# Vérification des systèmes modulaires

F. Ouazar Lounnaci M.Ioualalen M.C.Boukala

MOVEP, USTHB, Algérie

fouazar@gmail.com mioualalen@usthb.dz mboukala@usthb.dz

## Résumé

La vérification des systèmes par model-checking est souvent confrontée au problème de l'explosion combinatoire de l'espace d'états du système. La modularité est l'une des approches qui permet de pallier à ce problème. Cette approche permet de représenter et de vérifier des systèmes de grande taille sous forme de modules. Dans ce travail, nous nous intéressons à la vérification des propriétés spécifiques exprimées en logique temporelle CTL sur un réseau de Petri modulaire. Nous avons alors proposé des algorithmes de vérification des propriétés CTL sur des systèmes modélisés par un réseau de Petri modulaire.

## I. INTRODUCTION

Les systèmes informatiques ont pris une grande importance dans la vie de la société moderne. Ils sont utilisés pour effectuer des contrôles, tâches et fonctions critiques **[1]** (centrale nucléaire, l'aéronautique au travers des systèmes de pilotage embarqués: avions, satellites...). Il est donc impératif d'avoir des outils et techniques de modélisation et de vérification qui puissent garder un haut degré de performance et de correction pour ces systèmes. Dans ce cadre, on peut profiter de la puissance des réseaux de Petri (RdPs), pour effectuer une modélisation et une vérification du système sur le plan logique du séquencement des événements.
Les RdPs permettent de vérifier les propriétés générales du système telle que : la bornitude, la vivacité, le non blocage…, celles-ci doivent être complétées par l'analyse de propriétés spécifiques du système modélisé.
Plusieurs axes de recherche ont été développés visant à mettre en place des outils permettant une description formelle des propriétés spécifiques des systèmes informatiques. La logique temporelle initiée par EMERSON & AL **[4],** est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives.

Après la description des propriétés spécifiques, vient l'étape vérification de ces propriétés sur le système. Cette étape consiste à vérifier si celui-ci satisfait ses spécifications, c'est-à-dire de s'assurer que les propriétés qu'il possède correspondent bien aux propriétés attendues. Parmi les techniques de vérification, les approches par model-checking, se sont largement développées ces dernières années **[5].** Cela suppose le comportement du système à vérifier modélisé et que la propriété de correction attendue est énoncée sous forme d'une formule de logique temporelle, ensuite, on peut utiliser un model-checker afin de vérifier si le modèle satisfait ou non la propriété.

Néanmoins, la vérification et la validation des systèmes, qui sont souvent basées sur l'exploration de l'espace d'états, souffrent de l'explosion combinatoire de la taille de l'espace d'états qui croit de façon exponentielle.
Plusieurs approches ont été proposées pour pallier à ce problème. Parmi ces approches, les techniques de réduction, telles que la vérification à la volée, qui vérifient les propriétés en même temps que l'exploration, la représentation symbolique tels que les diagrammes de décision, la distribution, et celles qui évitent de représenter tous les états du système en prenant en compte certaines propriétés du modèle telles que la symétrie **[6]**, la vérification modulaire **[7].**

Dans le cadre de la vérification modulaire, on trouve le travail de **[3]**, dans lequel les auteurs s'intéressent à la vérification des propriétés exprimées en logique temporelle LTL, on trouve aussi le travail de **[11]** dans lequel les auteurs s'intéressent qu'aux propriétés générales telle que la vivacité, l'accessibilité…,

Dans ce papier, nous nous intéressons à la vérification des propriétés spécifiques exprimées en logique CTL des systèmes modélisés par les RdPs modulaires tels que définis dans **[2]** (avec transition de fusion) pour éviter le problème de l'explosion combinatoire en générant l'espace d'états de chaque module à part, avec un graphe de synchronisation permettant de déduire les états et les comportements globaux.
Nous utilisons une technique basée sur l'idée de vérifier si la propriété est vérifiée par chaque module ainsi que sur le graphe de synchronisation pour décider si elle est satisfaite par le système.

Dans le reste de ce document, nous présentons le principe des réseaux de Petri modulaires dans la section 2, la logique temporelle CTL dans la section 3. La section 4 portera sur la présentation de la technique de vérification par model-checking des propriétés CTL sur des systèmes modulaires et nous terminerons par une conclusion et quelques perspectives.

## II. LES RESEAUX DE PETRI MODULAIRES

Nous considérons les réseaux de Petri modulaires avec partage de transitions définis dans **[2]**.

**Définition : Un réseau de Petri modulaire** est un couple MN = (S, TF) tel que :

- S est un ensemble fini de modules tel que :
  - Chaque module $s \in S$ est un réseau de Petri $s = (P_s, T_s, W_s, M_{s0})$.
  - L'ensemble des nœuds des différents modules sont deux à deux disjoints :
    $\forall s1, s2 \in S : [s1 \neq s2 \Rightarrow (Ps1 \cup Ts1) \cap (Ps2 \cup Ts2) = \emptyset]$.
    $P=\cup_{s\in S}(Ps)$ et $T =\cup_{s\in S}(Ps)$, Ps et Ts constituent respectivement l'ensemble des places et l'ensemble des transitions de tous les modules du système.
  - $TF \subseteq 2^T$ est un ensemble fini non vide d'**ensembles de transitions de fusion**.

Dans ce qui suit TF désignera aussi $\cup_{tf\in TF}$ tf .

**Définition :** Un **groupe de transitions** $tg \subseteq T$ consiste soit en une transition qui n'est pas une transition de fusion $t \in T \setminus TF$, soit en l'ensemble des éléments d'un ensemble de fusion de transitions $tf \in TF$.
L'ensemble des groupes de transitions est noté TG.
Une transition peut être élément de plusieurs groupes de transitions puisqu'elle peut être synchronisée avec différentes transitions. Ainsi, un groupe de transitions correspond à une action synchronisée.
La fonction de poids des arcs W est étendue aux groupes de transitions :
$\forall p \in P, \forall tg \in TG, W(p, tg) =\sum_{t\in tg} W(p,t)$ , $W(tg, p) =\sum_{t\in tg} W(t,p)$

Le marquage d'un réseau de Petri modulaire est défini de la même manière que dans un réseau de Petri ordinaire, sur l'ensemble des places de tous les modules. La restriction d'un marquage *M* à un module *s* est notée *Ms*. Les règles de sensibilisation et de franchissement des transitions d'un réseau de Petri modulaire peuvent alors être exprimées.

**Définition : Le franchissement d'un groupe de transitions : tg** est sensibilisée pour un marquage M, noté par M[tg⟩, si et seulement si :
$\forall p \in P : W(p, tg) \leq M(p)$
Lorsqu'un groupe de transitions tg est sensibilisé pour un marquage $M_1$, il peut être franchi, donnant le nouveau marquage $M_2$, défini par :
$\forall p \in P : M_2(p) = (M_1(p) - W(p, tg)) + W(tg, p)$

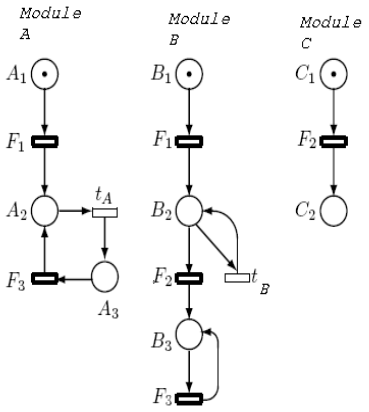


*figure1: un RdP modulaire avec 3 modules*

La figure1 illustre un réseau de Petri modulaire qui consiste en trois modules A, B et C. Les modules A et B comportent tous les deux les transitions étiquetées F1 et F3, tandis que B et C contiennent la transition F2. Ces transitions sont alors considérées comme étant des transitions de fusion et forment l'ensemble des transitions de fusion.

## Génération de l'espace d'états d'un RdP modulaire

L'espace d'états modulaire consiste en deux parties : les espaces d'états des différents modules et le graphe de synchronisation.

**Définition :** Soit MN = (S, TF), un réseau de Petri modulaire avec le marquage initial M0. L'**espace d'états modulaire** de MN est un couple $MSS = ((SS_s)_{s\in S}, SG)$ où :

I. $SS_s = (Vs, As)$ est l'espace d'états local du module s :

(a) $Vs=\cup_{v\in VSG} [s\rangle s$

(b) $As = \{(M1, t, M2) \in Vs \times (T \setminus TF)s \times Vs \mid M1[t\rangle M2\}$

II. $SG = (V_{SG}, A_{SG})$ est le **graphe de synchronisation** de MN :

(a) $V_{SG} = [[M_0\rangle\rangle^c \cup M_0{}^c$

(b) $A_{SG} = \{(M_1{}^c , (M_1{}'^c , tf ), M_2{}^c ) \in V_{SG} \times ([M_0\rangle^c \times TF) \times V_{SG} \mid M'_1 \in [[M_1\rangle\wedge M'_1[tf\rangle M_2\}$

Le graphe de synchronisation comporte les informations sur les nœuds atteignables lors du franchissement des transitions de fusion. Les graphes des espaces d'états des modules comportent donc des informations locales, c'est-à-dire les marquages des modules et les arcs correspondants aux transitions internes mais pas les transitions de fusion.
Les nœuds du graphe de synchronisation représentent tous les marquages accessibles à partir d'autres marquages en franchissant une séquence de transitions internes suivie d'une transition de fusion. Le marquage initial est aussi représenté. Les arcs du graphe de synchronisation représentent les occurrences des transitions de fusion. Les étiquettes des arcs du graphe de synchronisation comportent le nom de la transition franchie, ainsi que sa source.

**Exemple**

L'espace d'états modulaire du réseau de Petri modulaire de la figure1 est donné dans la figure 2

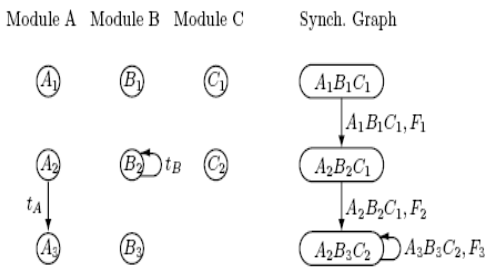


Figure2

**Intérêt des RdP modulaires**

Les RdP modulaire permettent de représenter des systèmes de grande taille, des cas pratiques ont été illustrés dans **[12]**.

# III. LA LOGIQUE TEMPORELLE CTL

La logique temporelle est une logique modale introduite par Pnueli **[10]**, afin de décrire et de spécifier au moyen d'expressions axiomatiques, des propriétés qualitatives et quantitatives de systèmes.
Une logique temporelle est un langage permettant de vérifier certaines propriétés spécifiques des systèmes, comme la propriété d'exclusion mutuelle. La syntaxe de ce langage est basée essentiellement sur une grammaire procurant une spécification sûre, précise et sans ambiguïté des propriétés temporelles. Elle est composée de la logique propositionnelle (formée à l'aide de propositions c.-à-d. des énoncés susceptibles d'être vrais ou faux) à laquelle sont ajoutés des opérateurs temporels linéaires qui sont essentiellement la potentialité F (éventuellement dans le futur), l'invariance G (toujours dans le futur) la précédence U (jusqu'à dans le futur {Until}), et les quantificateurs A (pour tout chemin) et E (pour certains chemins).

**Définition : La logique CTL** est une logique temporelle arborescente, introduite par Clarke et Emerson. Elle est destinée à la spécification des systèmes concurrents à états finis.
Elle permet d'exprimer les propriétés usuelles des systèmes telles que la sûreté (fiabilité) et la vivacité.

**Syntaxe de CTL [Eme96] :** CTL est formée des règles S1, S2, S3 et P0.
S1 : chaque proposition atomique P est une formule d'état.
S2 : si *f* et *g* sont des formules d'état alors ($f \wedge g$) et ($\neg f$ ) sont des formules d'état.
S3 : si *f* est une formule de chemin alors *E f* et *A f* sont des formules d'état.
P0: si *f* et *g* sont des formules d'états alors *Xf* et *fUg* sont des formules de chemin.

La logique CTL se focalise sur la notion d'état. En effet, on peut décrire la syntaxe sans définir les formules de chemin à l'aide des quatre opérateurs:
***AXf*** : pour tout état successeur de l'état considéré, *f* est vérifiée
***EXf*** : il existe un état successeur de l'état considéré pour lequel *f* est vérifiée.
***AfUg*** : pour toute séquence issue de l'état considéré, *f* est vérifiée jusqu'à ce que *g* le soit.
***EfUg*** : il existe une séquence issue de l'état considéré telle que *f* est vérifiée jusqu'à ce que *g* le soit.

**Quelques abréviations de CTL** : Soit f une formule d'état, la sémantique de CTL est définie comme suit :
La formule ***EFf*** dit qu'il existe une exécution pour laquelle f est nécessairement vraie. Cette propriété est connue sous le nom de la potentialité (figure 3.b).
La formule ***AFf*** signifie que, pour toute exécution possible, on atteindra un état qui vérifie f (f est nécessairement vraie).C'est la propriété d'inévitabilité (figure 3.a ).
La formule ***EGf*** spécifie qu'il existe une exécution pour laquelle f reste toujours vraie. C'est la propriété de quasi-invariance (figure 3.d).
La formule ***AGf*** signifie que pour toute exécution possible, la propriété f est vraie sur tous les états de cette exécution (f reste toujours vraie). C'est la propriété d'invariance (figure 3.c).
La combinaison de ces opérateurs permet d'exprimer des propriétés plus ou moins complexes.

**Exemple**
Soit la proposition pi désignant l'évènement "le processus i demande la section critique" et qi l'évènement "le processus i entre effectivement en section critique".
La formule ***AG*** (pi⇒ ***AF***qi) permet de spécifier la propriété de vivacité "tout processus demandant la section critique l'obtiendra nécessairement).

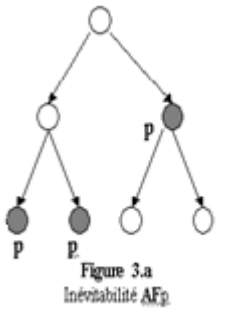


Figure 3.a
Inévitabilité AFp

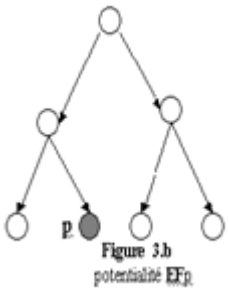


Figure 3.b
potentialité EFp

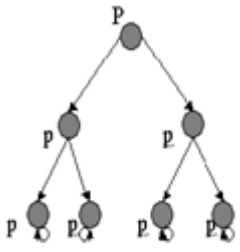


Figure 3.c
Invariance AGp

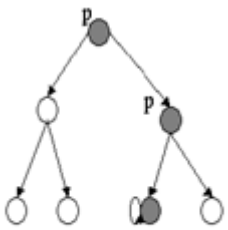


Figure 3.d
quasi-invariance EGp

**Sémantique de CTL :**

L'interprétation des formules CTL est définie sur les structures de kripke :
**Définition**: une structure Kripke est un 4-uplets M = (S;T;λ; $s_0$) tel que:

- S est un ensemble fini d'état,
- T ⊆ S×S associe chaque état s ∈ S avec ses successeurs
- λ : S ⇒ $2^{AP}$, associe à chaque état s ∈ S un ensemble de propositions atomiques valides dans s.
- $s_0$ ∈ S est l'état initial.

Soit $p \in AP$ une proposition atomique, $M = (S, T, \lambda, s_0)$ une structure de Kripke, $\delta = (s_0, s_1, ...)$ une séquence d'exécution de M, $s_0 \in S$, $\varphi$ et $\psi$ des formules CTL. La relation de satisfaction $|=$ est définie inductivement par :

$s |= p$ ssi $p \in \lambda(s)$
$s |= \neg\varphi$ ssi $\neg(s |= \varphi)$
$s |= \varphi \vee \psi$ ssi $(s |= \varphi) \vee (s |= \psi)$
$s |= EX\varphi$ ssi $\exists\delta$ telle que $\delta(0) = s$, $\delta(1) |= \varphi$
$s |= AX\varphi$ ssi $\forall\delta$ telle que $\delta(0) = s$, $\delta(1) |= \varphi$
$s |= E[\varphi \cup \psi]$ ssi $\exists\delta$ telle que $\delta(0) = s$ et $\exists j \geq 0$, $\delta(j) |= \psi \wedge (\forall 0 \leq k < j, \delta(k) |= \varphi)$
$s |= A[\varphi \cup \psi]$ ssi $\forall\delta$ telle que $\delta(0) = s$ et $\forall j \geq 0$, $\delta(j) |= \psi \wedge (\forall 0 \leq k < j, \delta(k) |= \psi')$

Dans ce travail, l'interprétation et la vérification des propriétés CTL se fait sur le graphe des marquages accessibles de chaque module et sur le graphe de synchronisation.

# IV. LE MODEL-CHECKING

Le but de la vérification est d'assurer qu'un système satisfait un certain nombre de propriétés. Pour cela, il est nécessaire de modéliser formellement le comportement de ce système. Rappelons que le système est modélisé par un RdP modulaire, le graphe de synchronisation et l'espace des états de chaque module décrivent l'évolution de ce système. Une fois le système décrit, et les propriétés souhaitées spécifiées en logique temporelle (CTL), l'algorithme dit model-checking permet de répondre automatiquement à la question : «est-ce que le système satisfait les propriétés souhaitées ?». A cet effet, nous avons proposé les algorithmes ci-dessous pour les différents opérateurs de la logique CTL (X,U,…)

Soit un réseau de Petri modulaire MN= (SS, TF) avec n modules ($S_i$, $i \in \{1..n\}$), Anc = ensemble des ancêtres, sat(φ)= ensemble des états satisfaisant φ

## Model-checking de φ=p

Le but de l'algorithme est de détecter tous les états qui satisfont φ, le principe est le suivant :

- voir sur le module auquel p appartient, et étiqueter les états satisfaisant p par φ.
- chercher les ancêtres de ces états.
- marquer (étiqueter) par φ, chaque état contenant un ancêtre dans le graphe de synchronisation.
- ensuite étiqueter la projection de chaque état marqué du graphe de synchronisation sur tous les modules composant ces états.
- enfin si l'état initial de chaque module est étiqueté par la formule φ alors on déduit que la formule ou la propriété est vraie (vérifiée) sinon la formule est fausse.

==========
Procédure de vérification check(φ=p)
==========
Begin
Anc=∅, sat(φ)=∅
//rechercher dans le module $S_i$ auquel p appartient si p est marqué
Foreach $M \in S_i$ do
If $p \in \lambda(M)$ then sat(φ)←sat(φ)∪{M}; Mark M par φ
Else return false
// chercher les ancêtres des états M marqués par φ
Foreach $s \in$ sat(φ)
Anc = Mark_ancestorSCC (s)

//procéder a l'étiquetage des états du graphe de synchronisation qui contiennent les états de Anc
For all $ss_i \in S$ //($ss_i$ etat du graphe de synchronization)
For all $z \in$ Anc
if ($z \in ss_i$)
mark $ss_i$ par φ
For all $ss_i$ marked by φ
mark $\Pi(M_{Si})$ by φ (marquer la projection de l'état du graphe de synch dans les modules)
For all $S_i$ (i=1..n) //pour tous les modules
If ($M_{0i}$ is marked by φ) //si l'état intitial de chaque module est marqué par φ
return true //property is true
Else return false //property is false
end

==========

**Exemple**

Vérification de φ=A1 (place A1 est marquée) sur le graphe de la figure 2

1- vérifier sur le module auquel A1 appartient si A1 est marquée → module A, A1 est marqué.
2- chercher les ancêtres de cet état→ état initial et il n'a pas d'ancêtre donc ensemble des ancêtres =état lui-même.
3- voir sur le graphe de synchronisation les états qui contiennent A1 → A1B1C1 marqué par φ.
4- étiqueter la projection A1B1C1 sur les modules → B1 étiqueté par φ et C1 étiqueté par φ
5- voir si l'état initial de chaque module est étiqueté par φ → oui, donc propriété vérifiée sur ce system.

## Model-checking de φ=¬ψ

Le principe de cet algorithme est le suivant :

- voir sur les module auxquels ψ appartient, et étiqueter les états qui ne satisfont pas ψ par φ.
- chercher les ancêtres de ces états.
- marquer (étiqueter) par φ, chaque état contenant un ancêtre dans le graphe de synchronisation.
- ensuite étiqueter la projection de chaque état marqué du graphe de synchronisation sur tous les modules composant ces états.
- enfin si l'état initial de chaque module est étiqueté par la formule φ alors on déduit que la formule ou la propriété est vraie (vérifiée) sinon la formule est fausse.

==========
Procédure de vérification check(φ=¬ψ)
==========
Begin
//pour tous les modules auxquels ψ appartient
For all $S_i \in SS$ (i∈{1..n}) do
Foreach $M \in S_i$ do
If($M \notin$ sat(ψ)) then sat(φ)← sat(φ)∪{M}, Mark M , Anc=Mark_ancestorSCC(M) by φ

For all $ss_i \in S$ //($ss_i$ etat du graphe de synchronisation)
For all $z \in Anc$
if ($z \in ss_i$)
mark $ss_i$ par φ
For all $ss_i$ marked by φ
mark $\Pi(M_{Si})$ by φ (marquer la projection de l'état du graphe de synch dans les modules)
For all $S_i$ (i=1..n) //pour tous les modules
If ($M_{0i}$ is marked by φ) //si l'état initial de chaque module est marqué par φ
return true //property is true
Else return false //property is false

End
==========

## Model-checking de φ=φ1∨φ2

Le principe de cet algorithme est le suivant :

- voir sur les modules auxquels φ1 ou φ2 appartiennent, et étiqueter les états satisfont φ1 ou φ2 par φ.
- chercher les ancêtres de ces états.
- marquer (étiqueter) par φ, chaque état contenant un ancêtre dans le graphe de synchronisation.
- ensuite étiqueter la projection de chaque état marqué du graphe de synchronisation sur tous les modules composant ces états.
- enfin si l'état initial de chaque module est étiqueté par la formule φ alors on déduit que la formule ou la propriété est vraie (vérifiée) sinon la formule est fausse.

==========
Procédure de vérification check(φ=φ1∨φ2)
==========
Begin
//pour tous les modules auxquels φ1 ou φ2 appartient
For all $S_i \in SS$ ($i \in \{1..n\}$) do
Foreach $M \in S_i$ do
If($M \in sat(\varphi1 \vee \varphi2)$) then $sat(\varphi) \leftarrow sat(\varphi) \cup \{M\}$, Mark M , Anc=Mark_ancestorSCC(M) by φ

For all $ss_i \in SS$ //($ss_i$ état du graphe de synchronisation)
For all $z \in Anc$
if ($z \in ss_i$)
mark $ss_i$ par φ
For all $ss_i$ marked by φ
mark $\Pi(M_{Si})$ by φ (marquer la projection de l'état du graphe de synch dans les modules)
For all $S_i$ (i=1..n) //pour tous les modules
If ($M_{0i}$ is marked by φ) //si l'état initial de chaque module est marqué par φ//
return true //property is true
Else return false //property is false

End
==========

## Model-checking de φ=EXψ

Le principe de cet algorithme est le suivant :

- voir sur les modules auxquels ψ appartient, et étiqueter les états pères (le prédécesseur direct) de tous les états qui satisfont ψ par φ.
- chercher les ancêtres de ces états.
- marquer (étiqueter) par φ, chaque état contenant un ancêtre dans le graphe de synchronisation.
- ensuite étiqueter la projection de chaque état marqué du graphe de synchronisation sur tous les modules composant ces états.
- enfin si l'état initial de chaque module est étiqueté par la formule φ alors on déduit que la formule ou la propriété est vraie (vérifiée) sinon la formule est fausse.

==========
Procédure de vérification check(φ=EXψ)
==========
Begin
//pour tous les modules auxquels ψ appartient
For all $S_i \in SS$ ($i \in \{1..n\}$) do
Foreach $M \in S_i$ do
If($M \in sat(\psi)$)
//marquer chaque état prédécesseur direct de //M par φ
Foreach transition $t \in (T \backslash TF)_{Si}$ and $M'[t>M$ then $sat(\varphi) \leftarrow sat(\varphi) \cup \{M\}$,Mark M , Anc=Mark_ancestorSCC(M) by φ
For all $ss_i \in SS$ //($ss_i$ état du graphe de synchronisation)
For all $z \in Anc$
if ($z \in ss_i$)
mark $ss_i$ par φ
For all $ss_i$ marked by φ
mark $\Pi(M_{Si})$ by φ (marquer la projection de l'état du graphe de synch dans les modules)
For all $S_i$ (i=1..n) //pour tous les modules
If ($M_{0i}$ is marked by φ) //si l'état initial de chaque module est marqué par φ
return true //property is true
Else return false //property is false
End

==========

## Model-checking de φ=Eφ1Uφ2

Le principe de cet algorithme est le suivant :

- voir sur les modules auxquels φ1 et φ2 appartiennent, s'il y a un chemin dans lequel φ1 est vérifiée jusqu'à ce que φ2 le soit et étiqueter ces états par φ (même si φ1 et φ2 n'appartiennent pas au même module on effectue une vérification de φ1 et φ2 a part et on cherche les chemins dans lesquels φ est vérifiée et on procède a l'étiquetage).
- chercher les ancêtres de ces états.
- marquer (étiqueter) par φ, chaque état contenant un ancêtre dans le graphe de synchronisation.
- ensuite étiqueter la projection de chaque état marqué du graphe de synchronisation sur tous les modules composant ces états.

- enfin si l'état initial de chaque module est étiqueté par la formule φ alors on déduit que la formule ou la propriété est vraie (vérifiée) sinon la formule est fausse.

```
==========================================
Procédure de vérification  check(φ=Eφ1Uφ2 )
==========================================
Begin
//pour tous les modules auxquels φ1 et φ2 appartiennent
        For all Si∈SS  (i∈{1..n}) do
        Foreach M∈Si do
        If(M∈sat(φ))
                //marquer chaque état prédécesseur direct
        de M par φ
                Foreach  transition t∈(T\TF)Si  and
        M'[t>M then sat(φ)← sat(φ)∪{M}, Mark M ,
        Anc=Mark_ancestorSCC(M) by φ

        For all ssi∈SS //(ssi état du graphe de
        synchronisation)
            For all z∈Anc
              if (z ∈ssi)
                 mark ssi par φ
        For all ssi marked by φ
                 mark Π(MSi) by φ  (marquer la
        projection de l'état du graphe de synch dans les
        modules)
        For all Si (i=1..n) //pour tous les modules
          If (M0i is marked by φ) //si l'état initial de chaque
        module est marqué par φ
              return true //property is true
            Else return false //property is false
End

==========================================
```

## CONCLUSION

La vérification et la validation des systèmes par model-checking est souvent confrontée au problème de l'explosion combinatoire liée à la taille du système qui est exponentielle en nombre d'états du système.

Dans ce travail, nous nous sommes intéressés à la vérification des propriétés spécifiques des systèmes exprimées en logique temporelle CTL en utilisant les RdPs modulaires pour la modélisation du système, donc le système est représenté sous forme de modules et chaque module est modélisé par un RdP pour éviter le problème de l'explosion combinatoire.

Nous utilisons une technique basée sur l'idée de vérifier si la propriété est vérifiée par chaque module ainsi que sur le graphe de synchronisation pour décider si elle est satisfaite par le système ou non. Cette technique permet de vérifier des systèmes de grande taille.

Notre contribution dans ce travail est la propositions des algorithmes de vérification des propriétés exprimées en logique CTL sur les RdP modulaires, comme perspective à ce travail, une implémentation de ces algorithme est en cours de finalisation pour prouver qu'ils sont corrects, qu'ils permettent de vérifier des systèmes de grande taille et que le problème de l'explosion combinatoire est réduit.

## REFERENCES


**[1]:** A.Gu, K.G.Shin «Analysis of event-driven real time systems with time Petri nets, A translation approch», IFIP conference proceedings, vol,219.Proceedings of the IFIP 17th world computer congress-Tc10 stream on distributed and parallel Embedded systems :design and analysis of distributed embedded systems, pages 31-40, 2002

**[2]**: S. Christensen and L. Petrucci. Modular analysis of Petri nets. The Computer Journal, 43(3) :224–242, 2000.

**[3]**: K.Klai, L. Petrucci: Modular construction of the symbolic observation graph ACSD2008: 88-97

**[4]:** E.A.Emerson,A.P.Sistla «Symetrie and Model-Checking »,In formal Methods and System Design 9,pp105- 031, 1996

**[5]:** P. Schnoebelen, B. Bérard, M.Bidoit, F.Laroussinie,A. Petit «Systems and software verification-model-checking techniques and tools», Springer, 2001.

**[6]:** J.B. Jorgensen, L.M. Krinsten.«Computer aided verification of Lamportas fast mutual extension algorithme using Colored Petri Nets and occurence graphs with symetries». IEEE transaction on parallel and distributed systems ;1999.

**[7]:** L. Petrucci, C.Lakos.«Distributed and modular state space exploration for timed Petri nets», LIPN, CNRS UMR 7030, Université Paris VIII, 2005.

**[8]**:S. Haddad, F.Vernadat «vérification et mise en oeuvre des réseaux de Petri». Vérification des propriétés spécifiques Paris,Hermes science publication, 2003

**[9]** :M.C. Boukala, L. Petrucci. Distributed verification of modular systems. In In Proc. workshop on Petri Nets Compositions (CompoNet'11, associated with Petri Nets 2011), Volume 726 of CEUR workshop proceedings, pages 1–15, June 2011.

**[10]**: A. Pnueli. The temporal logic of programs. In Proceedings of the 18th IEEE Symposiu on Foundations of Computer Science, pages 46–57, 1977.

**[11]**: C.Lakos, L.Petrucci: Modular Analysis of Systems Composed of Semiautonomous Subsystems. ACSD2004:185-196

**[12]:**L.Petrucci: Experiments with Modular State Spaces. Petri Net Newsletter 68, April 2005.