SORBONNE UNIVERSITÉ

FACULTÉ DES SCIENCES ET INGÉNIERIE

DEPARTEMENT D'INFORMATIQUE

Model-Checking Contest

Cas : *Global Properties*

Réalisé par

Amine BENSLIMANE

Sofiane BRANECI

Yann THIERRY-MIEG

Master 1 STL - 2020 / 2021

24 mai 2021

Table des matières

Table des figures

1 Introduction et mise en contexte

Aujourd'hui, les systèmes informatiques complexes ont envahi le quotidien. En parallèle, la complexité de ces systèmes augmente constamment, cela est dû à l'adaptation de protocoles de communication filaires ou sans fil lors de la réalisation de ces solutions logicielles.

Le challenge qui se pose est de trouver des formalismes, techniques et outils qui permettront de concevoir des systèmes corrects malgré leur complexité. Le génie logiciel offre plusieurs alternatives, on citera le développement dirigé par les tests *(Test Driven Development)*.

Toutefois et bien qu'utilisée dans l'industrie, cette approche comme toutes les approches de tests nécessite la réalisation (implementation) du système et reste inadaptée quant aux systèmes concurrents.

En effet, l'ensemble des entrelacements possibles des processus de ces derniers ne sont généralement pas couverts, et de ce fait, des exécutions non désirées (même si rares) peuvent se produire.

Le Model-Checking est une méthode de vérification formelle qui permet aux propriétés comportementales souhaitées d'un système donné d'être vérifiées sur la base d'un modèle approprié du système par une inspection systématique de tous les états du modèle. L'attractivité de cette technique repose sur le fait qu'elle est entièrement automatique et offre des contre-exemples dans le cas où la vérification échoue.

L'approche naïve des model-checkers serait un parcours exhaustif des états du système étudié, cependant l'explosion combinatoire des états du système que cela provoque est un véritable frein et suscite le développement de stratégies matures pouvant réduire l'espace à vérifier sans toutefois fausser le résultat.

Ce rapport présente notre travail dans le cadre du projet "Soumission dans la catégorie Propriétés Globales au Model-Checking Contest".

Le Model Checking Contest (MCC) est un évènement scientifique annuel dédié aux outils de vérifications formelles pour les systèmes concurrents.

Notre encadrant participe à ce contest avec un outil ITS-tools, mais l'outil ne savait pas au début du projet traiter certaines propriétés globales.

ITSTools est un model-checker principalement développé par notre encadrant, M. Yann Thierry-Mieg au sein du LIP6. Il supporte plusieurs formalismes pour décrire des systèmes concurrents dont les réseaux de Petri et leurs variantes colorées utilisées dans le MCC.
L'outil sait prouver formellement l'intégrité des systèmes exprimée comme des invariants ou

des propriétés exprimées en logique temporelle.

Dans le cadre de ce projet, nous avons donc réalisé l'implémentation de la vérification formelle de cinq propriétés globales dites *Global Properties* en vue d'une soumission au Model Checking Contest qui se tiendra en Juin 2021 à Paris.

Pour cela, nous avons implanté une approche combinée :

— Une version relativement simple qui s'appuie sur une transformation des propriétés vers un format supporté par l'outil actuel

— Pour plusieurs des propriétés nous avons introduit des conditions nécessaires et/ou suffisantes pour conclure sans invoquer l'outil actuel, ou pour limiter le nombre d'invocations nécessaires.

Le plan de ce rapport est le suivant, nous commençons en section 2 par définir les objets et notations utiles pour la suite. En section 3 à section 7 nous traitons les propriétés globales du contest une par une. La section 8 propose une évaluation du résultat sur les modèles utilisées dans le concours 2020 et une comparaison (très favorable) aux outils ayant participé en 2020. Nous terminons par une conclusion.

2 État de l'art

Dans cette section, nous présentons les fondements de l'art et les avancées de la science qui permettent de prendre en mains et d'introduire nos travaux.

2.1 Principe du model checking

On appelle modèle de comportement tout système formel représenté par un réseau de Petri, automates, etc.

Etant donné un modèle de comportement $\mathcal{M}$ et un ensemble de propriétés ϕ.
Un Model-Checker est l'algorithme de vérification permettant de décider si le système $\mathcal{M}$ satisfait ϕ à travers une énumération exhaustive (explicite ou implicite) de tout les états accessibles par le système et les comportements qui les traversent. Le cas échéant, il produit une trace d'un contre exemple pour au moins une propriété de ϕ.

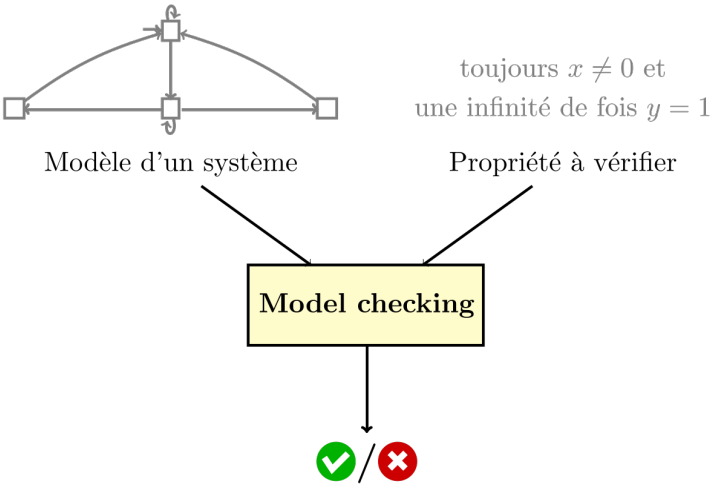

FIGURE 1 – Principe du Model-Checking

2.2 Classe des propriétés des systèmes réactifs

Il existe plusieurs catégories de propriétés, certaines sont exprimées de façon spécifique à un système, par exemple "il existe un état accessible où deux processus sont en section critique". Ce type de propriété s'exprime sous la forme d'invariants ou de propriétés d'accessibilité. Dans ce travail, on s'appuie sur un outil qui sait traiter ce type de propriétés, exprimées comme des invariants ou de l'accessibilité. Les propriétés dites "globales" au contraire sont exprimées indépendamment du système, par exemple, il n'existe pas d'état de blocage (deadlock) ou encore le système est vivace (le système admet toujours un avancement). Notre objectif dans ce travail est de fournir une procédure de décision permettant de conclure pour ce type de questions globales.

utile la suite de cette sous sect. ?

Les propriétés vérifiables par un model-checker sont partitionnées en des classes que voici :

— Accessibilité : Un(e) certain(e) état (situation) peut être atteint(e)
— Invariant : Propriété vraie en chaque état

— Sûreté : Contrôle du comportement du système
— Vivacité : Le système admet toujours un avancement (réponse)
— Equité : Aucune hypothèse sur les privilèges des actions

2.3 Modélisation des systèmes de comportement

Les systèmes de transitions sont souvent utilisés dans la littérature comme modèles pour décrire le comportement des systèmes. Ce sont des graphes orientés où les noeuds sont les états et les arcs sont les transitons du système. Les transitions spécifient l'évolution du système d'un état à l'autre.

La description de ces systèmes peut se faire par divers formalismes comme les réseaux de Petri ou encore les automates.
Dans le Model-Checking Contest [] la modélisation des systèmes utilise les réseaux de Petri, que nous définissons ci-dessous.

2.3.1 Réseau de Petri

Les définitions dans cette section d'un réseau de Petri sont adaptées de [1].

Definition 2.1 (Réseau de Petri)**.** Un réseau de Petri $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}_-, \mathcal{W}_+, m_0 \rangle$ est un tuple où $\mathcal{P}$ est l'ensemble des places du réseau, $\mathcal{T}$ l'ensemble des transitons, $\mathcal{W}_- : \mathcal{P} \times \mathcal{T} \mapsto \mathbb{N}$ et $\mathcal{W}_+ : \mathcal{P} \times \mathcal{T} \mapsto \mathbb{N}$ représentent respectivement la matrice pré et post incidence, $m_0 : \mathcal{P} \mapsto \mathbb{N}$ est le marquage initial.

Les réseaux de Petri peuvent être représentés graphiquement comme sur la Fig 2 [1]où les cercles représentent des places et les transitions sont des rectangles.

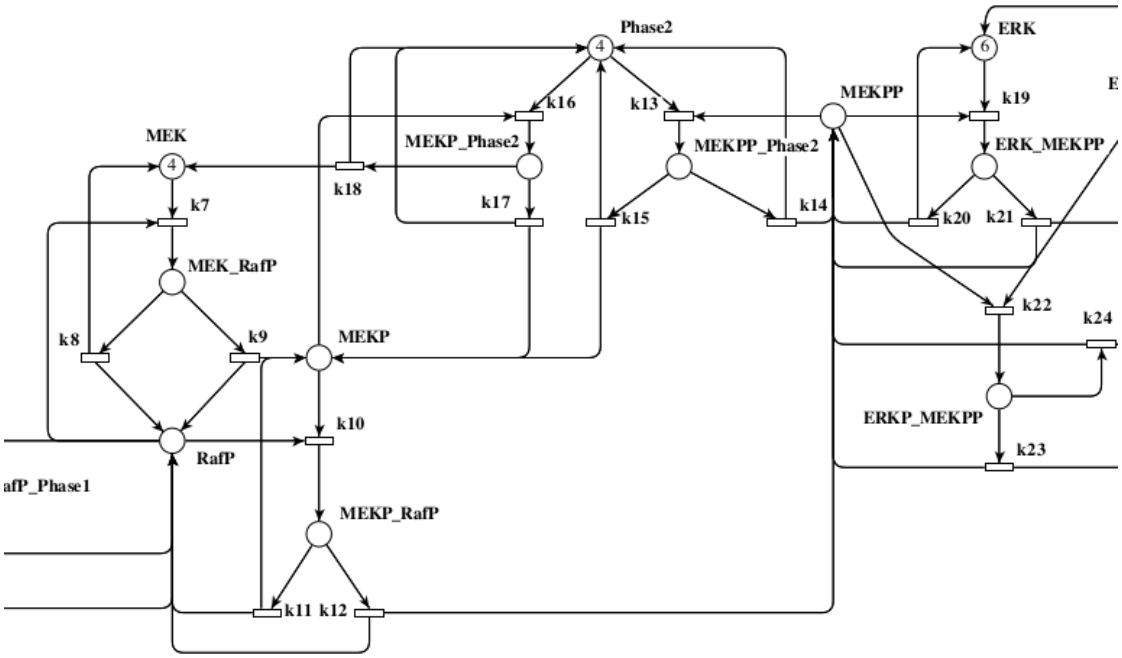

FIGURE 2 – Extrait du modèle MAPK représentant une réaction biochimique

1. Prise dans la description officielle des modèles du MCC ici : https://mcc.lip6.fr/pdf/MAPK-form.pdf

On note p (resp t) une place (resp. transition). Un marquage m est composé de $|\mathcal{P}|$ entrées. $\mathcal{W}_-^\top, \mathcal{W}_+^\top$ sont les matrices de flot transposée, où $\mathcal{W}_-^\top(p)$ est un vecteur de $|\mathcal{T}|$ entrées. On note $\mathcal{W}_e = \mathcal{W}_+ - \mathcal{W}_-$ la matrice d'effet.

On précise que l'on utilise $v \geq v'$ pour dénoter $\forall i, v(i) \geq v'(i)$. On note $\bullet n$ (resp. $n\bullet$) l'ensemble des prédecesseurs (resp. successeurs) d'un noeud n (place ou transition). E.g. pour une transition t, $\bullet t = \{p \in \mathcal{P} \mid \mathcal{W}_-(p,t) > 0\}$. On dit d'une transition t qu'elle est *activable* par un marquage m si et seulement si $m \geq \mathcal{W}_-(t)$. Une transition t est dite *lisible* d'une place p si $\mathcal{W}_-(p,t) > 0 \wedge \mathcal{W}_e(p,t) = 0$.

Sémantique de réseaux de Petri.

Un réseau de Petri est une spécification d'un système concurrent, dont l'espace d'états accessibles correspond à une structure de Kripke.

Definition 2.2 (Sémantique)**.** La sémantique d'un réseau de Petri est donnée par la règle $\xrightarrow{t}$ qui relie des pairs de marquage : $\forall\, m \in \mathbb{N}^{|\mathcal{P}|}$, si $t \in \mathcal{T}$ satisfait $m \geq \mathcal{W}_-(t)$, alors $m \xrightarrow{t} m'$ avec $m' = m + \mathcal{W}_+(t) - \mathcal{W}_-(t)$.
L'ensemble accessible *Reachable* $\mathcal{R}$ est défini inductivement par le plus petit sous-ensemble de $\mathbb{N}^{|\mathcal{P}|}$ satisfaisant $m_0 \in \mathcal{R}$, et $\forall t \in \mathcal{T}, \forall m \in \mathcal{R}, m \xrightarrow{t} m' \Rightarrow m' \in \mathcal{R}$.

2.3.2 Réseau de Petri coloré

Nous introduisons succinctement, dans cette section, un réseau de Petri coloré. Ces réseaux de haut-niveau permettent des définitions paramétriques de systèmes dans une syntaxe plus riche. Le MCC [2] contient un certain nombre de modèles colorés.

La figure3 [2] représente un réseau de Petri correspondant au problème classique des philosophes.

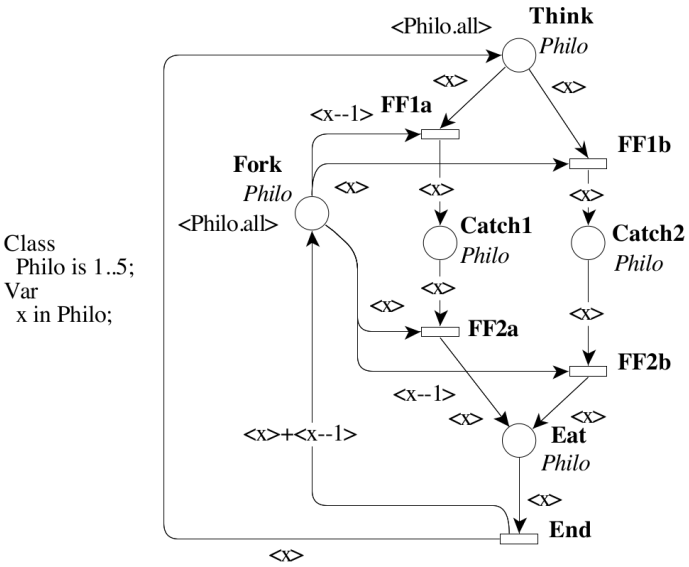

FIGURE 3 – Illustration d'un réseau de Petri coloré

2. Prise dans la description officielle des modèles du MCC ici : https://mcc.lip6.fr/pdf/Philosophers-form.pdf

Pour traiter ces modèles colorés, l'outil ITS-tools qui nous a été fourni propose deux approches correspondant à construire un réseau de Petri classique à partir du modèle coloré :

— Le "dépliage" du réseau coloré engendre un réseau de Petri dont la sémantique reflète de façon exacte celle du réseau coloré. Cependant, pour certains réseaux colorés, ce processus peut engendrer une explosion du nombre de places, transitions et arcs du système.
— Le "squelette" du réseau coloré engendre un réseau de Petri dont le comportement est un sur-ensemble strict du du comportement du coloré. En d'autres termes on a une sur-approximation des comportements, si le squelette peut le faire le réseau d'origine aussi, mais l'inverse est faux. La construction du squelette est beaucoup moins explosive que le dépliage complet.

En conséquence, dans la suite de ce rapport, on utilisera plutôt l'un de ces deux objets pour raisonner sur les modèles colorés.

3 Global Properties

Dans le cadre de nos travaux, nous avons été chargés d'implémenter les stratégies de vérification des propriétés globales.

Nous introduisons donc dans cette section, l'ensemble de ces propriétés utilisées dans le MCC [2]. Les cinq propriétés du MCC dans la catégorie "GlobalProperties" sont :

1. OneSafe : aucune place du réseau ne contiendra jamais plus d'un jeton
2. StableMarking : au moins une place du réseau est stable, son marquage ne sera jamais modifié
3. QuasiLiveness : il existe une façon de franchir chacune des transitions du système
4. Liveness : dans tout état accessible, il existe une façon de franchir chacune des transitions du système
5. ReachabilityDeadlock : il existe un état ou aucune transition n'est franchissable. Cette dernière propriété était déjà disponible dans ITSTools au début du projet, et n'est donc pas abordée en détail dans la suite.

Chacune des sections qui suit va présenter la façon de traiter la propriété, d'abord en s'appuyant sur une transformation vers une ou plusieurs propriétés déjà supportées par l'outil ITSTools, puis en introduisant des stratégies permettant soit de conclure, soit de limiter le travail du model-checker sous-jacent.

4 Propriété OneSafe

On dit qu'un modèle est OneSafe si dans tous ses marquages accessibles, il y a au plus un jeton par place, c-à-d :

Definition 4.1 (OneSafe Property)**.** Un réseau de Petri N est OneSafe si et seulement si :

$$\forall p \in \mathcal{P}, \forall m \in \mathcal{R}, m(p) \leq 1$$

Dans cette rubrique, nous présentons l'ensemble des stratégies utilisées pour répondre à l'examination OneSafe.

4.1 Stratégie par transformation

La stratégie de base consiste donc à exprimer un invariant par place p, correspondant à la définition de la propriété. Si au moins un de ces invariants est faux, le système n'est pas OneSafe. Si tous les invariants sont prouvés, le système est OneSafe.

La vérification des invariants peut être déléguée à ITSTools. Nous avons pour cela construit une traduction vers un ensemble d'invariants exprimés dans un format compatible avec ITSTools. Puis nous avons implanté une stratégie qui s'interrompt dès qu'un invariant est violé, via un mécanisme basé sur des exceptions.

Cette stratégie est une stratégie de décision complète, à condition qu'ITS-Tools parvienne à prouver tous les invariants.

4.2 Stratégie basée sur des informations structurelles

Certains réseaux du MCC portent dans leur définition des informations sur le fait que le réseau sous-jacent est one-safe, car par construction l'auteur du modèle le garantit. Il suffit donc si cette étiquette est présente de répondre vrai. Sinon on ne peut pas encore conclure c'est donc une condition suffisante, mais pas nécessaire.

4.3 Stratégie spécifiques aux réseaux colorés

Pour les réseaux colorés, on peut contrôler si le squelette est OneSafe, si c'est le cas, comme le squelette est une sur-approximation, il est garanti que le réseau aussi le sera. Ceci évite le dépliage dans beaucoup de cas, surtout que pour beaucoup de modèles colorés, la réponse est que le réseau n'est *pas* OneSafe, et cela peut être déterminé en regardant l'état initial du système.

Si le squelette n'est pas OneSafe, on ne peut pas encore conclure, c'est donc une condition suffisante mais pas nécessaire.

4.4 Algorithme global pour OneSafe

L'algorithme complet pour OneSafe consiste donc :

1. Si le réseau porte une information structurelle conclure vrai
2. Si le réseau est coloré, tester si le squelette est OneSafe, si oui conclure vrai
3. Si les autres tests ont échoué, traduire vers une série d'invariants, les transmettre à ITS-Tools, s'interrompre à la volée dès qu'un des invariants est violé et conclure faux. Si tous les invariants sont prouvés par ITSTools dans le temps imparti, conclure vrai.

Ainsi, nous présentons l'algorithme formel :

Algorithm 1: OneSafe Algorithm

Entrée: Réseau N
Sortie: N OneSafe TRUE / FALSE
  pour toute place $p \in \mathcal{P}$ **faire**
    Construire l'invariant $m(p) \leq 1$
    si invariant **alors**
      la place p est OneSafe
    sinon
      renvoyer N OneSafe FALSE
    fin si
  fin pour
  renvoyer N OneSafe TRUE

5 Propriété StableMarking

On dit qu'un modèle est à marquage stable (StableMarking noté SM) si le marquage d'au moins une place reste stable dans tous les états accessibles.

Definition 5.1 (Stable Marking)**.** Un réseau de Petri N est Stable si et seulement si :

$$\exists p \in \mathcal{P}, \forall m \in \mathcal{R}, m(p) = m_0(p)$$

5.1 Stratégie par transformation

La stratégie de base consiste donc à exprimer un invariant par place p, correspondant à la définition de la propriété. Si au tous ces invariants sont faux, le système n'est pas SM. Si au moins un des invariants est vérifié le système est SM.

Similairement à la stratégie pour OneSafe, la vérification des invariants peut être déléguée à ITSTools et interrompue à la volée.

Cette stratégie est une stratégie de décision complète, à condition qu'ITSTools parvienne à prouver qu'aucun des invariants n'est vérifié.

Lemme 5.1. Il suffit de montrer que le marquage d'une place est stable pour conclure sur la propriété, et on écrit :

$$\exists p \in \mathcal{P}, \mathcal{W}_e^\top(p) = 0 \Rightarrow \textit{StableMarking True}$$

Démonstration. Aucune transition n'ayant d'effet sur le marquage de la place p, il ne peut donc pas varier. Si une place est invariante le réseau vérifie StableMarking. ■

5.2 Stratégie basée sur des informations structurelles

L'outil ITSTools est capable de détecter les places structurellement constantes dans le réseau coloré selon un algorithme qui revient à calculer si $\mathcal{W}_e^\top(p) = 0$ en coloré. Dans notre cas d'étude, il suffit donc vérifier pour une place $p \in \mathcal{P}$, son attribut *isConstant* qui permet de contraindre ou non une place à être constante (marquage constant) : si c'est bel et bien une contrainte (booléen à vrai), le résultat est immédiat. Sinon, nous ne pouvons pas conclure.

5.3 Stratégie spécifiques aux réseaux colorés

Pour les réseaux colorés, on peut contrôler si le squelette $s(N)$ est stable, si c'est le cas, comme le squelette est une sur-approximation, il est garanti que le réseau aussi le sera. Ceci évite le dépliage dans beaucoup de cas, surtout que pour beaucoup de modèles colorés, la réponse est que le réseau n'est *pas* stable, et cela peut être déterminé en regardant l'état initial du système.

Si le squelette n'est pas stable, on ne peut pas encore conclure, c'est donc une condition suffisante, mais pas nécessaire.

Lemme 5.2.

$$s(N) \text{ StableMarked} \Rightarrow \text{N StableMarked}$$

Démonstration. Trivial par définition. ■

5.4 Réduction structurelle : composante fortement connexes *SCCTest*

Nous définissons ci-dessous la notion de composante fortement connexe d'un graphe. Soit $G = (\mathcal{P}, \mathcal{T})$ le graphe orienté d'un réseau de Petri.

Definition 5.2 (Composante fortement connexe)**.** Une composante fortement connexe (SCC) ψ de G est un sous-ensemble maximal de sommets de G tel que :

$$u, v \in \psi \Leftrightarrow u \xrightarrow{*}_G v \wedge v \xrightarrow{*}_G u$$

Principe : Le but est de réduire le nombre d'invariants à invalider par ITSTools. La stratégie par défaut prévoit en effet de vérifier $|\mathcal{P}|$ invariants. Pour cela on peut éliminer des places que l'on prouve instables en examinant simplement la structure du réseau.

Pour cela on peut étudier un graphe G (introduit dans [1]) qui contient un noeud par place du réseau, et un arc de p vers p' si et seulement si il existe une transition qui consomme un jeton dans p et le place dans p'.

Le graphe contient un noeud pour chaque place $p \in \mathcal{P}$ et un arc de p vers p' ssi.

$$\exists t \in \mathcal{T}, \bullet t = \{p\} \wedge \mathcal{W}_-(p,t) = 1, t\bullet = \{p'\} \wedge \mathcal{W}_+(p',t) = 1$$

Dans une SCC, si le marquage d'au moins une place est positif, toutes les places de la SCC sont instables, car les jetons circulent librement entre ces places. Nous retirons donc toutes les places appartenant à une SCC dont au moins une place est initialement marquées du graphe : elles sont instables strcturellement.

On procède au calcul de l'ensemble ψ des composantes fortement connexes (SCC) du graphe engendré à partir du modèle. Pour ce faire, nous avons implémenté l'algorithme de recherche de composantes fortement connexes Tarjan.

Lemme 5.3. Soit ψ l'ensemble des SCC de G. $\forall s \in \psi, \exists p \in s$ tel que $|s| > 1, m(p) > 0 \Rightarrow \forall p \in s$, p instable $\wedge\ G = G \backslash \{s\}$

Démonstration. Soient deux places $u, v \in \psi$,
On a : $u \xrightarrow{*}_G v \ \wedge \ v \xrightarrow{*}_G u$, par définition de G, un chemin d'une place u vers v veut dire l'activation de toutes les transitions de l'état u vers v dans N, et ainsi l'instabilité de toutes les places de u vers v. ■

Nous présentons ci-dessous l'algorithme pour SCCTest :

Algorithm 2: SCCTest Algorithm

Entrée: Réseau N
Sortie: Graphe du réseau sans places prouvées instables

```
SCC du graphe N
- Construire le graphe G (N × N matrice)
- S = TarjanAlgorithm(G)
pour toute SCC s ∈ S faire
  pour toute place p ∈ s faire
    si ∃p ∈ s, m(p) > 0 alors
      ∀p ∈ s, p instable
      G = G\{s} // enlever p de G, ∀p ∈ s
    fin si
  fin pour
fin pour
```

5.5 Algorithme général pour StableMarking

Ainsi, en tenant compte des conditions suffisantes et des réductions structurelles précédemment énoncées, on propose l'algorithme général suivant traitant l'examination StableMarking :

1. Si le réseau porte une information structurelle, conclure vrai
2. Si le réseau est coloré, tester si le squelette est StableMarked, si oui conclure vrai
3. Les tests ont échoués, on réduit le nombre de places du graphe par *SCCTest* et on le transmet à ITS-Tools, on s'interrompt à la volée dès qu'un invariant est violé, conclure faux. Si tout les invariants sont prouvés par ITS-Tools dans le temps imparti, conclure vrai.

6 Propriété QuasiLiveness

On entend par un modèle quasi-vivant *QuasiLive*, un modèle dont toutes les transitons sont activables dans au moins un marquage accessible.

Definition 6.1 (QuasiLiveness)**.** Un réseau de Petri N est quasi-live si et seulement si :

$$\forall t \in \mathcal{T}, \exists m \in \mathcal{R}, m \geq \mathcal{W}_-(t)$$

* : On note $|s|$ la cardinalité de la composante fortement connexe s, c-à-d le nombre de places qu'elle contient.

6.1 Stratégie par Transformation

La stratégie de base consiste donc à exprimer un invariant $\forall m \in \mathcal{R}, m < \mathcal{W}_-(t)$ par transition t, correspondant à la définition de la propriété. Si pour au moins une transition, c'est un invariant pour tout marquage accessible alors N est QuasiLive. Sinon, on itère sur les transitions. Si au bout de toutes les transitons, l'invariant n'est pas tenu, on en conclue que N n'est pas QuasiLive.

Soient $\mathcal{T}$ l'ensemble des transitions et $\mathcal{R}$ l'ensemble des marquages accessibles du réseau.

Lemme 6.1.

$$\exists t \in \mathcal{T}, \forall m \in \mathcal{R}, m < \mathcal{W}_-(t) \Rightarrow \text{Not QL}$$

Démonstration. Par définition. ■

6.2 Réduction structurelle : transitions dominées

Soient $t, t' \in \mathcal{T}$

Definition 6.2 (Transition dominée)**.** Une transition t est dite dominée par une transition t' si et seulement si elles ont le même effet mais t' a plus de pré-conditions. Et on écrit : $\exists t, t' \in \mathcal{T}, \mathcal{W}_e(t) = \mathcal{W}_e(t') \wedge \mathcal{W}_-(t) \geq \mathcal{W}_-(t')$, t est dominée par t'. [1]

Principe : On enlève toutes les transitions dominées pour simplifier le réseau et accélérer le temps de réponse des stratégies de l'outil.

6.3 Algorithme global pour QuasiLiveness

Ainsi, on résume les étapes dans l'algorithme global suivant :

1. On commence par enlever du modèle toutes les transitions dominées
2. On transmet les invariants à vérifier à ITS-Tools, on s'interrompt à la volée dès qu'un invariant est violé, conclure faux. si tous les invariants sont prouvés par ITS-Tools dans le temps imparti, conclure vrai.

7 Propriété Liveness

On entend par un modèle vivant *Live*, un modèle dont toutes les transitons sont activables dans tout marquage accessible.

Definition 7.1 (Liveness)**.** Un réseau de Petri N est vivant si et seulement si :

$$\forall t \in \mathcal{T}, \forall m \in \mathcal{R}, m \geq \mathcal{W}_-(t)$$

7.1 Stratégie par Transformation

La stratégie de base consiste donc à exprimer un invariant $\exists m \in \mathcal{R}, m < \mathcal{W}_-(t)$ par transition t, correspondant à la définition de la propriété. Si pour toutes les transitions, c'est un invariant pour tout marquage accessible alors N est Live. Sinon, on itère sur les transitions. Si au bout de toutes les transitons, l'invariant n'est pas tenu, on en conclue que N n'est pas Live.

Soient $\mathcal{T}$ l'ensemble des transitions et $\mathcal{R}$ l'ensemble des marquages accessibles du réseau.

Lemme 7.1.

$$\exists t \in \mathcal{T}, \exists m \in \mathcal{R}, m < \mathcal{W}_-(t) \Rightarrow \text{Not L}$$

Démonstration. Par définition. ■

7.2 Stratégie spécifiques aux réseaux colorés

Comme pour l'examination StableMarking, une condition suffisante pour que le réseau soit stable est la vivacité du squelette. Si le squelette est vivace alors on conclue que le réseau l'est aussi. Si le squelette n'est pas vivace, on ne peut pas encore conclure, c'est donc une condition suffisante, mais pas nécessaire.

Lemme 7.2.

$$s(N) \text{ Live} \Rightarrow \text{N Live}$$

Démonstration. Trivial par définition. ■

7.3 Réduction structurelle : SCCTest

La réduction structurelle définie en 5.4 pour l'examination StableMarking reste applicable pour l'examination Liveness quant à la franchissabilité des transitions. En effet, toutes les transitions des places appartenant à une SCC sont franchissables seulement si un marquage d'une place est positif.

Lemme 7.3. Soit ψ l'ensemble des SCC de $G = (\mathcal{P}, \mathcal{T})$. $\forall s \in \psi$, $\exists p \in s$ tel que $|s| > 1, m(p) > 0 \Rightarrow \forall t \in s$, t franchissable $\wedge\ G = G \backslash \{s\}$

Démonstration. Soient deux places $u, v \in \psi$,
On a : $u \xrightarrow{*}_G v \ \wedge\ v \xrightarrow{*}_G u$, par définition de G, un chemin d'une place u vers v veut dire l'activation de toutes les transitions de l'état u vers v dans N, immédiat. ■

7.4 Réduction structurelle : transitions dominées

On enlève toute transition dominée du graphe.

Lemme 7.4. Soient $t, t' \in \mathcal{T}$, t est dominée par $t' \Rightarrow \mathcal{T} = \mathcal{T} \backslash \{t\}$.

Démonstration. Par définition, tout les états qui activent t activent aussi t' et le résultat est le même. ■

Nous présentons donc l'algorithme ComputeDominatedTransition suivant :

Algorithm 3: ComputeDominatedTransition Algorithm

Entrée: Réseau N
Sortie: N sans transitions dominées

```
pour toute transition t ∈ T faire
    pour toute transition t' ∈ T faire
        si t dominée par t' alors
            N = N \ {t}
        fin si
    fin pour
fin pour
renvoyer N
```

7.5 Réduction structurelle : SiphonTest

Un siphon est un ensemble de place qui ne possède pas de transition sortante, l'existence d'un siphon implique que le réseau n'est pas Live

Lemme 7.5. N Siphon $\Rightarrow$ N NotLive

Démonstration. Soient $p_1, p_2, .., p_n$ les places qui composent un siphon. Par définition de $G = (\mathcal{P}, \mathcal{T})$, à chaque arc (transition) $t_j \in \mathcal{T}$ que l'on franchit pour passer de p_{j-1} à p_j avec $1 \leq j \leq n$, on consomme des jetons dans p_j, et donc on finit par ne plus avoir de jetons au bout d'un certain nombre de franchissements $\Rightarrow$ NotLive. ■

7.6 Conditon suffisante : Not QuasiLiveness

Une condition suffisante pour montrer que le réseau n'est pas Live est de montrer que le réseau n'est pas QuasiLive.

Lemme 7.6. N NotQuasiLive $\Rightarrow$ N NotLive

Démonstration. Par définition. ■

7.7 Condtion suffisante : DeadLockTest

Une condition suffisante pour montrer que le réseau n'est pas Live est de montrer l'existence d'un interblocage *(Deadlock)*

Lemme 7.7. N DeadLock $\Rightarrow$N$NotLive$

Démonstration. Par définition. ■

7.8 Algorithme global pour Liveness

Ainsi, on résume les étapes dans l'algorithme global suivant :

1. Si le réseau est coloré, alors tester sur le squelette. Dans le cas où le squelette est vivace, alors retourner vrai.
2. Déplier le réseau et réduire le nombre de transitions par SCC Test. Si on trouve de siphons avec SiphonTest, alors retourner Liveness Faux. Sinon sur le même réseau déplié lancer le test sur QuasiLiveness, si le réseau n'est pas Quasi-Live alors il ne sera pas Live. Sinon, si le test QuasiLiveness ne permet pas de conclure alors, exécuter le test sur les Deadlocks, l'existence de ce dernier confirme que le réseau n'est pas live.
3. Si tous ces tests échouent, alors on transmet les invariants à vérifier à ITS-Tools, on s'interrompt à la volée dès qu'un invariant est violé, conclure faux. si tous les invariants sont prouvés par ITS-Tools dans le temps imparti, conclure vrai.

8 Résultats obtenus

Dans cette section, nous présentons l'ensemble des résultats obtenus de nos travaux.
Tous les résultats que l'on énonce ont été obtenus suite à des exécutions sur le cluster du laboratoire informatique de Paris 6 *LIP6* par notre encadrant.

Les résultats ont été obtenus avec des contraintes de temps de 900 secondes *TimeOut* sur le Benchmark officiel 2020 du MCC. Ce benchmark contient 1229 modèles différents dont 213 colorés. Pour chacune des cinq propriétés, nous rapportons ici les performances de l'outil ITSTools soumis au MCC 2021.

8.1 Best Virtual Tool (BVT)

Une mesure pertinente de performance consiste à se comparer à l'ensemble des outils de la concurrence. Nous définissons le Best Virtual Tool comme suit : Soit e une examination.

$$(BVT)(e) = \begin{cases} 1 & \text{si au moins un outil répond} \\ 0 & \text{sinon (aucun outil ne répond)} \end{cases}$$

Il peut être vu comme le max de tous les outils pour chaque examination.

La figure 4 montre l'efficience du travail obtenue. En effet, ITS-tools est plus performant sur toutes les examinations que le BVT 2020.

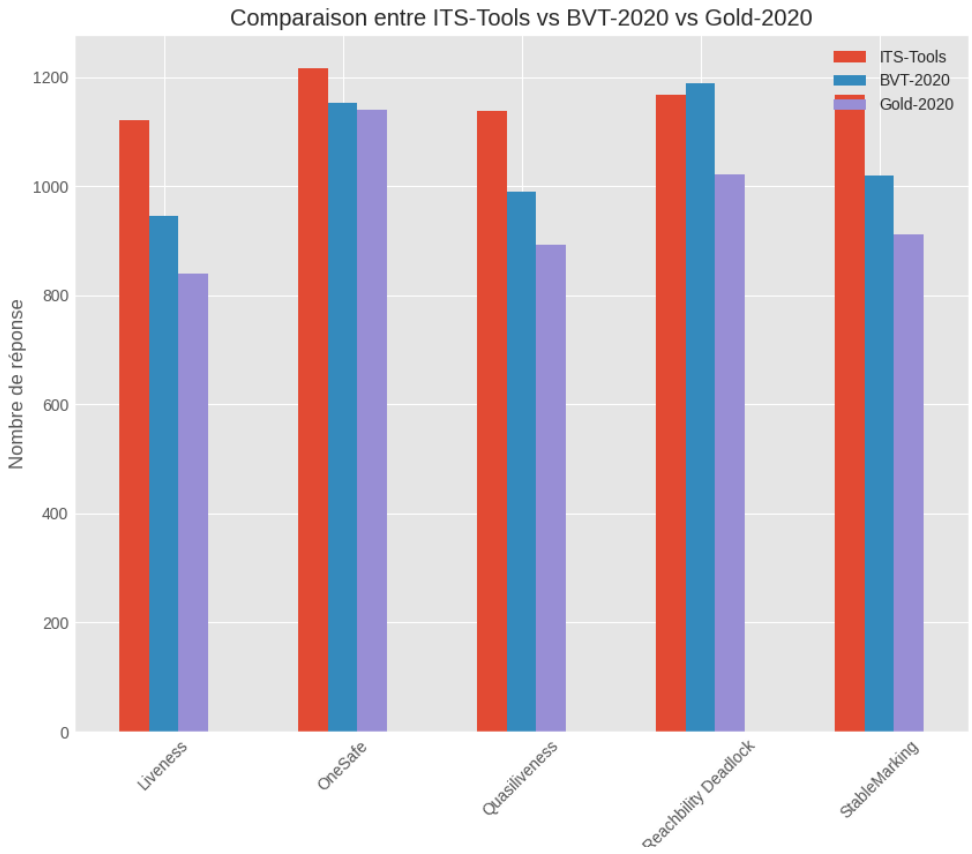

FIGURE 4 – Résultat global pour l'examination GlobalProperties

Nous répondons à plus de 94.4 % du BenchMark 2020 officiel du MCC contre 74.2 % pour Taapal, le vainqueur du MCC de la même édition (2020).
Cette version d'ITS-Tools aurait donc décroché le premier prix de l'an dernier, et y aspire cette année. (dans l'attente des résultats en Juin)

Remarque : L'éxamination DeadlockReachability avait déjà d'excellents résultats et donc n'a pas été traitée dans le cadre de ce projet.

La figure 8.1 ci-dessous représente le résultat de certains modèles pour toutes propriétés globales

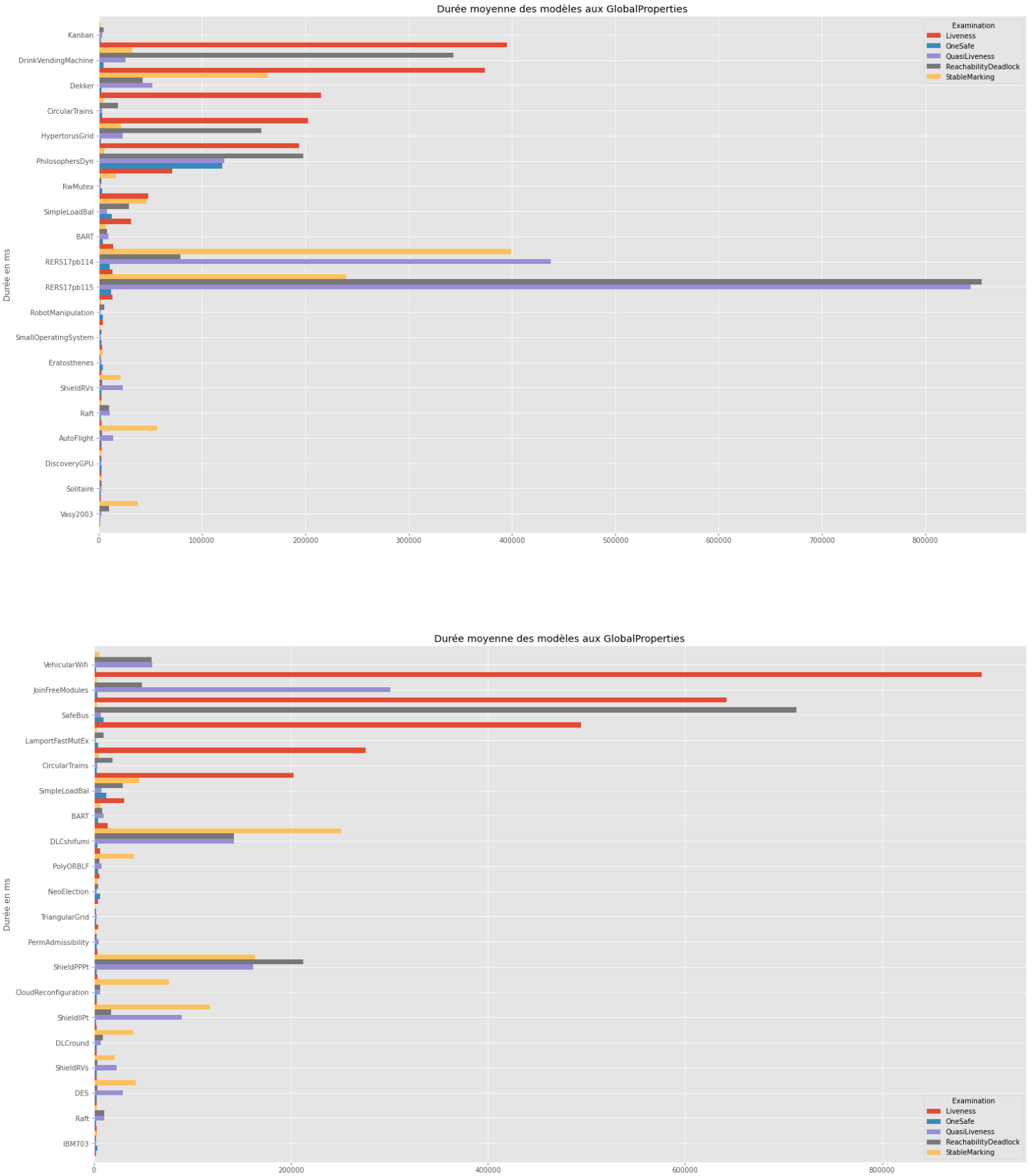

FIGURE 5 – Résultat de certains modèles sur chaque examination globale

9 Conclusion générale et perspectives

À l'issue de ce travail, nous avons réussi à implémenter un ensemble conséquent de stratégies et de techniques pouvant considérablement répondre aux examinations *GlobalProperties* et résultats à l'appui, l'outil ITS-Tools est maintenant capable de défier toutes concurrences à ce propos.

Bien sûr, l'ensemble des modestes contributions que l'on amène ne seraient pas si affûtées sans les outils déjà présents dans ITS-Tools, à savoir l'outil d'accessibilité performant de notre encadrant.

La difficulté primaire que l'on a rencontré est l'abstraction et le *BackGround* rigoureux et théorique que l'on a su par la suite surmonter et prendre en mains pour ainsi y découvrir tout un jargon riche et solide qui nous a permis de produire une implémentation correcte et efficace. Tout cela a été rendu possible par l'aide conséquente de notre encadrant.

Le système a été réalisé en JAVA et déployé sur notre répertoire GitHub suivant : ITS-Tools. L'historique des *commits* permettra une traçabilité de notre travail.

Nous avons préféré une approche théorique sur ce rapport, néanmoins tous les éléments décrits sur ce dernier restent fidèles à leur implémentation.

Finalement, nous avons mergés et soumis au Model Checking Contest notre travail, dans l'attente des résultats (Juin 2021). Toutefois et suite aux résultats obtenus, nous nous attendons à une performance honorable, c'est-à-dire en être les vainqueurs.

Durant tout le long du projet, on s'est documenté sur ITS-tools afin d'assimiler son fonctionnement de base, ce qui nous a permis de mieux anticiper les tâches à venir, ensuite on a abordé les "Global Properties" sous l'axe théorique afin d'appréhender le sujet avant de procéder à la réalisation du système sous l'encadrement et la bienveillance de M. Yann Thierry-Mieg que l'on souhaite remercier pour tous ses efforts, sa disponibilité et par-dessus tout, d'avoir cru en nous.

Le sujet étant vaste et riche, les idées que l'on a eues au fil du projet restent encore ouvertes à réalisation. Le délai de soumission étant un frein, on n'a pas pu tout réaliser.

Ce travail nous a permis d'accentuer nos connaissances dans à la fois le domaine de la réalisation de plug-in, les systèmes de vérifications automatiques en particulier le model-checking, les réseaux de Petri, la théorie des graphes et à tout jargon connexe à notre étude.

Enfin, ce travail nous a rappeler l'art du travail en groupe en divisant les taches et en coopérant pour un but commun, ce qui incontournable dans notre domaine.

Références

[1] Yann THIERRY-MIEG. « Structural Reductions Revisited ». In : *Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Paris, France, June 24-25, 2020, Proceedings*. Sous la dir. de Ryszard JANICKI, Natalia SIDOROVA et Thomas CHATAIN. T. 12152. Lecture Notes in Computer Science. Springer, 2020, p. 303-323. DOI : 10.1007/978-3-030-51831-8_15. URL : https://doi.org/10.1007/978-3-030-51831-8%5C_15 (pages 6, 11, 13).

[2] Elvio Gilberto AMPARORE, Bernard BERTHOMIEU, Gianfranco CIARDO, Silvano DAL-ZILIO, Francesco GALLÀ, Lom-Messan HILLAH, Francis HULIN-HUBARD, Peter Gjøl JENSEN, Loig JEZEQUEL, Fabrice KORDON, Didier Le BOTLAN, Torsten LIEBKE, Jeroen MEIJER, Andrew S. MINER, Emmanuel PAVIOT-ADET, Jiri SRBA, Yann THIERRY-MIEG, Tom van DIJK et Karsten WOLF. « Presentation of the 9th Edition of the Model Checking Contest ». In : *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS : TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*. 2019, p. 50-68. DOI : 10.1007/978-3-030-17502-3_4. URL : https://doi.org/10.1007/978-3-030-17502-3%5C_4 (pages 7, 8).