the same level of abstraction as the system model, while *Sub-request* and *Sub-response* interact with the lower level models. The principle is to project this detailed model to remove all the event which does not belong to the alphabets shared with the sequencer.

Considering two sets L^N and L^{N-1} representing the labels interacting respectively with levels N and $N-1$, a projection operator $\mathbb{P}_{L^a \to L^b}$ is defined, such that:

$$\begin{gathered} \mathbb{P}_{L^a \to L^b}(\epsilon) = \epsilon \\ \forall\ l \in L^a,\ \mathbb{P}_{L^a \to L^b}(\sigma) = \begin{cases} l\ if\ l \in L^b \\ \epsilon\ else \end{cases} \\ \forall\ l_a \in L^a, l_b \in L^b,\ \mathbb{P}_{L^a \to L^b}(l_a l_b) = \mathbb{P}_{L^a \to L^b}(l_a)\mathbb{P}_{L^a \to L^b}(l_b) \end{gathered} \tag{1}$$

In the resulting timed automata models, transitions with ϵ are then suppressed, and both source and sink locations of the transition are merged. The other transitions are kept, with the associated guards, events and updates. The result of the projection of the generic detailed model of Figure 3 is also a generic abstract model, shown in Figure 4. This model remains structured with two sequences, starting and ending by *request* and *report* events, and keeps clock constraints in locations.

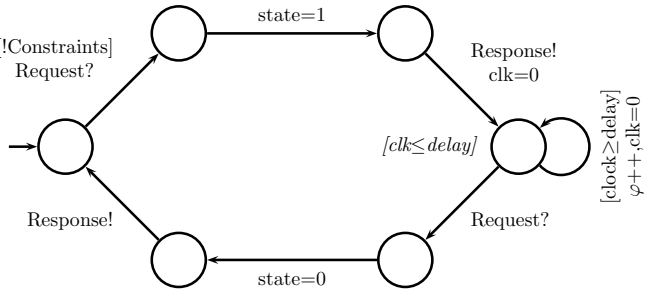

Figure 4. Abstract generic model obtained by projection

C. Reachability analysis and sequencers generation

1) Higher level sequencer: The role of a sequencer is to restrict the behaviour of the system. Sequencers are build iteratively by refining a fully permissive sequencer down to a device level action sequencer.

The higher level sequencer, which is the most permissive, is only concerning function events and recipe constraints (Figure 5). It enables to start every functions of the system, which set is dependant of the considered system. After each function start request, it evaluates if the goal situation is reached. This goal is modelled by a guard corresponding to the characterization of the goal situation in the recipe model (guard on the transition entering in the goal location).

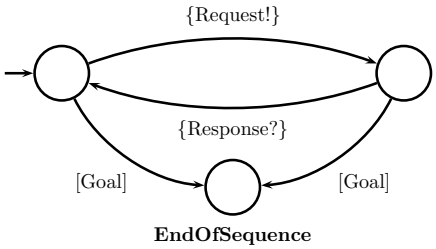

Figure 5. Higher level sequencer

The refinement of sequencers concerns the events which are involved. This is done using iterative reachability analyses, which generate at each hierarchical level execution traces interpreted into a lower level sequencer, as described in the next subsections.

2) Execution trace generation: The reachability analysis is performed *via* model checking, using on the one hand system and sequencer models, and on the other hand a property expressing that the sink location of the sequencer is reachable (*EF Sequencer.EndOfSequence*), as shown in Figure 1. If the sink location is reachable, the analysis returns a timed execution trace leading from the initial location of the model to the satisfaction of the property. This trace consists in a timed set of states and transitions, which is then interpreted into a lower level sequencer.

3) Interpretation of an execution trace into a sequencer: In order to generate a new sequencer at level $N-1$, which is a refinement of the sequencer of level N, transformations are made to the execution trace. Indeed, the trace consists in a sequence of events, defined on events of L^N and L^{N-1}, which reception and emission by the system are necessary to reach the goal location. In order to restrict the behaviour of the system at the level $N-1$, the sequencer will emit and receive these events:

- Requests received by the functions at level $N-1$ become emitted events by the new sequencer, at the same clock value;
- Responses emitted by the functions at level $N-1$ become received events by the new sequencer, at the same clock value.

IV. A CASE STUDY: CISPI

A. Case study presentation

CISPI[2] is a lab platform dedicated to Safe and Interactive Operating of Industrial Processes. It represents some key features of an auxiliary feed water system, with various physical redundancies. A partial process flow diagram of the platform is given in Figure 6.

[2]http://safetech.cran.univ-lorraine.fr/

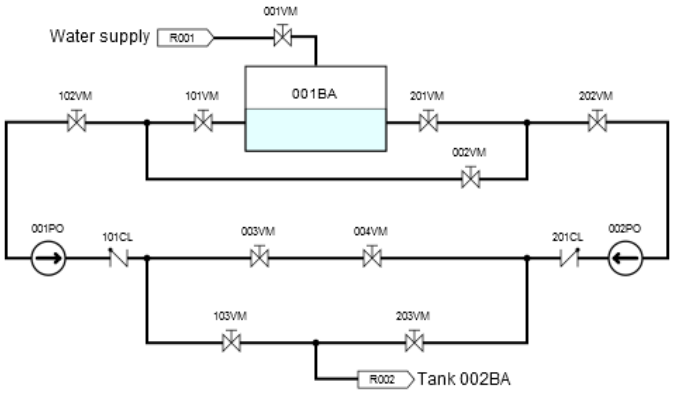

Figure 6. Process flow diagram of the CISPI platform

Various operational situations of CISPI are included in the recipe model (Figure 7). It corresponds to a view of the system at the higher hierarchical level. The possible situations considered are characterized by tank levels. As three operating situations are considered, three locations are used in the model. Transition from one location to another is conditioned by values on physical variables. For example, *_002BA>=2* indicates that it is necessary to have a level higher than 2 in the tank 002BA to reach the situation modelled by the location on the right.

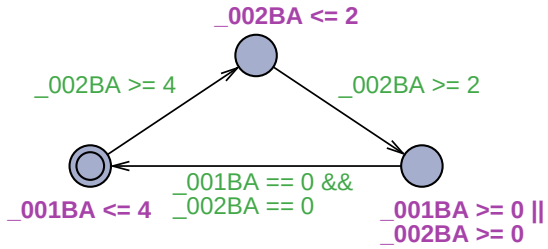

Figure 7. Recipe model

According to the various flows redundancies and devices of the process, 9 functions of this platform are considered (F1 to F9). Their configurations are given in Table I and have an influence on the levels in tanks 001BA and 002BA. In the table:

- ”o” means that the valve must be opened;
- ”c” means that the valve must be closed;
- ”r” means that pump must be running;
- ”-” means that the state of the device is of no importance;
- ”Delay” gives the delay which is needed to have a evolution of physical values;
- $\downarrow$ of $\uparrow$ indicate if the variable is decremented or incremented by the function.

	001VM	101VM	102VM	103VM	201VM	202VM	203VM	002VM	003VM	004VM	001PO	002PO	Delay	001BA	002BA
F1	-	o	o	o	-	-	c	c	c	-	r	-	4	↓	↑
F2	-	o	o	c	-	-	o	c	o	o	r	-	8	↓	↑
F3	-	-	-	c	o	o	o	c	-	c	-	r	7	↓	↑
F4	-	-	-	o	o	o	c	c	o	o	-	r	8	↓	↑
F5	-	o	c	c	c	o	o	o	-	c	r	-	9	↓	↑
F6	-	o	c	o	c	o	c	o	o	o	r	-	10	↓	↑
F7	-	c	o	o	o	c	c	o	-	c	-	r	9	↓	↑
F8	-	c	o	c	o	c	o	o	o	o	-	r	10	↓	↑
F9	o	-	-	-	-	-	-	-	-	-	-	-	1	↑	-

Table I
FUNCTIONS CONFIGURATIONS

The generic models presented in section III have been instantiated to model the 12 devices and 9 functions of the CISPI platform. It is assumed that the system is in an initial situation where all devices are closed or stopped and all tanks are empty. The function detailed models have been projected to obtain function abstract models, according to the projection operator defined in section III.

The implementation of recipe, function, device and sequencer models with timed automata has been performed with the Uppaal model checking tool [19]. This tool is also used to perform reachability analyses.

B. Iterative sequence generation results

As the case study proposed in this article considers three hierarchical levels, the sequence generation is applied iteratively two times:

- First step, on model of the recipe, abstract models of the functions, higher level sequencer, to obtain a refined functions sequencer;
- Second step, on detailed models of the functions, abstract models of the devices, sequencer previously generated, to obtain the final action sequence.

1) First step: "recipe" and abstract "functions" levels: Using models of the recipe, abstract models of the functions, and the higher level sequencer (the most permissive), a first reachability analysis is performed to evaluate the property *EF Sequencer.EndOfSequence*. Eventually, as the property is evaluated to true, an execution trace is generated, and given thereafter:

- at T=0
  - StartF1?
  - F1Started!
  - StartF9?
  - F9Started!
- at T=8
  - StopF9?
- at T=20
  - *Property is verified.*

This trace indicates that, to reach the goal situation, F1 and F9 have to be started at T=0, F9 has to be stopped at T=8, and the goal situation is reached at T=20.

2) Second step: detailed "functions" and abstract "devices" levels: The execution trace produced then is transformed, using the rules defined in section III, into a timed automata modelling this action sequence (Figure 8).

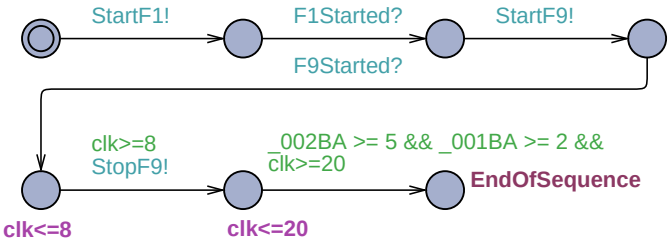

Figure 8. Sequencer generated using previous execution trace

This new sequencer automata is used, along with detailed models of functions and abstract models of device, in a second generation step. The verification of the property *EF Sequencer.EndOfSequence* eventually leads to the generation of a new execution trace, which is a refinement of the previous one, given thereafter:

- at T=0
  - StartF1?
    - Open101VM?
    - _101VMOpened!
    - Open102VM?
    - _102VMOpened!
    - Open103VM?
    - _103VMOpened!
    - Launch001PO?
    - _001POLaunched!
  - F1Started!
  - StartF9?
    - Open001VM?
    - _001VMOpened!
  - F9Started!
- at T=8
  - StopF9?
- at T=20
  - *Property is verified.*

This trace is a refinement of the previous one, with the addition of device related events. It precises how the functions F1 and F9 can be configured by actions on devices. This trace can finally be proposed to operate the system in order to reach the goal situation.

V. Discussion and perspectives

This article proposes a framework using jointly a bottom-up approach to build models and a top-down iterative approach to generate action sequences. It has been illustrated here on a reduced scale lab case study, which size enables its inclusion in an article, to show its feasibility. For this example, the system was decomposed in three levels, though the proposed approach could be used with more. Using a breadth first search algorithm, reachability analysis has then been performed twice. On the upper level, the property verification took 539 ms to return the first sequence indicating in which order some functions can be used. It then took 284 ms to determine the actions sequence to perform on devices.

The scalability of the approach has been evaluated on larger systems, including an industrial size case study from the CONNEXION Cluster[3], composed of 71 devices and 11 functions. Sequences where then generated in a time lower than 2 seconds, where the approach proposed previously in [1] failed to find a possible action sequence due to combinatorial explosion.

In future works, the projection operator will be formalized in order to take into account explicitly guards and invariants, and an algorithm to automatically deduce sequencers from execution trace will be proposed. Also, the notion of cone of influence [15], which principle is to focus only on the variables of interest to reach the goal situation, will be investigated in order to reduce even more the state space to explore.

Acknowledgment

This works is supported by the CONNEXION Cluster, financed through the "*Investissements d'Avenir / Briques Génériques du Logiciel Embarqué*".

References

[1] T. Cochard, D. Gouyon, and J.-F. Pétin, "Generation of safe plant operation sequences using reachability analysis," *20th IEEE International Conference on Emerging Technologies and Factory Automation*, 2015.

[2] ISA, "Ansi/isa-88.01-1995 : Batch control - part 1 : Models and terminology," *The Instrumentation, Systems and Automation Society*, 1998.

[3] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[4] I. TC65, "Iec 61512-1: Batch control–part 1: Models and terminology," *International Electrotechnical Commission*, 1997.

[5] S. Viswanathan, C. Johnsson, R. Srinivasan, V. Venkatasubramanian, and K. E. Ärzen, "Automating operating procedure synthesis for batch processes: Part i. knowledge representation and planning framework," *Computers & chemical engineering*, vol. 22, no. 11, pp. 1673–1685, 1998.

[6] ——, "Automating operating procedure synthesis for batch processes: Part ii. implementation and application," *Computers & chemical engineering*, vol. 22, no. 11, pp. 1687–1698, 1998.

[7] Y.-F. Wang, H.-H. Chou, and C.-T. Chang, "Generation of batch operating procedures for multiple material-transfer tasks with petri nets," *Computers & chemical engineering*, vol. 29, no. 8, pp. 1822–1836, 2005.

[8] J.-H. Li, C.-T. Chang, and D. Jiang, "Systematic generation of cyclic operating procedures based on timed automata," *Chemical Engineering Research and Design*, vol. 92, no. 1, pp. 139–155, 2014.

[9] J. R. Rivas and D. F. Rudd, "Synthesis of failure-safe operations," *AIChE Journal*, vol. 20, no. 2, pp. 320–325, 1974.

[10] K.-E. Arzen and C. Johnsson, "Object-oriented sfc and isa-s88. 01 recipes presented at the world batch forum," *ISA transactions*, vol. 35, no. 3, pp. 237–244, 1996.

[11] M. Lind, H. Yoshikawa, S. B. Jørgensen, M. Yang, K. Tamayama, and K. Okusa, "Multilevel flow modeling of monju nuclear power plant," *Nuclear safety and simulation*, vol. 2, no. 3, pp. 274–284, 2011.

[12] A. Valmari, "The state explosion problem," in *Lectures on Petri nets I: Basic models*. Springer, 1998, pp. 429–528.

[13] A. Bouajjani, P. Habermehl, and T. Vojnar, "Abstract regular model checking," in *Computer Aided Verification*. Springer, 2004, pp. 372–386.

[14] G. Faraut, L. Piétrac, and E. Niel, "Formal approach to multimodal control design: Application to mode switching," *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 4, pp. 443–453, 2009.

[15] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[16] P.-A. Brameret, A. Rauzy, and J.-M. Roussel, "Automated generation of partial markov chain from high level descriptions," *Reliability Engineering & System Safety*, vol. 139, pp. 179–187, 2015.

[17] D. Gouyon, J.-F. Pétin, and A. Gouin, "Pragmatic approach for modular control synthesis and implementation," *International Journal of Production Research*, vol. 42, no. 14, pp. 2839–2858, 2004.

[18] C. Vogel, *Génie cognitif (In French)*. Masson, 1988.

[19] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "Uppaal 4.0," in *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*. IEEE, 2006, pp. 125–126.

[3] http://www.cluster-connexion.fr/