# On Improving Parallel Rebuilding of R-TNCESs

Mohamed Ramdani[1,2,3,4], Laid Kahloul[3] and Mohamed Khalgui[1,2][a]

[1]School of Electrical and Information Engineering, Jinan University, China

[2]LISI Laboratory, National Institute of Applied Sciences and Technology, University of Carthage, Tunis 1080, Tunisia

[3]LINFI Laboratory, Computer Science Department, Biskra University, Algeria

[4]University of Tunis El Manar, Tunis, Tunisia

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

Abstract: This study presents an improved parallel rebuilding of reconfigurable timed net condition-event systems (R-TNCESs) modeling reconfigurable discrete-event systems (RDESs). Computation tree logic (CTL) model repair is one of the existing approaches that extends formal verification using model checking, by an automatized debugging phase and updating directly the model to cope with the desired behavior. Our proposition aims to generalize simple rebuilding with one CTL-based function property to parallel rebuilding which allows both verification and modification of a model according to a set of non verified functional properties simultaneously. A couple of transformation algorithms are proposed to conserve the coherency of the model and a property classification method is proposed to frame the parallel execution order. To demonstrate the paper's contribution, a FESTO MPS platform is used as a case study.

## 1 INTRODUCTION

Reconfigurable discrete event/control systems (RDECSs) are auto-control systems which work in an asynchronous and a non-deterministic way due to events occurrence. RDECSs are compatible with a variety of conditions (concurrency, control, communication, etc.) which can be in several systems (manufacturing systems (Gu et al., 2018), real time systems (Ben Aissa et al., 2019), and intelligent control systems (Khalgui and Mosbahi, 2010), etc.). Reconfiguration concerns changes in the structure, functionality, and/or control algorithms to cope with external changes and user requirements (Zhang et al., 2015).

In spite of RDECSs's complexity, model-checking is the formal technique most used for its automatic verification. This efficient technique checks the satisfaction of a functional property specification, which is specified by a temporal logic (Baier et al., 2008) (computation tree logic) against a behavior formal model (Petri nets). Many formalisms are proposed and improved to cope with reconfigurability such as reconfigurable time net condition/event systems (R-TNCES) (Zhang et al., 2013) and reconfigurable Petri nets (Padberg and Kahloul, 2018). Several studies have worked on model checking improvement to be more efficient at errors debugging and in their auto-correction. Ding and Zhang in (Ding and Zhang, 2007) have proposed a method named computation tree logic *CTL update* that modifies the system model in order to satisfy a given formula, based on five basic operations and minimal change criteria. Martinez and Lopez have proposed another method named *CTL repair* for different classes of Petri nets such in: (Martínez-Araiza and López-Mellado, 2014), (Martínez-Araiza and López-Mellado, 2015), and (Martinez-Araiza and López-Mellado, 2016). Carrillo and Rosenblueth have introduced the protection concept to *CTL update* (Carrillo and Rosenblueth, 2014). Both the layer-by-layer verification proposed in (Zhang et al., 2013) and the formal verification proposed in (Hafidi et al., 2018) ignore the debugging phase treatments to automatize the correction of a model which does not satisfy a property formula. The convergence of this lack with the system complexity and the high number of properties to be checked makes R-TNCESs verification a hard task.

In the current paper, we assigned a parallel rebuilding to the analysis technique of an R-TNCES. The purpose of our proposition is to deal with the

[a] https://orcid.org/0000-0001-6311-3588

Ramdani, M., Kahloul, L. and Khalgui, M.
On Improving Parallel Rebuilding of R-TNCESs.
DOI: 10.5220/0007977903150322
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 315-322
ISBN: 978-989-758-379-7

big number of faulty properties with a parallel strategy to make a gain in the both rebuilding and validation time as well as to reduce the complexity of debugging process which relieves the design efforts. We deal with the auto-correction of R-TNCESs by proposing a new approach that includes verification and model updating according to several CTL formulas simultaneously (parallel rebuilding). The proposed methodology stands on the R-TNCES rebuilding approach (Ramdani. et al., 2019). In order to avoid unnecessary calculation, R-TNCES rebuilding will be improved by pretreatment and classification of properties that assure the best parallelization of auto-correction. In order to confirm results, SESA model checker is used (Starke and Roch, 2002). Indeed, SESA is a software to analyze TNCESs and to compute the exact reachable set of states. To validate the proposed methodology, we use academic case study FESTO MPS (Zhang et al., 2013) which is a lab-scale station. Experimentations on FESTO MPS show the applicability and performance of our proposition to check and rebuild properties of broadcasting and synchronizations simultaneously.

The paper is organized as follows. Section 2 presents preliminary concepts. Section 3 gives the R-TNCES rebuilding methodology. Improved parallel rebuilding is presented in Section 4. An academic case study is presented to show the virtue of the proposed approach and its algorithms in Section 5. Finally, Section 6 concludes this work and describes our perspectives.

## 2 BACKGROUND

In this section, we present a brief description of R-TNCESs and Ding's CTL update approach.

### 2.1 Reconfigurable Time Net Condition-Event Systems

R-TNCES is a control component based formalism proposed in (Zhang et al., 2013) to specify and verify reconfigurable discrete event control systems (RDECSs). A control component (CC) is a logical software unit that represents data flows and sensors/actuators actions (algorithms, data extraction or activation) (Khalgui and Hanisch, 2011). An R-TNCES is a structure $RTN = (B,R)$ composed of two modules, where $B = (Conf_1,\dots,Conf_n)$ is the behavior module formed by $n$ configurations ($Conf_i$), each one is a TNCES, possibly redundant and $R = (r_1,\dots,r_m)$ is a set of reconfiguration functions which represent the control module ($n,m \in \mathbb{N}$). Formally, $B$ is a tuple given by

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

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.

### 2.2 Computation Tree Logic Model Repair

"Computation tree logic model update" is a formal approach for automatic verification and modification of system models based on minimal change criteria over Kripke structure models. This approach was developed in (Ding and Zhang, 2007). To repair software errors, *CTL update* is deployed to generate admissible models that represent the correct design (Zhang and Ding, 2008). The model updater functions modify the models using five primitives ($PU1,\dots,PU5$) which are described in their simplest forms as: (i) $PU1$ for adding a relation, (ii) $PU2$ for removing a relation, (iii) $PU3$ for changing one state label of one state, (iv) $PU4$ for adding a state, and (v) $PU5$ for 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 PROBLEM FOR R-TNCESs MODELS

Given R-TNCES $M = (B,R)$ representing a system model and $\phi$ a CTL formula in existential normal form ENF representing a functional property such that $(M,Z_0) \nvDash \phi$. Rebuilding problem is the construction of a new model $M'$ by applying minimal changes to $M$ such that $(M',Z_0') \vDash \phi$. $M'$ must satisfy specification and conserve good requirements.

Automatic rebuilding of behavior module can be summarized in six step methodology as follows.

- *Step 1:* Transform behavior module $B$ to Kripke structure $SK$.
- *Step 2:* Transform formula $\phi$ to adapt the same value in the Kripke structure.
- *Step 3:* If $SK \vDash \phi$, then the model is well specified, otherwise, go to the next step.

- *Step 4:* Modify *SK* according to CTL model update approach (Zhang and Ding, 2008); (action to be supervised by the designer).
- *Step 5:* Re-check the satisfaction $SK \models \phi$. If $SK \models \phi$, then the model is well modified, otherwise, inconsistency in the specification (formula, model, or the both).
- *Step 6:* Rebuild *B* using primitives that are equivalent (see Table 1) to those executed in the $4^{th}$ step. Table 1 represents the equivalence between the primitives of CTL update and R-TNCES rebuilding.

R-TNCES rebuilding operation is well described and detailed in (Ramdani. et al., 2019).

In this work, we assume that the control part *R* and the modular entities (*CC*s) are not faulty and represent the desired behavior. Therefore, we focus on the behavior module response when a reconfiguration is requested or an error is occurred. In such context, rebuilding operation (*RO*) consists of checking the synchronization faults between transitions in different CCs (the broadcasting correctness).

Table 1: Equivalence between Ding primitives and R-TNCES rebuilding modifications (Ramdani. et al., 2019).

| CTL-update primitives | R-TNCES rebuilding modifications | Modification instructions |
|---|---|---|
| $PU1$ | 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 | $De(CC))$ |

## 4 IMPROVED PARALLEL REBUILDING

We propose in this section a parallel rebuilding for automatizing the correction of an R-TNCES according to a high number of faulty properties simultaneously. With the proposed approach, we can deal the deficiency of other works on automatizing the debugging of reconfigurable system model at its verification. The gain of our contribution resides in two major points. On one hand, an ordinary rebuilding of an R-TNCES with one CTL formula expands the classical CTL update to be dealing with reconfigurable system models. On the other hand, the parallel rebuilding of an R-TNCES makes the rebuilding more efficient to deal with the big number of properties in complex system models. Using the proposed method, we can minimize the validation time of a system model and we can also control the complexity of the debugging phase in a complicated system. We improve the R-TNCES rebuilding by adding a pre-processing of properties which identifies functional dependencies, precedence order and classifies them to extract the optimal order of the rebuilding process. The global process of parallel rebuilding reposes on four sub phases: (i) Abstraction: is the transformation of R-TNCES model (resp. CTL formula) to Kripke structure (resp. reduction of the granularity). (ii) Classification: is the detection of relationships among properties. It defines dominance, equivalence and composition among those properties. (iii) Rebuilding order is the construction of a tree that defines the order of rebuilding and the parallelization strategy. (iv) Parallelization: is the assignment of properties to the parallel environment. Figure 1 resumes the global process of parallel rebuilding.

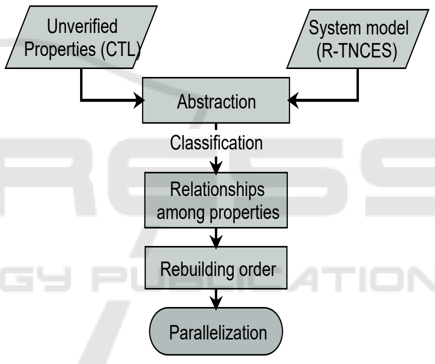


Figure 1: Parallel Rebuilding of R-TNCES.

### 4.1 Abstraction and Properties Classification

Abstraction subprocess reposes on two different and independent operations that assure the transformation of both model and formula. The first operation is the computation of a new Kripke structure model from behavior module *B*. The second one, is the granularity reduction of a CTL formula to adapt it for the Kripke structure verification.

In properties classification, we classify properties according to the precedence order and relationships among them. At the first step of this sub-process, we divide set of unverified properties $\sum\Phi$ into two different sets according to their functional purpose, which are defined as follows:

- **Universal Properties:** A set of properties that are expressed to check a functional property in the whole system (all configurations).

- **Local Properties:** A set of properties that are expressed to check a special functional property in one and only one execution in the system (just one configuration).

To avoid redoing the rebuilding operation, we must ensure the precedence order among properties. In this context, we can define the relations between two properties $\phi_i$ and $\phi_j$ as:
**Dominance:** We say $\phi_i$ dominates $\phi_j$ iff $\phi_i$ is universal and $\phi_j$ is local, or the state formula of $\phi_i$ becomes in front at the same configuration. In the general case, we denote the dominance by '$\Longrightarrow$'.

The classification of CTL properties, according to their semantic relationships (Ramdani et al., 2018), consists of extracting possible relationships among properties for guiding an efficient verification. The process starts by sorting properties according to local/global criteria. Then, each kind of properties will be arranged separately according to precedence order. Finally, we perform an analysis of dominance among properties to accomplish the global order of properties. Figure 2 resumes the analysis procedure of properties in the classification sub-process.

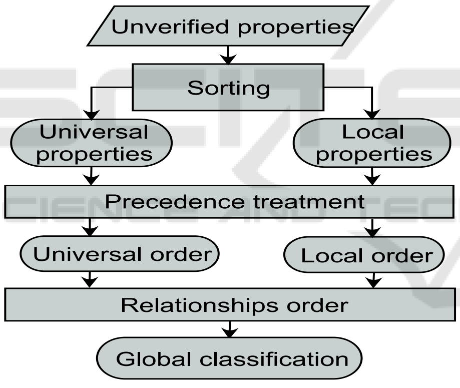


Figure 2: Properties classification sub-process.

## 4.2 Rebuilding Order and Parallelization

After classification and extraction of possible relationships between properties, the rebuilding order and its assignment in a parallel environment achieves the whole process. Let $\sum\pi_{i,i=1...n}$ be the set of traces in a given Kripke structure $Sk$ and let $m$ be the number of properties in $\sum\Phi$. To get an optimal parallelization and avoid unnecessary calculation, we define a matrix that resumes the properties appearance in the set of traces, i.e., for each state formula of each property, we localize which paths appear. Formally, the appearance matrix $AM[n,m] = [0/1]$ is given by

$$\forall\pi_i \in \sum\pi_{i,i=1...n}, \forall\phi_j \in \sum\Phi, \exists\psi_j \in \pi_i : AM[i,j] = [1]. \tag{2}$$

where, $\psi_j$ is state formula from $\phi_j$. Using this matrix, the properties will be ordered descendingly according to the number of paths where they had participated in.

The execution tree $ET$ is a graph, where each node denotes a task. The node content indicates the property to be checked. An arrow from one node to another indicates that the second node - the one at the end of the arrow - can only be executed when the first one has finished its execution. Nodes at the same level can be executed in parallel. Formally an execution tree $ET$ is given by

$$ET = (N,T) \tag{3}$$

where, $N$ is a set of nodes of execution, and $T$ is a set of transitions between nodes.

Using a hybrid scan between the matrix and the relationship among properties (supervised by designer), we build a tree of precedence and independence, applying the following steps.

- *Step 1:* From appearance matrix, we extract a descending order of properties according to the number of appearances in paths.
- *Step 2:* From the relationships set of properties, we extract the dominance among properties.
- *Step 3:* Using Algorithm 1, we coordinate properties order and dominance relationships among them to generate the execution tree $ET$. In order to respect the model coherency, each dominant property will be rebuild separately.

**Algorithm 1: Execution Tree Generation:** generates a node for each independent property and for each couple of properties (dominant, dominated), the algorithm creates a couple of nodes (father, son) in the

Algorithm 1: Execution tree generation.

```
Input: $\sum\Phi$+ Relationships classification;
Output: $ET$;
  Create Node($Idl$, _);
  for $i$=1 to $i$=$n$ do
    if ($\phi_i$ is not dominated) then
      Create Node($Idl$, $\phi_i$);
    end
    else
      int $temp = 0$;
      for ($j = 1; j \leqslant n; i++$) do
        if ($\phi_j$ domines $\phi_i$) then
          $temp \leftarrow j$;
        end
      end
      Create Node($\phi_j$, $\phi_i$);
    end
  end
  Return ($ET$);
```

execution tree (all nodes have the same precedence will have just one father.).

Given a parallel environment $PE$ which is composed of a set of $n$ execution units $EU$ (cores). $EU$s preform parallel rebuilding calculation respecting the mutual exclusion. $ET$'s nodes will be assigned level by level in the $PE$, i.e., in each level, each node is assigned to one $EU$ from left to right with the respect of the availability of the $PE$. Formally, $PE$ is given by

$$PE = (EU, ET, cond) \tag{4}$$

where, $EU$ is a set of execution units in a parallel machine, $ET$ is a the execution tree, and $cond : EU \rightarrow (True, False)$ is the condition of execution in the parallel environment, initially declared $False$.

By the proposed parallel rebuilding of R-TNCES, automatic analyses techniques as model checking can be more efficient to verify a complex model of reconfigurable systems. Parallel rebuilding makes debugging a practical process to converge a reconfigurable system model to meet several functional properties simultaneously and with a simple way.

## 5 EXPERIMENTATIONS

### 5.1 Case Study

In order to validate and demonstrate the gain of the proposed approach, we use FESTO modular production system (MPS) (Zhang et al., 2013), which is a lab-scale production line simulating different functions of a manufacturing system. FESTO is composed of 12 physical processes, organized into four operational configurations. Each physical process is modeled by one control component ($CC$) and each configuration $Conf_i$ has one control chain ($C_{chain_i}$). The control chains are well described in (Ramdani. et al., 2019). Figure 3(a) depicts a faulty model of behavior module of MPS modeled by an R-TNCES. To assure the production quality of a good product, once the workpiece is rejected by unit quality test (quality test unit checks: color, material, and height of workpieces), this workpiece can not be drilled in next steps. In particular, we need to ensure that each configuration is safe and can proceed normally (liveness of model). Each control chain must produce one product. To check those behaviors, we use CTL formulas written in the ENF form presented in Table 2. All formulas are proven to be $False$, we apply a parallel rebuilding on the behavior module to get a new correct one that respects the functional properties expressed, previously. First, we need to compute Kripke structure $SK$ from the above behavior module. The result is shown in Figure 3(b). Second, formulas presented in Table 2 are abstracted to be expressed on $SK$. Those formulas become: $\phi_1 = EG(cc_3 \wedge EGcc_7)$, $\phi_2 = EG(cc_3 \wedge EGcc_8)$, $\phi_3 = EG(cc_3 \wedge EGcc_{11})$, $\phi_4 = EG(cc_3 \wedge EGcc_{12})$, and $\phi_5 = \neg EF(cc_4 \wedge EGcc_{10})$. According to the functional purpose, $\phi_5$ is universal and the rest of properties are local. According to the precedence order, we preform the following classification of dominance relationships among properties: $\phi_5 \Longrightarrow \phi_1$, $\phi_5 \Longrightarrow \phi_2$, $\phi_5 \Longrightarrow \phi_3$, $\phi_5 \Longrightarrow \phi_4$, and $\phi_1 \Longrightarrow \phi_2$.

Table 2: CTL based Functional properties.

| CTL formula | Normalization in the ENF form |
|---|---|
| $\phi_1$=$AF(p_9 \rightarrow AFp_{20})$ | $\phi_1$=$EG(p_9 \wedge EGp_{20})$ |
| $\phi_2$=$AF(p_9 \rightarrow AFp_{26})$ | $\phi_2$=$EG(p_9 \wedge EGp_{23})$ |
| $\phi_3$=$AF(p_9 \rightarrow AFp_{32})$ | $\phi_3$=$EG(p_9 \wedge EGp_{32})$ |
| $\phi_4$=$AF(p_9 \rightarrow AFp_{36})$ | $\phi_4$=$EG(p_9 \wedge EGp_{36})$ |
| $\phi_5$=$AG(p_{12} \rightarrow AF\neg p_{29})$ | $\phi_5$=$\neg EF(p_{12} \wedge EGp_{29})$ |

Using $\sum \pi_{i,i=1\dots6}$ presented in Figure 4(a) and unverified set of properties $\sum\Phi$, we build appearance matrix $AM$ presented in Figure 4(b).

Let $PE$ denotes a parallel environment with 4 execution units $EU_i, i = 1\dots4$. We generate the execution tree of the above properties. Figure 5(a) depicts the generated execution tree by using Algorithm 1 and Figure 5(b) shows its assignment in $PE$.

At $Level_1$, only $EU_1$ is activated to update $SK$ according to $\phi_5$, and this operation can be deployed by deleting output relations from $s_4$ using primitive $PU2$ to remove the relation between $s_4$ and $s_9$ (resp. between $s_4$ and $s_5$). Then, in the next level, $EU_1$, $EU_2$, and $EU_3$ are activated to rebuild $SK$ according to $\phi_1$, $\phi_2$, and $\phi_3$ respectively. Using primitive $PU1$, $EU_1$ will add a relation between $s_3$ and $s_5$, $EU_2$ will add a relation between $s_6$ and $s_{11}$, and $EU_3$ will add a relation between $s_{12}$ and $s_9$. Finally, at the third level, $EU_1$ is activated to rebuild $SK$ according to $\phi_4$ (this property is auto corrected by $EU_1$ in the second level). Figure 7(a) depicts new structure $SK'$ after rebuilding. To get the new correct behavior module, we regenerate the equivalent R-TNCES of $SK'$, using modification instruction. Results are shown in Figure 7(b).

Using SESA model checker (Starke and Roch, 2002), an exact reachability graph of the FESTO MPS's behavior model is computed. The obtained graph is finite and it contains 57926 reachable states. We check that the new model satisfies the above properties and its parallel rebuilding is done without system degradation. All properties are proven to be True as shown in Figure 6.

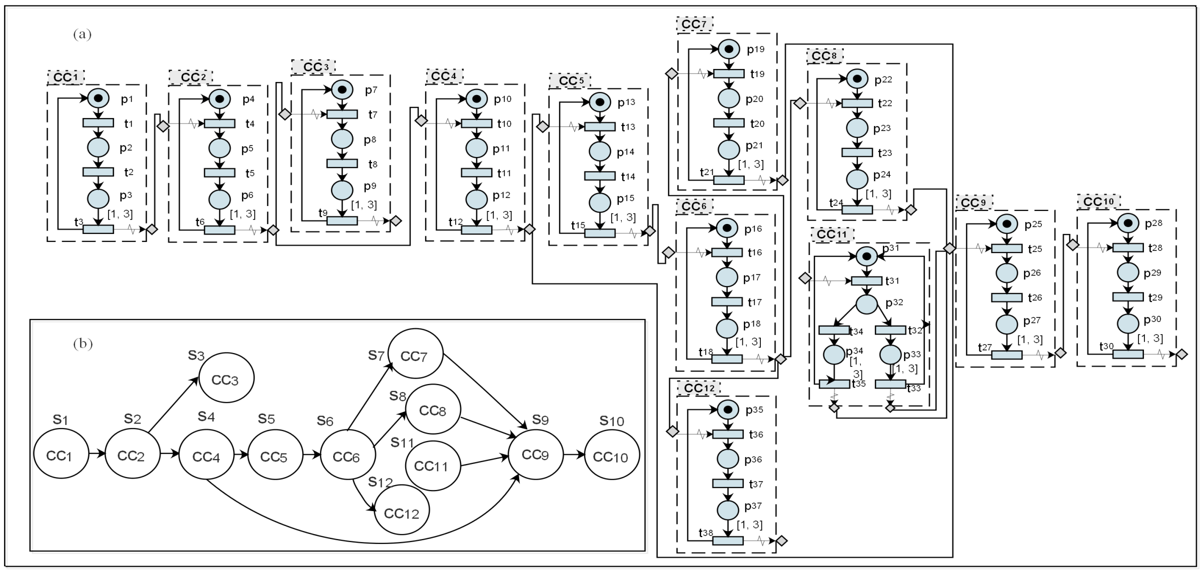


Figure 3: MPS's behavior module *B* and its computed *SK*.

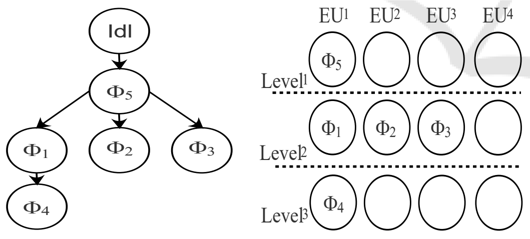
- $\pi_1 = S_1, S_2, S_3$.
- $\pi_2 = S_1, S_2, S_4, S_5, S_6, S_7, S_9, S_{10}$.
- $\pi_3 = S_1, S_2, S_4, S_5, S_6, S_8, S_9, S_{10}$.
- $\pi_4 = S_1, S_2, S_4, S_5, S_6.S_{12}$
- $\pi_5 = S_1, S_2, S_4, S_9, S_{10}$.
- $\pi_6 = S_{11}, S_9, S_{10}$.

(a)

$$AM = \left(\begin{array}{c|ccccc} & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 \\ \pi_1 & 1 & 1 & 1 & 1 & 0 \\ \pi_2 & 1 & 0 & 0 & 0 & 1 \\ \pi_3 & 0 & 1 & 0 & 0 & 1 \\ \pi_4 & 0 & 0 & 0 & 1 & 1 \\ \pi_5 & 0 & 0 & 0 & 0 & 1 \\ \pi_6 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$$

(b)

Figure 4: Properties appearance in the set of traces of *Sk*.

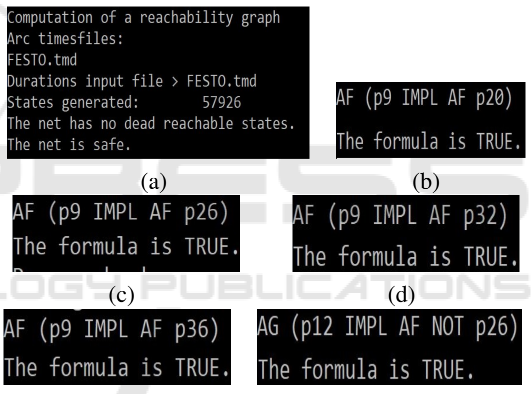


(a) (b)

Figure 5: Execution tree and its affectation.

## 5.2 Performance Evaluation

Let us assume that we have a big system model to be rebuilt according to 100 faulty properties, and let us assume that the average dominance among properties can be in, 0-25%, 25-50%, 50-75%, and 75-100% respectively. Let us suppose that we have four parallel execution units, each one can rebuild one property in a one time unit. A quantitative difference between execution time for sequential rebuilding of faulty properties and parallel rebuilding. The parallel rebuilding approach avoids unnecessary wait time in execution by avoiding redundancies and imposing order of relations among properties. The gain with this approach is clearly shown in the rebuilding of properties which have a less dominance relationships among them (i.e., Independent properties can be rebuilt parallely without any waiting time). As shown in Figure 8, rebuilding time is proportional to the degree of dominance relations between properties.



```
Computation of a reachability graph
Arc timesfiles:
FESTO.tmd
Durations input file > FESTO.tmd
States generated:        57926
The net has no dead reachable states.
The net is safe.
```

(a)

```
AF (p9 IMPL AF p20)
The formula is TRUE.
```

(b)

```
AF (p9 IMPL AF p26)
The formula is TRUE.
```

(c)

```
AF (p9 IMPL AF p32)
The formula is TRUE.
```

(d)

```
AF (p9 IMPL AF p36)
The formula is TRUE.
```

(e)

```
AG (p12 IMPL AF NOT p26)
The formula is TRUE.
```

(f)

Figure 6: Screen shots of SESA model-checking results.

Number of properties to be rebuilt is directly related to the system size and its number of TNCESs. Let assume that the correctness of each TNCES must be checked by three properties (safety, liveness, and non-deadlock). The average of those properties is faulty and needs rebuilding. Let us assume a variety of systems that have 50, 100, 200, 500 and 1000 TNCESs and each case have 75, 150, 300, 750, and 1500 faulty properties respectively with a rate of 25% dominance among them. A rebuilding of one property

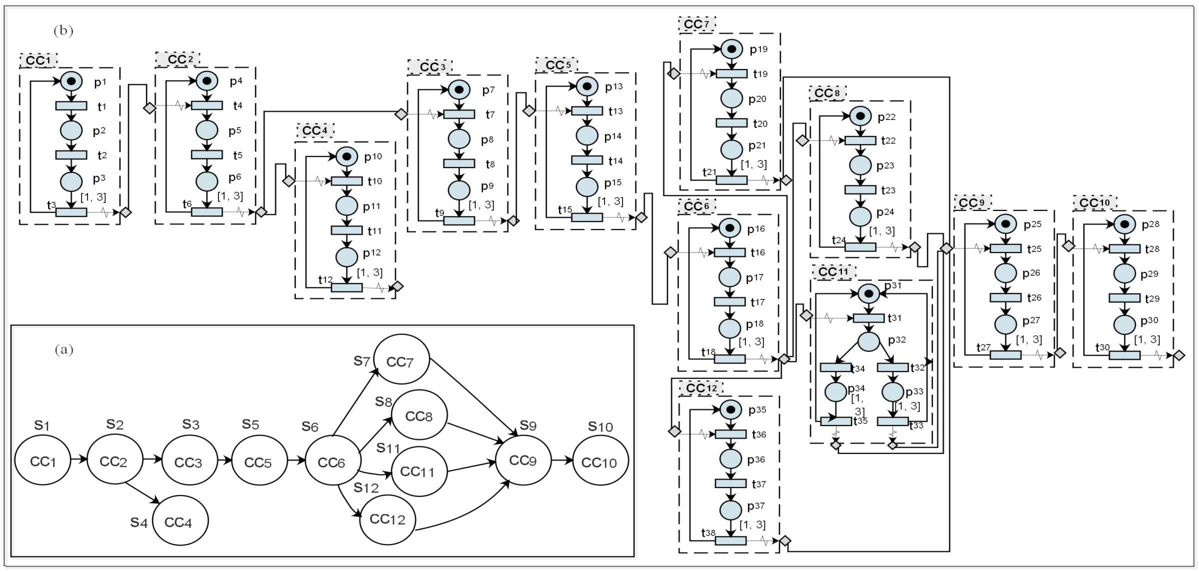


Figure 7: $SK'$ and the behavior module $B$ of MPS after rebuilding.

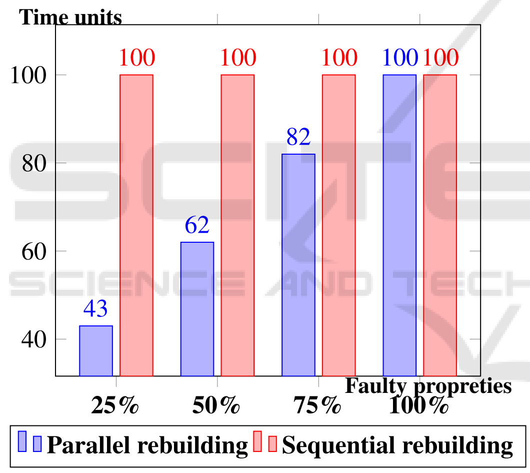


Figure 8: Sequential rebuilding VS Parallel rebuilding.

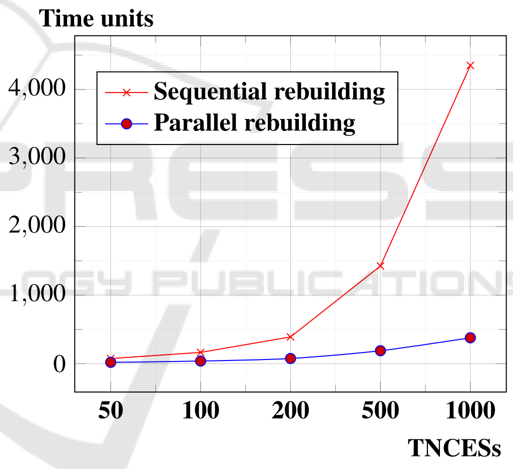


Figure 9: Improved performance of Parallel rebuilding.

in a system less than 50 TNCESs needs one time unit. Indeed, for each 50 TNCES additional in the system size need 10% additional in the execution time.

## 5.3 Discussion

For reconfigurable systems, there is no study dealing with parallel 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. Table 3 describes a short qualitative comparison between the proposed contribution in our paper and the most recent related works.

## 6 CONCLUSIONS

This work deals with the parallel auto-correction of reconfigurable systems modeled with R-TNCES formalism according to CTL-based functional properties. In this work, we have presented a parallel rebuilding approach for reconfigurable timed net condition/event systems (R-TNCESs). Our contribution reposes on two fundamental techniques: (i) An abstraction of R-TNCES model (resp. its CTL-based functional formulas) to a Kripke structure (resp. to a simple CTL formulas with the same verification value

Table 3: Qualitative comparison with some related works.

| Works | Used formalisms | Reconfiguration | Model repair | Execution |
|---|---|---|---|---|
| (Ding and Zhang, 2007)<br>(Zhang and Ding, 2008), | Kripke structure | No | Yes | Sequential |
| (Carrillo and Rosenblueth, 2014). | Kripke structure | No | Yes | Sequential |
| (Martínez-Araiza and López-Mellado, 2014),<br>(Martínez-Araiza and López-Mellado, 2015),<br>(Martinez-Araiza and López-Mellado, 2016). | Petri nets | No | Yes | Sequential |
| (Zhang et al., 2013) | R-TNCESs | Yes | No | - |
| (Hafidi et al., 2018) | R-TNCESs | Yes | No | - |
| (Ramdani. et al., 2019) | R-TNCESs | Yes | Yes | Sequential |
| Our work | R-TNCESs | Yes | Yes | Parallel |

expressed on the new Kripke structure). (ii) A technique to extract the precedence order between properties and to classify the relations of dominance among them. This technique helps to build an execution tree to be executed in a parallel environment. Therewith, we define an equivalence between Ding primitives and R-TNCESs rebuilding modification instructions. At the end, we confirm the obtained results of parallel R-TNCESs rebuilding operation by an experimental case study. Contrary to what exists, our approach updates the system model directly and simultaneously, which results in the gain of design effort and thus reduces the verification time. Whereas classically, the designer has to repeat the verification debugging cycle for each violated functional property separately. This work opens several possible perspectives. First, we plan to apply our approach to real large case studies. Second, we plan to take into consideration more details of CTL formulas. Finally, we plan to consider reconfigurable systems with distributed behaviors.

## REFERENCES

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

Ben Aissa, Y., Bachir, A., Khalgui, M., Koubaa, A., Li, Z., and Qu, T. (2019). On feasibility of multichannel reconfigurable wireless sensor networks under real-time and energy constraints. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–16.

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.

Gu, C., Li, Z., Wu, N., Khalgui, M., Qu, T., and Al-Ahmari, A. (2018). Improved multi-step look-ahead control policies for automated manufacturing systems. *IEEE Access*, 6:68824–68838.

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.

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.

Khalgui, M. and Mosbahi, O. (2010). Intelligent distributed control systems. *Information and Software Technology*, 52(12):1259 – 1271.

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.

Ramdani, M., Kahloul, L., and Khalgui, M. (2018). Automatic properties classification approach for guiding the verification of complex reconfigurable systems. In *Proceedings of the 13th International Conference on Software Technologies - Volume 1: ICSOFT,*, pages 591–598. INSTICC, SciTePress.

Ramdani., M., Kahloul., L., Khalgui., M., and Hafidi., Y. (2019). R-TNCES rebuilding: A new method of ctl model update for reconfigurable systems. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,*, pages 159–168. INSTICC, SciTePress.

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

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.