R-TNCES Rebuilding: A New Method of CTL Model Update for Reconfigurable Systems

Mohamed Ramdani[1,2,3,4], Laid Kahloul[2], Mohamed Khalgui[1,3] and Yousra Hafidi[1,2,3,4]
[1]*LISI Laboratory, National Institute of Applied Sciences and Technology, University of Carthage, Tunis 1080, Tunisia*
[2]*LINFI Laboratory, Computer Science Department, Biskra University, Algeria*
[3]*School of Electrical and Information Engineering, Jinan University, China*
[4]*University of Tunis El Manar, Tunis, Tunisia*

Keywords: Reconfigurable Discrete-Event System, Reconfigurable Timed Net Condition Event System, Computation Tree Logic, Model Rebuilding.

Abstract: This paper deals with improved rebuilding of models in formal verification of reconfigurable discrete-event systems (RDESs) modeled by reconfigurable timed net condition event systems (R-TNCESs). Automated computation tree logic (CTL) model update and repair are approaches that extend model-checking to generate a new correct model that represents the desired behavior. We propose R-TNCES rebuilding method, which is a formal method that allows both the verification and the modification of reconfigurable models. The proposed approach generates from an incorrect model a new one that satisfies a given CTL formula. First, CTL is defined to deal with the system functional properties specification. A set of transformation rules with an algorithm are proposed to achieve the rebuilding phase. Finally, platform FESTO MPS is used as a case study to demonstrate the paper's contribution. The obtained results show the efficiency of the proposed contribution even in large complex systems.

1 INTRODUCTION

Discrete event systems (DESs) can change their state in an asynchronous and a nondeterministic way due to the occurrence of events. The auto-control deployment in such systems innovates a new class of systems called: reconfigurable discrete event/control systems (RDECSs), which can work under various conditions: concurrency, control, communication, etc. Such class includes manufacturing systems(Zhang et al., 2018), real time systems (Lakhdhar et al., 2018), and embedded systems (Housseyni et al., 2018), etc. Reconfiguration is the set of internal changes to adapt the external changes and user requirements (Zhang et al., 2015) (changes in the structure, functionality, and control algorithms). In spite of the complexity of RDECSs, model-checking is an effective technique for the automatic verification of those systems. Model-checking verifies the satisfaction between a behavior formal model (Petri nets, automaton, etc.) and a functional property specification, which is specified by a temporal logics (computation tree logic, linear temporal logic, etc.). In order to deal with reconfigurable systems, many formalisms are proposed and extended. Petri net is the most used formalism developed to cope with reconfigurability (reconfigurable Petri nets (Padberg and Kahloul, 2018)). Reconfigurable time net condition/event systems (R-TNCES) are one of their extensions (Zhang et al., 2013).

In the last decade, model-checking is also progressing and improved to be more efficient in the debugging of errors and their auto-correction. Computation tree logic (CTL) *update* and *repair* is one of this extension, which is a method that modifies the system model in order to satisfy a given formula. In such a context, many works of system model modification are developed. Ding and Zhang in (Ding and Zhang, 2007), (Zhang and Ding, 2008) proposed an algorithm based on five basic operations and minimal change criteria. Carrillo and Rosenblueth proposed another algorithm and introduced the protection concept in (Carrillo and Rosenblueth, 2014). Martinez and Lopez proposed a CTL repair methodology for different classes of Petri nets, (Martínez-Araiza and López-Mellado, 2014) for labeled state machines (LSM), (Martínez-Araiza and López-Mellado, 2015) for bounded and deadlock free Petri nets, and (Martinez-Araiza and López-Mellado,

Ramdani, M., Kahloul, L., Khalgui, M. and Hafidi, Y.
R-TNCES Rebuilding: A New Method of CTL Model Update for Reconfigurable Systems.
DOI: 10.5220/0007736801590168
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 159-168
ISBN: 978-989-758-375-9

2016) for open work-flow nets (oWFN).

The verification layer-by-layer proposed in (Zhang et al., 2013) and the formal verification proposed in (Hafidi et al., 2018) are incapable to automate the correction of a model which does not satisfy a property formula. The system complexity, the high number of properties to be checked and the absence of a technique that facilitates the debugging task make the verification phase of R-TNCESs a hard task.

In this paper, we deal with the verification and model updating (modification) of reconfigurable systems modeled with R-TNCESs. In order to deal with the absence of CTL model update approaches for this formalism, we propose a new methodology called R-TNCESs rebuilding. First, we compute a Kripke structure model from the behavior module of R-TNCES. Second, we formulate a CTL formula to be verified in the Kripke model, based on the original formula written on R-TNCES. Then, the error will be localized and isolated using the five primitives of Ding and the minimal change criteria (Ding and Zhang, 2007). Finally, according to the equivalence between the changes in the Kripke model and the modification instructions of the R-TNCESs formalism, we apply the rebuilding of R-TNCES to get a new model which satisfies the given property.

We illustrate the contribution and validate our methodology through an academic case study FESTO MPS (Zhang et al., 2013), which is a lab-scale station. We use the behavior module of FESTO MPS to show the performance of the different algorithms and to check properties of broadcasting and synchronizations. To confirm the result of R-TNCESs rebuilding, we use the SESA model checker (Starke and Roch, 2002). Indeed, SESA is a software to analyze TNCESs and to compute the exact reachable set of states.

In this research, we aim to ensure the viability of the R-TNCES model by checking properties on the broadcasting and its synchronization. This work contains four main contributions: (i) Providing a generalization of the CTL model update from Petri nets formalism to a rebuilding problem concerned by corrections in a complex systems (i.e., RDECSs) modeled by R-TNCESs formalism. In this granularity passage, we paid attention to the correct abstraction between formalisms. The classical CTL update, in classical Petri nets, is a place/transition rebuilding, whilst the current one rebuilds modular entities (called control components: CCs) and their connections in the model which facilitates the on-line intervention and fault isolation. (ii) Developing an approach for the model rebuilding. Our approach is able to generate a new model that satisfies requirements specified by a CTL formula while respecting the original model and the minimal changes stated by Ding (Ding and Zhang, 2007). (iii) Developing an algorithm to compute a Kripke structure from an R-TNCES model, and an another algorithm to adapt a CTL formula without losing information. (iv) Demonstrating the impact of the modular rebuilding in R-TNCESs to isolate the modular bugs in reconfigurable systems.

The paper contains four main sections organized as follows: In section 2, we present backgrounds and some preliminary theories. Section 3 analyzes the semantics of CTL update approach and gives the R-TNCES rebuilding operation methodology. Section 4 illustrates the approach and algorithms application through an academic case study. Finally, Section 5 concludes this work and describes our perspectives.

2 PRELIMINARIES

This section presents the basic concepts and notations used in this paper.

2.1 Reconfigurable Time Net Event Condition Systems

R-TNCESs represent a formalism which was proposed in (Zhang et al., 2013) to specify and verify reconfigurable discrete event control systems (RDECSs). An R-TNCES RTN is a couple $RTN = (B,R)$. B is the behavior module such that, $B = (Conf_1,\ldots,Conf_n)$ (n configurations, each one is a TNCES, possibly redundant). R is the control module such that, $R = (r_1,\ldots,r_m)$ (set of reconfiguration functions with $n,m \in \mathbb{N}$). Formally, the behavior module is a tuple, defined as follows.

$$B = (\mathbb{P},\mathbb{T},\mathbb{F},W,\mathbb{CN},\mathbb{EN},\mathbb{DC},V,Z_0) \qquad (1)$$

where, $\mathbb{P}$ (resp, $\mathbb{T}$) is a superset of places (resp. transitions), $\mathbb{F}$ is a superset of arcs, $W : (P \times T) \cup (T \times P) \rightarrow \{0,1\}$ maps a weight to a flow arc, $\mathbb{CN}$ (resp. $\mathbb{EN}$) is a superset of condition signals (resp, event signals), $\mathbb{DC}$ is a superset of clocks on output arcs, V: $T \rightarrow \{AND, OR\}$ maps an event processing mode for every transition, and $Z_0 = (M_0, D_0)$, where M_0 is the initial marking, and D_0 is the initial clock position.

Definition 1. (Control component CC) is a logical software unit (Khalgui and Hanisch, 2011), which represents the data-flows and actions of sensors/actuators (algorithms, extraction or activation). Every CC resumes the physical process in three actions: activation, working, and termination. Figure 1 shows a generic model of a CC.

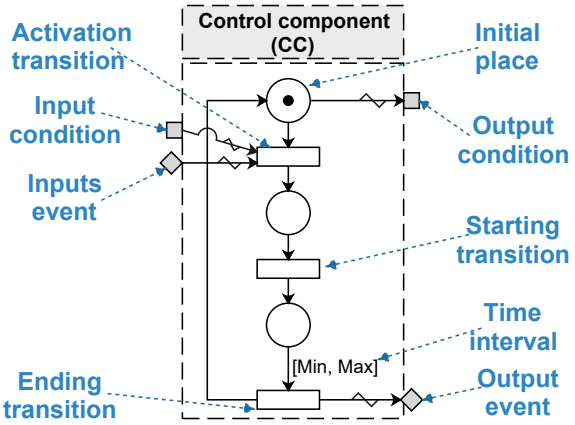


Figure 1: The generic model of a control component.

Definition 2. (Time Net Condition/Event System TNCES) is a set of CCs interconnected by signals. The order of CCs describes the desired behavior of the TNCES.

2.2 Kripke Structure

Over a finite set of atomic propositions AP, Kripke structure M is a 4-tuple $M = (S, s_0, R, L)$, such that:

- S is a finite set of states.
- s_0 is the initial state.
- $R : r \leftarrow S \times S$ is a transition relation, i.e., $\forall s \in S$, $\exists s'$ such that $(s, s') \in R$.
- $L : S \leftarrow 2^{AP}$ is a label function.

A path $\pi = s_0, s_1, s_2, \ldots$ is an infinite sequence of states in M from the initial state such that for all $i \geq 0$ $\exists (s_i, s_{i+1}) \in R$.

2.3 Computation Tree Logic

The model-checking of R-TNCESs is an automatic verification technique of a system using finite-state systems and their reachability graphs. The properties, to be checked, are specified using one temporal logic such as CTL and its extensions. CTL (Baier et al., 2008) is used to specify the functional properties. The time is not explicitly expressed using CTL, but it is possible to say if a property will frequently/infrequently be verified or will never be verified. CTL offers facilities for the specification of properties that must be fulfilled by the system, like safety, liveness, reachability, etc. A formula holds in the system if it is proved true in the initial state of that system. The set of Computation Tree Logic formulas is defined inductively in (Baier et al., 2008) by the following grammar.

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi \qquad (2)$$

where a is an atomic proposition and φ is a path formula with the following syntax ($E \equiv \exists$ and $A \equiv \forall$):

$$\varphi ::= EX\Phi \mid AX\Phi \mid EF\Phi \mid AF\Phi \mid EG\Phi \mid AG\Phi \mid E\Phi_1 U \Phi_2 \qquad (3)$$

where, Φ, Φ_1 and Φ_2 are CTL state formulas. Due to the existence of duality rules in CTL (Baier et al., 2008), we can omit the path quantifier $\forall$ (A) and use CTL existential normal form (ENF). The set of CTL formulas in ENF is given by:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid EX\Phi \mid EG\Phi \mid E(\Phi_1 U \Phi_2) \qquad (4)$$

2.4 Computation Tree Logic Update

Ding and Zhang have developed a formal approach for computation tree logic model update based on minimal change criteria over Kripke structure models (Ding and Zhang, 2007). CTL model update is an approach for the automatic verification and modification of system models. The principle used is to generate admissible models that represent the correct design(Zhang and Ding, 2008) in order to repair software errors. The model updater functions modify the models using five primitives ($PU1, \ldots, PU5$).These primitives are described in their simplest forms as follows.

- PU1: Adding a relation.
- PU2: Removing a relation.
- PU3: Changing the label of one state.
- PU4: Adding a state and its associated relations.
- PU5: Removing a state and its associated relations.

The semantics of the above primitives and of the minimal changes principle are detailed in (Zhang and Ding, 2008).

3 REBUILDING OPERATION FOR RECONFIGURABLE MODELS

3.1 Formalization

Given a system model M with s_0 its initial state and a CTL formula ϕ such that $(M, s_0) \not\models \phi$, the rebuilding problem can be defined as finding a new model M' such that $(M', s'_0) \models \phi$. M' is the repaired model of M, otherwise, there is a problem in the specification of the model/property (inconsistency). M' must respect

the good specification and must conserve the good requirements of the system with minimal changes. Figure2 illustrates the general problem of checking and rebuilding.

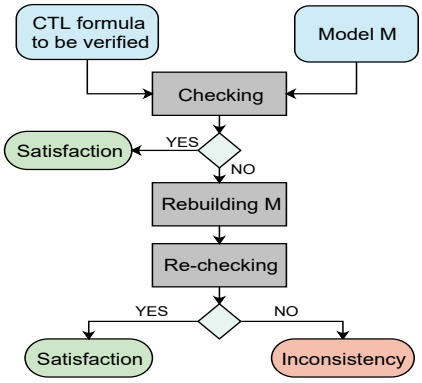


Figure 2: General problem.

Rebuilding operation (*RO*) can be formalized as follow:

$$RO = (Z_0, \phi, I) \tag{5}$$

Where,

- $Z_0 = (m_i, D_i)$ is the initial state of the system, such that m_i is the marking of the system and D_i is the clock of the system.
- ϕ is a CTL formula that specifies a functional property on the behavior module.
- I is the rebuilding operation instruction chain. I can be restricted to add/delete components operations.

According to Ding's primitives in the CTL update for the Kripke structure and using the R-TNCES's modification instructions, we define the following dualities between the above instructions to be deployed in the rebuilding of reconfigurable systems specified using R-TNCESs. Table 1 resumes dualities.

Table 1: Equivalence between Ding primitives and R-TNCES rebuilding modifications.

	R-TNCES rebuilding modifications	modification instruction
$PU1$	Add event signal	$Cr(ev(t,t'))$
$PU2$	Delete event signal	$De(ev(t,t'))$
$PU3$	Add/Delete condition signal	$Cr(cn(p,t))$ / $De(cn(p,t))$
$PU4$	Add control component	$Cr(CC))$
$PU5$	Delete control component	$De(CC))$

3.2 TNCESs Rebuilding

Given a TNCES TN and a CTL formula ϕ, the rebuilding operation (*RO*) consists of synchronization verification between CCs, e.i., checking the synchronization faults between transitions in different CCs (the broadcasting correctness) then to repair the TNCES model if necessary. We can resume (*RO*) of a TNCES as follows.

1. Structural rebuilding of signals which consists to enable/disable one control component by adding/ deleting its signals.
2. Update of TNCES (configuration) which consists of adding/deleting a whole CC (or set of CCs).

We denote by $EN(^\bullet T)$/ $EN(T^\bullet))$ the set of entering/ exiting events of transition T. Table 2 recapitulates the above faults and their correction.

3.2.1 Illustrative Example

Let's consider Z as the TNCES shown in figure 3. As a functional property, CC_2 and CC_3 should not be joined in the same execution. Trivially, the CTL formula 6 is not satisfied by this TNCES, so that we need to rebuild the TNCES to be adequate for the required functional property.

$$\phi = AF(p_6 \rightarrow \neg EF\, p_9) \tag{6}$$

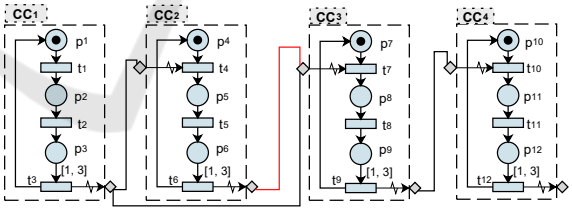


Figure 3: illustrative example .

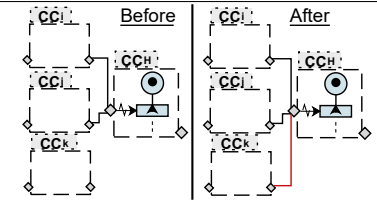
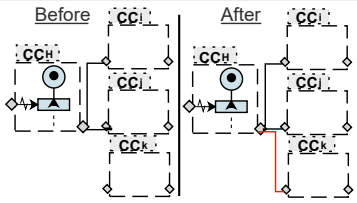
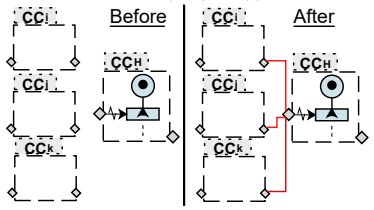
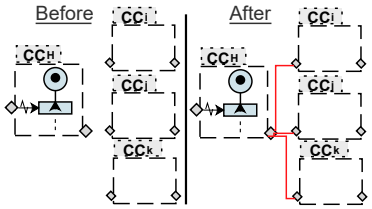
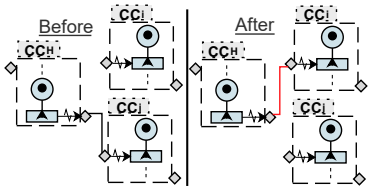
The rebuilding operation *RO*, in this case, can be achieved by deleting the synchronization signal between CC_2 and CC_3. Formally:

$$RO = (Z, \phi, I = (De(ev(t_6, t_7))) \tag{7}$$

For TNCES update, let's suppose that the designer needs to substitute CC_3 by CC_5 and to add a new process after CC_4. Formally, instructions chain of this *RO* is described as: $I = De(ev(t_3,t_7)) + De(CC_3) + De(ev(t_9,t_{10})) + Cr(CC_5) + Cr(ev(t_3,t_{13})) + Cr(ev(t_{15},t_{10})) + Cr(CC_6) + Cr(ev(t_{12},t_{16}))$.

Figure 4 depicts the TNCES update result.

Table 2: Table of synchronization faults and their corrections.

Faulty results	$\exists t_i \in EN(^\bullet T)/W(t_i,T)=0$	$\exists t_i \in EN(T^\bullet)/W(T,t_i)=0$
Correction	$Cr(ev(t_i,T))$	$Cr(ev(T,t_i))$
Explication graphic		
$\forall t_i \in EN(^\bullet T)/W(t_i,T)=0$	$\forall t_i \in EN(T^\bullet)/W(T,t_i)=0$	$\exists t_i \in EN(T)/W(t_i,T')=1$
$\forall t_i Cr(ev(t_i,T))$ 	$\forall t_i Cr(ev(T,t_i))$ 	$De(en(t_i,T'))+Cr(ev(t_i,T))$ 

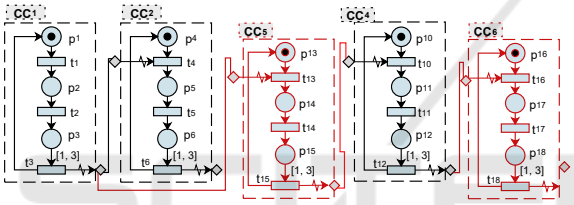


Figure 4: TNCES update illustrative example.

3.3 Generalization of TNCES Rebuilding

Given M an R-TNCES and Φ a set of CTL formulas in ENF to be checked. For the verification process of those CTL formulas, we can divide the whole process into two steps, so that exactitude of the first step initiates the second, as follows:

1. Check the consistency of the control module and apply the rebuilding if needed.
2. Check the consistency of the behavior module and apply the rebuilding if needed.

Rebuilding an ordinary Petri net consists in adding/deleting place/transition (Carrillo and Rosenblueth, 2014). Whilst, control module rebuilding, in R-TNCESs, consists in enable/disable one configuration by adding/deleting of signals or adding/deleting an entire configuration of the control module (according to the fundamental structure modification instructions of R-TNCES (Zhang et al., 2013)).

Assumption.1 In this work, we assume that the control module is not faulty and represents the desired behavior, i.e., the reachability of each configuration is covered by the control module.

Assumption.2 Every CC respects the good requirements of the designer and it is not in a faulty case.

We concentrate on the behavior module response when a reconfiguration is requested or an error occurs. In this work, we focus on the deployment of the second step. Figure 5 illustrates the verification steps of an R-TNCES.

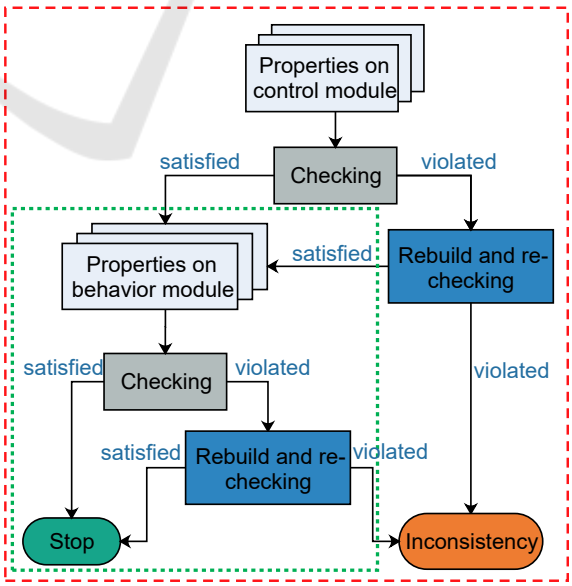


Figure 5: The verification process of R-TNCES.

3.4 R-TNCES Rebuilding

Rebuilding operation (*RO*) of an R-TNCES reposes on two basic sub-processes that assure the needed abstraction of both model and formula to facilitate the CTL update process. The first one is the computation of a new Kripke structure model from the behavior module B using Algorithm1. Indeed, each control component (resp, event signal) in B becomes a state (resp, relation) in the Kripke structure. The second sub-process consists in the transformation of a CTL formula expressed in R-TNCES to adapt it for the Kripke structure verification. This transformation preserves path formulas and expresses state's formulas according to their CCs. Algorithm 2 computes the abstraction of CTL formulas without any loss of information for the Kripke structure verification.

Algorithm 1: Kripke Structure generation.

Input: $B = \sum TNCES$;
Output: $Sk = (S, s_0, R, L)$;
  for *each control component* CC_i,*i=1..n* **do**
    Create state $s_i \in S$;
    Create label $L : s_i \rightarrow (cc_i)$;
  end
  for *each control component* CC_i,*i=1..n* **do**
    if $(\exists ev(t_j, t_k) / t_j \in CC_i$ *and* $t_k \in CC_k)$ **then**
      Create relation $(s_j, s_k) \in R$;
    end
  end
  for *each control component* CC_i,*i=1..n* **do**
    if $(\exists cn(p_j, t_k) / p_j \in CC_i$ *and* $t_k \in CC_k)$ **then**
      Create label $L : s_k \rightarrow (cc_j)$;
    end
  end
  $s_0 \leftarrow s_1$;/*CC_1 is the 1^{st} physical process*/
  Return (SK);

Given an R-TNCES $RTN = (B, R)$ and a CTL formula ϕ in ENF, we can define a *six steps* methodology for automatizing the rebuilding of behavior modules as follows.

1. Transform the behavior module B to Kripke structure SK using Algorithm 1.
2. Transform the formula ϕ to adapt the same semantic value in the Kripke structure using Algorithm 2.
3. Check the satisfaction of $SK \vDash \phi$. If $SK \vDash \phi$, then the model is well specified, otherwise, go to the next step.
4. Modify SK according to CTL model update approach (Zhang and Ding, 2008); action to be supervised by the designer.

Algorithm 2: CTL transformation.

Input: Φ expressed on R-TNCES;
Output: Φ' expressed on Sk;
  for *each sub-formula* $\phi \in \Phi$ **do**
    if $(\phi$ *is a path formula)* **then**
      CTL transformation(ϕ, Φ');/*Recursivity*/
    end
    else
      if $(\phi$ *is a state formula)* **then**
        for *each* p_i, t_i *expressed in* ϕ **do**
          Replace $(p_i, s_i) / p_i \in CC_i$;
          Replace $(t_i, s_i) / t_i \in CC_i$;
        end
      end
    end
  end
  Return (Φ');

5. Re-check the satisfaction of the modified model $SK \vDash \phi$. If $SK \vDash \phi$, then the model is well modified, otherwise, there is an inconsistency in the specification (formula/model or both).
6. Rebuild B using primitives that are equivalent to those executed in the 4th step. Table 1 presents the equivalence between the primitives of CTL update and R-TNCES rebuilding.

Figure 6 summarizes the above steps of the rebuilding operation *RO* for an R-TNCES model.

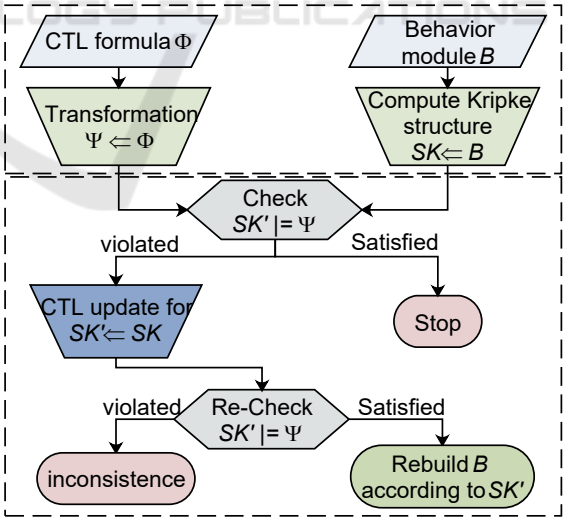


Figure 6: Methodology of R-TNCES rebuilding.

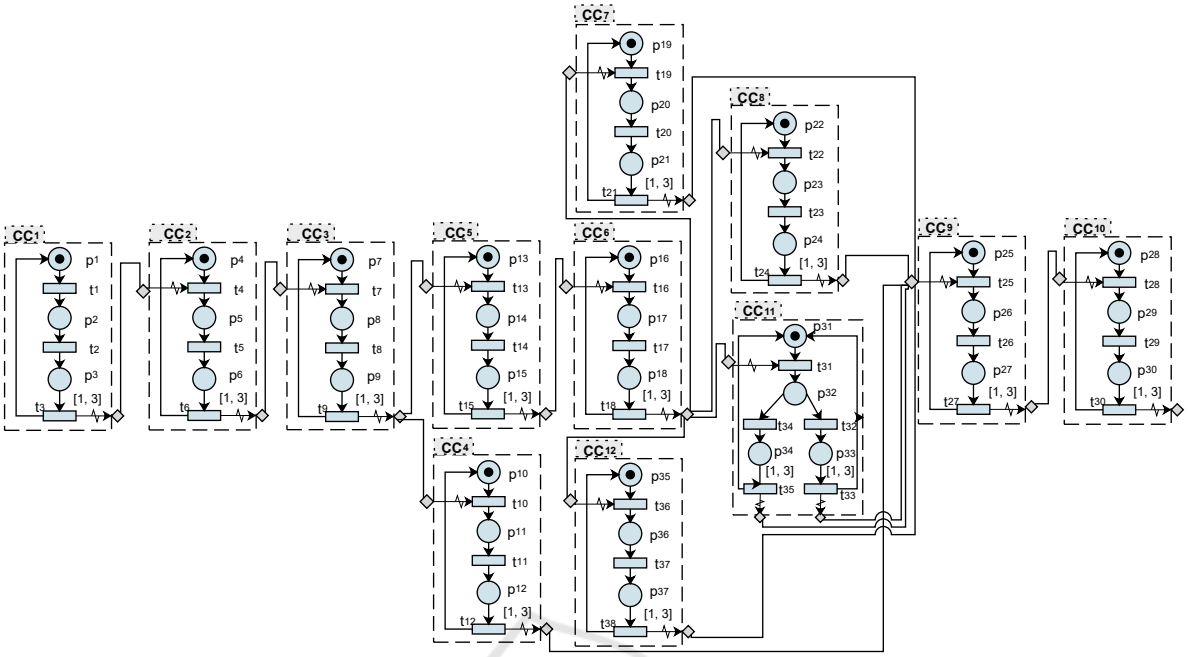


Figure 7: Behavior module *B* of FESTO MPS.

4 EXPERIMENTAL STUDY

4.1 Case study

To validate and to demonstrate the gain of the proposed contribution, let us consider the R-TNCES model shown in figure 7. It is a faulty model of behavior module of FESTO MPS, which is a lab-scale production line simulating different functions of a factory. FESTO MPS is well described and detailed in (Zhang et al., 2013). FESTO achieves the physical processes sequentially and executes them in configurations ($Config_1$,$Config_2$,$Config_3$, $Config_4$). It is assumed that every physical process (i.e., *Election, Convert, Test, Test failed, Elevate, Rotate, Drill1, Drill2, Drill1 OR Drill2, Drill1 And Drill2, Checker, Evacuation*) is modeled by one control component (*CC*) and every configuration $Config_i$ has one control chain (C_{chain_i}). The control chains describing the physical process are as following.

- $C_{chain_1} = (CC_1, CC_2, CC_3, CC_4)$, when $Test failed$ is executed.
- $C_{chain_2} = (CC_1, CC_2, CC_3, CC_5, CC_6, CC_7, CC_9, CC_{10})$, when $Drill1$ is executed.
- $C_{chain_3} = (CC_1, CC_2, CC_3, CC_5, CC_6, CC_8, CC_9, CC_{10})$, when $Drill2$ is executed.
- $C_{chain_4} = (CC_1, CC_2, CC_3, CC_5, CC_6, CC_{11}, CC_9, CC_{10})$, when workpieces alternatively drilled $Drill1$ or $Drill2$.
- $C_{chain_5} = (CC_1, CC_2, CC_3, CC_5, CC_6, CC_{12}, CC_9, CC_{10})$, when workpieces intensively drilled $Drill1$ and $Drill2$.

In order to check that the model satisfies the good requirements, we must ensure the CTL based functional properties. In particular, to ensure the product quality, every workpiece that fails in the first test (quality test: color, material, and height of workpieces) cannot be drilled. To assure the production of a good product, once the workpiece is rejected at the quality test, it is not allowed to that workpiece to proceed to next steps. To check this behavior, we use the following CTL formula:

$$\Phi := AG(p_{12} \rightarrow AF(\neg p_{26})) \quad (8)$$

This formula in the ENF form is written as follows.

$$\Phi := \neg EF(p_{12} \wedge EG(p_{26})) \quad (9)$$

First, we need to compute Kripke structure of the above behavior module using Algorithm 1. The results result is shown in Figure 8.

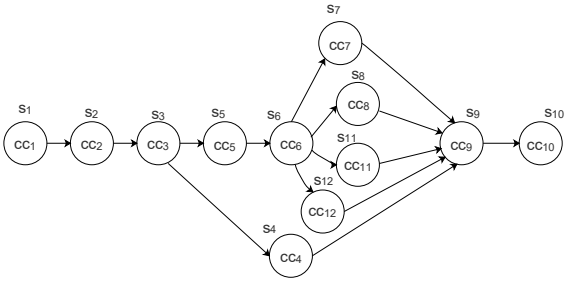


Figure 8: Kripke structure *SK* computation result.

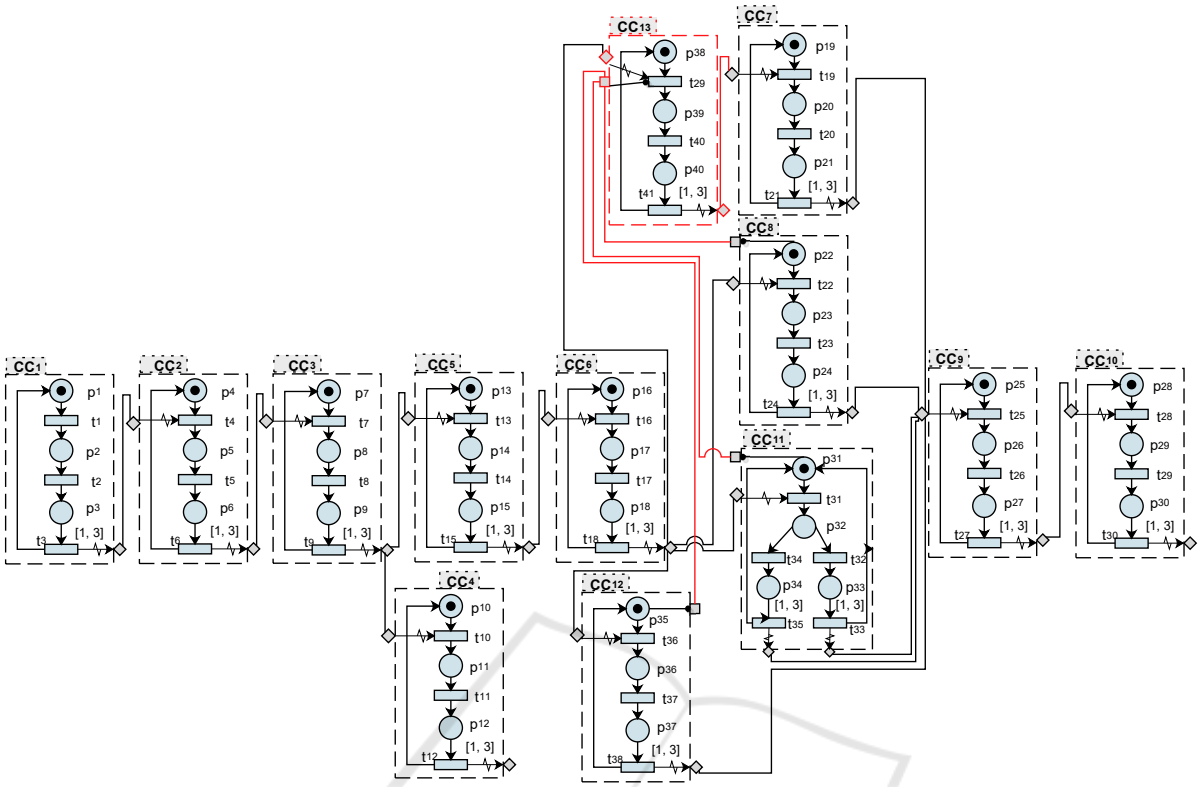


Figure 10: Behavior module B of FESTO MPS after rebuilding.

By applying Algorithm 2, we give the CTL transformation of Φ as follow.

$$\Phi := \neg EF(cc_4 \wedge EG(cc_9)) \quad (10)$$

Formula 10 is proven to be $False$. We select $EG(cc_9)$ to be checked: each path has a state with cc_9 in the model and satisfies $EG(cc_9)$. ($\pi_1 = [s_0 ... s_7, s_9, s_{10}]$; $\pi_2 = [s_0 ... s_8, s_9, s_{10}]$; $\pi_3 = [s_0 ... s_{11}, s_9, s_{10}]$; $\pi_4 = [s_0 ... s_{12}, s_9, s_{10}]$; $\pi_5 = [s_0 ... s_4, s_9, s_{10}]$). Then select the path which have cc_9 and cc_4 ($\pi_5 = [s_0 ... s_4, s_9, s_{10}]$). According to Ding, eventually we need to update SK. We apply $PU2$ to remove the relation (s_4, s_9). Thus, we obtain a new Kripke model SK', which simply states that no transition from state s_4 to state s_9 is allowed. The result of this update is depicted in Figure 9.

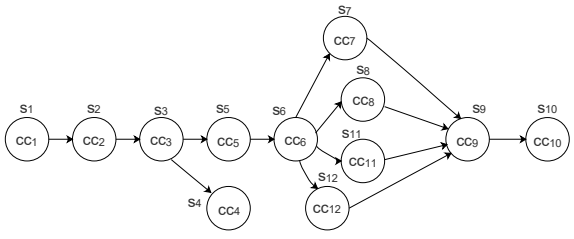


Figure 9: The new Kripke model SK'.

Finally, we apply the equivalent R-TNCES rebuilding modifications of $PU2$ on the module behavior to get a new correct module, i.e., we delete event signal $ev(t_{12}, t_{25})$ by modification instruction $De(ev(t_{12}, t_{25}))$. Results os this last step are shown in Figure 10.

We desire to rebuild the system in order to allow the behavior module to inject a new physical process of drilling $Drill3$. $Drill3$ must proceed before $Drill1$ in $Config_1$ and out of the system in the other configurations. This changes must be supervised and verified. Simply, this operation can be deployed by adding a new state s_{13} in SK' using successively primitive $PU2$ to remove the relation between s_6 and s_7 and $PU4$ to add state s_{13} and its associated relations.

For a more complex case, let's assume that the maps processing mode V of this R-TNCES works on V: $T \rightarrow \{AND\}$ mode. We must ensure that the system cannot execute $Drill3$ at the same time with

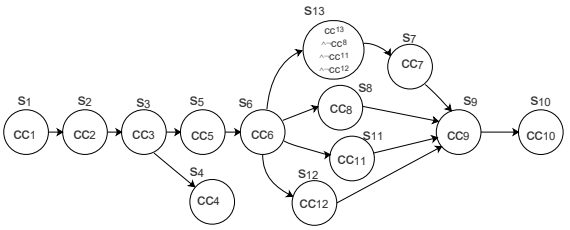


Figure 11: Kripke structure SK'' after adding the $Drill3$ operation.

Table 3: Qualitative comparison with some related works.

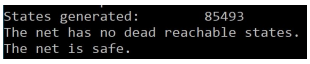
Work	Formalism used	Reconfiguration	Model repair
(Zhang and Ding, 2008), (Ding and Zhang, 2007).	Kripke structure	No	Yes
(Carrillo and Rosenblueth, 2014).	Kripke structure	No	Yes
(Martínez-Araiza and López-Mellado, 2014), (Martínez-Araiza and López-Mellado, 2015), (Martinez-Araiza and López-Mellado, 2016).	Petri nets	No	Yes
(Zhang et al., 2013)	R-TNCESs	Yes	No
(Hafidi et al., 2018)	R-TNCESs	Yes	No
Our work	R-TNCES	Yes	Yes

another configuration. For this safety condition, s_{13} must contain more information on its label. So we will change the label of this state using $PU3$. Figure 11 depicts the new SK'' model.

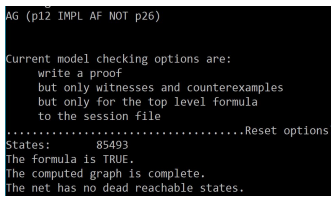
After performing these changes on Kripke model and applying equivalent R-TNCES rebuilding modifications, we get the correct model shown in Figure 10.

The exact reachability graph is computed using the SESA model checker (Starke and Roch, 2002), thus 85493 sates are obtained as shown in sub-figure 12(a). The computed graph is finite and it has no dead reachable states. SESA is applied automatically to verify the deadlock and boundedness properties and it is applied manually to check functional properties. Firstly, we check that the new model satisfies the property of quality ($\Phi := AG(p_{12} \rightarrow AF(\neg p_{26}))$). This formula is proven to be $True$ (sub-figure 12(b)). Then, we verify that it TNCESs's update is well done without system degradation. For this purpose, we verify the following formula.

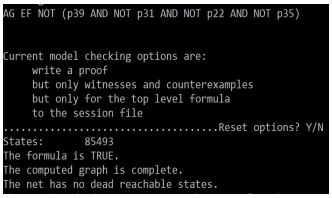
$$AG(EF\neg(p_{39} \wedge \neg p_{31} \wedge \neg p_{22} \wedge \neg p_{35})) \qquad (11)$$



(a)



(b)



(c)

Figure 12: A screen-shot on SESA verification.

Indeed, whatever a state in the reachability graph, $(p_{39} \wedge p_{31} \wedge p_{22} \wedge p_{35})$ must not be satisfied. The formula is proven to be $True$ (sub-figure 12(c)). Figure 12 depicts a screen shot of the model-checking results.

4.2 Discussion

Table 3 describes a short qualitative comparison between the proposed contribution and the most recent related works.

For reconfigurable systems, there is no study in the rebuilding and model correction. However, our proposed methodology facilitates the process of synchronization properties verification. Thus, the classical verification of R-TNCES checks these properties based on the whole model, contrariwise, the R-TNCESs rebuilding (*RO*) provides a verification of an abstract model (Kripke structure) with formal methods to ensure the correctness.

5 CONCLUSION

This work deals with the automatic rebuilding of a reconfigurable system modeled with the R-TNCES formalism and CTL properties specifications.

In this paper, we have presented a computation tree logic model update methodology for reconfigurable systems modeled with reconfigurable timed net condition/event systems (R-TNCESs) called R-TNCESs rebuilding. Based on the CTL update model of the Kripke structure, we define a method that deals with CTL formulas and repairs reconfigurable system models. Our contribution reposes on two fundamental techniques. The first one is an algorithm that computes a Kripke structure based on R-TNCES model. The second one is a new technique to transform a CTL formula expressed on R-TNCESs model to another one expressed on a Kripke model with the same verification value (a granularity passage). In ad-

dition, this work defines an equivalence between Ding primitives and R-TNCESs rebuilding modification instructions. At the end, we confirm the obtained results of R-TNCESs rebuilding operation by an experimental case study using SESA tool to validate the final model. Classically, the designer has to repeat the verification cycle for each violated functional property, with this proposition we use the formula to update the model directly which results in the gain of designer effort and thus reduces the verification time.

This work opens several possible avenues for future researches. First, we plan to apply our approach on real large case studies with different kind of properties not only with synchronization properties. Second, we plan to go further in the granularity degree of the approach, by taking into consideration more details of CTL formulas. Finally, we plan to deal with reconfigurable systems with distributed behaviors.

REFERENCES

Baier, C., Katoen, J.-P., and Larsen, K. G. (2008). *Principles of model checking*. MIT press.

Carrillo, M. and Rosenblueth, D. A. (2014). Ctl update of kripke models through protections. *Artificial Intelligence*, 211:51–74.

Ding, Y. and Zhang, Y. (2007). System modification case studies. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 2, pages 355–360. IEEE.

Hafidi, Y., Kahloul, L., Khalgui, M., Li, Z., Alnowibet, K., and Qu, T. (2018). On methodology for the verification of reconfigurable timed net condition/event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–15.

Housseyni, W., Mosbahi, O., Khalgui, M., Li, Z., and Yin, L. (2018). Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints. *IEEE Access*, 6:2068–2084.

Khalgui, M. and Hanisch, H.-M. (2011). Automatic NCES-based specification and sesa-based verification of feasible control components in benchmark production systems. *International Journal of Modelling, Identification and Control*, 12(3):223–243.

Lakhdhar, W., Mzid, R., Khalgui, M., Li, Z., Frey, G., and Al-Ahmari, A. (2018). Multiobjective optimization approach for a portable development of reconfigurable real-time systems: From specification to implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–15.

Martínez-Araiza, U. and López-Mellado, E. (2014). A ctl model repair method for petri nets. In *World Automation Congress (WAC), 2014*, pages 654–659. IEEE.

Martínez-Araiza, U. and López-Mellado, E. (2015). Ctl model repair for bounded and deadlock free petri nets. *IFAC-PapersOnLine*, 48(7):154–160.

Martinez-Araiza, U. and López-Mellado, E. (2016). Ctl model repair for inter-organizational business processes modelled as owfn. *IFAC-PapersOnLine*, 49(2):6–11.

Padberg, J. and Kahloul, L. (2018). Overview of reconfigurable petri nets. In *Graph Transformation, Specifications, and Nets*, pages 201–222. Springer.

Starke, P. H. and Roch, S. (2002). *Analysing Signal-net Systems*. Professoren des Inst. für Informatik.

Zhang, J., Frey, G., Al-Ahmari, A., Qu, T., Wu, N., and Li, Z. (2018). Analysis and control of dynamic reconfiguration processes of manufacturing systems. *IEEE Access*, 6:28028–28040.

Zhang, J., Khalgui, M., Li, Z., Frey, G., Mosbahi, O., and Salah, H. B. (2015). Reconfigurable coordination of distributed discrete event control systems. *IEEE Transactions on Control Systems Technology*, 23(1):323–330.

Zhang, J., Khalgui, M., Li, Z., Mosbahi, O., and Al-Ahmari, A. M. (2013). R-TNCES: A novel formalism for reconfigurable discrete event control systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(4):757–772.

Zhang, Y. and Ding, Y. (2008). Ctl model update for system modifications. *Journal of artificial intelligence research*, 31:113–155.