# Exploration efficace de l'espace d'état basée sur l'IA pour la vérification formelle

**Supervision** Kais Klai - kais.klai@lipn.univ-paris13.fr
Carlos Olarte - olarte@lipn.univ-paris13.fr
Jaime Arias - arias@lipn.univ-paris13.fr
**Lieu** LIPN, CNRS UMR 7030
Université Sorbonne Paris Nord
99 avenue Jean-Baptiste Clément
93430, Villetaneuse, France

## 1 Contexte et Problématique Scientifique

La vérification de la correction d'un système vis-à-vis d'une spécification peut se faire de deux manières : par le test ou par la vérification formelle. La vérification par test est la technique la plus souvent utilisée. Elle consiste à exécuter le logiciel sur des données d'entrée et à évaluer chaque couple entrée/sortie produit par rapport aux spécifications du logiciel, en utilisant un oracle dédié [1]. La vérification par test souffre de plusieurs limites principales : La première est qu'elle n'est pas exhaustive dans le sens où elle opère sur un sous-ensemble d'exécutions possibles (une sous approximation). Le fait que ce sous-ensemble vérifie les propriétés désirées n'implique pas forcément que le système les vérifie. Le deuxième obstacle est que les différents cas nécessaires pour tester un logiciel sont de plus en plus nombreux et difficiles à trouver. On parle de couverture structurelle/fonctionnelle : les cas de test doivent couvrir un maximum de scénarios possibles. La vérification formelle basée sur des fondements théoriques rigoureux et souvent bien outillés se présentent comme un moyen puissant et fiable à même d'apporter des solutions en termes d'analyse de ces systèmes. Parmi les approches de vérification, la vérification de modèles "model checking" [2] revient d'une part à modéliser le comportement du système à vérifier et d'autre part à exprimer la propriété de correction attendue sous forme d'une propriété d'accessibilité, d'un automate ou d'une formule de logique temporelle. Une fois le modèle du système et la propriété désirée sont formellement modélisés, il s'agira ensuite de construire l'espace des états atteignables en énumérant l'ensemble des trajectoires possibles du système afin de vérifier si le modèle satisfait ou non la propriété. Dans le cas de la logique temporelle LTL [9], la vérification se ramène à la recherche de cycles (acceptants) dans le produit synchronisé entre le graphe d'états du système et l'automate (de la négation) de la formule à vérifier. Cela rend cette technique limitée pour des applications sur des systèmes de taille importante (voir infinie), car on est rapidement confronté au problème de l'explosion combinatoire de l'espace des états du système. L'équipe SAFER du LIPN possède une expertise certaine dans les approches de vérification par model checking. Elle compte plusieurs contributions permettant de repousser les limites de ce problème en utilisant des approches de vérification modulaires ([17]), symboliques ([18,5,12,13]), parallèles/distribuées ([11,14] ou par réduction d'ordre partiel ([6]).

Dans les procédures de vérification fondées sur l'analyse d'atteignabilité, la stratégie d'exploration a un impact majeur sur le passage à l'échelle. En effet, l'ordre dans lequel les états sont explorés peut fortement influencer le nombre d'états visités, la consommation mémoire ainsi que le temps nécessaire pour détecter un contre-exemple. Une mauvaise stratégie peut aggraver ainsi le problème de l'explosion combinatoire des espaces d'états. Les progrès récents en apprentissage automatique ont montré que des techniques fondées sur l'apprentissage peuvent découvrir automatiquement des heuristiques non triviales pour des procédures de recherche complexes. En particulier, plusieurs travaux ont étudié l'utilisation de l'apprentissage par renforcement et du guidage neuronal afin d'améliorer la prise de décision dans des tâches de raisonnement combinatoire telles que la résolution SAT, SMT ou la démonstration automatique de théorèmes. Ces approches modélisent la procédure de recherche comme un processus de décision séquentiel dans lequel un agent sélectionne de manière répétée la prochaine branche, clause ou état à explorer en fonction du contexte courant de la recherche. Cette perspective s'applique naturellement aux algorithmes d'atteignabilité et de model checking, où la sélection répétée du prochain état à explorer détermine en grande partie l'efficacité globale de la procédure de vérification. L'objectif est alors d'apprendre des heuristiques d'exploration permettant de prioriser les états les plus susceptibles de conduire à un contre-exemple, à un point fixe, ou à un objectif de preuve, réduisant ainsi les ressources computationnelles nécessaires à la résolution du problème de vérification.

L'utilisation de stratégies de recherche guidées par apprentissage commence également à émerger dans le contexte de la vérification formelle. Par exemple, [8] exploite l'apprentissage par renforcement afin de guider les décisions de branchement dans la résolution de formules booléennes quantifiées. Cette approche repose sur l'idée d'apprendre automatiquement des heuristiques de recherche adaptées à l'exploration de grands espaces d'états. [7] propose des modèles probabilistes permettant de déterminer des abstractions optimales pour la vérification de programmes. Par ailleurs, [3]

s'intéresse au passage à l'échelle des techniques de synthèse de politiques de contrôle à partir de spécifications en LTL. Les auteurs proposent de décomposer le problème de synthèse en sous-problèmes résolus à l'aide de techniques d'apprentissage par renforcement, dans le but de guider plus efficacement l'exploration.

## 2 Objectifs

L'objectif de cette thèse est d'améliorer les techniques de vérification de l'équipe SAFER, basées sur l'exploration de graphes, à l'aide de l'IA – apprentissage supervisé, non supervisé, ou apprentissage par renforcement. Nous visons à mettre en place une méthodologie dans laquelle l'exploration du graphe sera guidée par des oracles afin d'atteindre le plus efficacement possible (engendrant moins de ressources) un objectif de vérification (assurer un critère de couverture donné, atteindre un cycle acceptant ou atteindre un état particulier). Cette méthodologie s'appliquera sur les trois problèmes de vérification suivants :

1. *Critères de couverture pour les tests basés sur des modèles.* En se basant sur les récents travaux de l'équipe [15,16] pour la couverture des transitions d'un SUT (il s'agit de trouver le plus petit ensemble des plus courts chemins d'exécution impliquant toutes les transitions d'un système), l'oracle basé sur l'IA visé ici permettra de guider dynamiquement le parcours du graphe du système. Plutôt que d'explorer l'espace d'états de manière exhaustive, l'oracle apprendra à identifier les chemins les plus pertinentes en fonction de l'état courant, le chemin parcouru, et les différents futurs possibles. Cette approche sera intégrée dans l'outil `sogMBT` développé par l'équipe.
2. *Model checking LTL.* Le model checking LTL est principalement basé sur la recherche de cycles acceptants dans le produit synchronisé entre le graphe du système et l'automate de la (négation de la) formule LTL à vérifier. L'existence d'un tel cycle prouve la violation de la formule. Il s'agira dans cette partie de la thèse de développer un oracle basé sur l'apprentissage qui permet d'orienter l'exploration afin de détecter un cycle acceptant s'il existe. Cette approche pourra être intégrée dans l'outil (`PMC-SOG`) pour bénéficier de l'accélération induite par l'approche de vérification parallèle et distribuée proposée dans [14].
3. *Problème d'accessibilité.* Nous avons montré que plusieurs problèmes de synthèse, notamment la synthèse de paramètres temporisés et la synthèse de stratégies pour des agents [12,13], peuvent être formulés comme des problèmes d'atteignabilité en logique de réécriture [10]. De plus, grâce à l'utilisation de stratégies de réécriture, il est possible, dans certains cas, d'identifier des chemins plus courts menant à l'état cible. L'objectif est : de guider l'exploration de l'espace d'états à l'aide de techniques basées sur l'IA ; d'intégrer ces méthodes dans le moteur de réécriture `Maude` [4] ; de comparer les performances de la commande de recherche "standard" avec celles d'un nouveau mécanisme de recherche guidé par un oracle ; et d'intégrer un tel oracle dans le langage de stratégies de Maude.

## 3 Plan de travail prévisionnel

1. **Année 1** : Fondations, état de l'art, premières expérimentations. L'objectif principal est de construire les bases théoriques et logicielles du projet. Les tâches principales prévues sont :
   (a) Revue de littérature approfondie
   (b) Prototype I : Environnement d'apprentissage pour l'exploration
   (c) Premières expérimentations
   (d) Livrables : Article de survey d'état de l'art ; prototype initial AI-guided exploration ; et 1er article sur l'approche.
2. **Année 2** : Extension de l'approche de couverture basée sur l'IA aux problèmes de model checking LTL et d'accessibilité. L'objectif principal et de développer les deux directions majeures du sujet et les tâches prévues sont :
   (a) Contribution A : IA pour la détection de cycles
   (b) Contribution B : IA pour l'accessibilité
   (c) Prototype II et III pour LTL et accessibilité respectivement, et expérimentation poussée.
   (d) Livrables : Article conférence internationale (e.g., CAV, TACAS, VMCAI, Petri nets, WRLA, etc) ; prototypes II et III.
3. **Année 3** : Intégration complète, optimisation et publications finales. L'objectif principal et de finaliser, optimiser et valider scientifiquement l'approche. Les tâches prévues sont :
   (a) Pipeline complet intégrant les trois outils mentionnés ci-dessus tout en intégrant une interface générique permettant l'utilisation de l'oracle IA par d'autres outils externes (e.g., `IMITATOR`, `Helena`)
   (b) Optimisation et étude de performances
   (c) Benchmarks réalistes (e.g., BEEM, Model checking contest, etc) et études quantitatives
   (d) Livrables : Manuscrit de thèse ; article journal pour les résultats finaux ; code source du pipeline.

## Références


1. Ammann, P., Offutt, J. : Introduction to Software Testing. Edition 2. Cambridge University Press, New York, NY (2017)
2. Clarke, E.M., Emerson, E.A., Sistla, A.P. : Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS) **8**(2), 244–263 (1986)
3. Cohen, M.H., Serlin, Z., Leahy, K., Belta, C. : Temporal logic guided safe model-based reinforcement learning : A hybrid systems approach. Nonlinear Analysis : Hybrid Systems **47**, 101295 (2023)
4. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.L. : Equational unification and matching, and symbolic reachability analysis in maude 3.2 (system description). In : Blanchette, J., Kovács, L., Pattinson, D. (eds.) Proc. of IJCAR 2022. pp. 529–540. Lecture Notes in Computer Science, Springer (2022)
5. Duret-Lutz, A., **Klai, K.**, Poitrenaud, D., Thierry-Mieg, Y. : Self-loop aggregation product - A New Hybrid Approach to On-the-Fly LTL Model Checking. In : ATVA. LNCS, vol. 6996, pp. 336–350. Springer (2011)
6. Evangelista, S. : Experimenting with stubborn sets on petri nets. In : Petri Nets. LNCS, vol. 13929, pp. 346–365. Springer (2023)
7. Grigore, R., Yang, H. : Abstraction refinement guided by a learnt probabilistic model. In : Bodík, R., Majumdar, R. (eds.) Proc. of POPL. pp. 485–498. ACM (2016)
8. Lederman, G., Rabe, M.N., Seshia, S.A., Lee, E.A. : Learning heuristics for quantified boolean formulas through reinforcement learning. In : Proc. of ICLR 2020. OpenReview.net (2020)
9. Manna, Z., Pnueli, A. : The temporal logic of reactive and concurrent systems. Springer-Verlag New York, Inc., New York, NY, USA (1992)
10. Meseguer, J. : Conditioned rewriting logic as a united model of concurrency. Theor. Comput. Sci. **96**(1), 73–155 (1992)
11. Ouni, H., **Klai, K.**, Abid, C.A., Zouari, B. : Parallel symbolic observation graph. In : ISPA/IUCC. pp. 770–777. IEEE (2017)
12. **Arias, J.**, Bae, K., **Olarte, C.**, Ölveczky, P.C., Petrucci, L. : A rewriting-logic-with-smt-based formal analysis and parameter synthesis framework for parametric time petri nets. Fundam. Informaticae **192**(3-4), 261–312 (2024)
13. **Arias, J.**, Bae, K., **Olarte, C.**, Ölveczky, P.C., Petrucci, L., Rømming, F. : Symbolic analysis and parameter synthesis for networks of parametric timed automata with global variables using maude and SMT solving. Sci. Comput. Program. **233**, 103074 (2024)
14. **Klai, K.**, Abid, C.A., **Arias, J.**, Evangelista, S. : Hybrid parallel model checking of hybrid LTL on Hybrid State Space Representation. In : VECoS. LNCS, vol. 13187, pp. 27–42. Springer (2021)
15. **Klai, K.**, Bennani, M.T., **Arias, J.**, Desel, J., Ochi, H. : Symbolic observation graph-based generation of test paths. In : TAP. pp. 127–146. LNCS, Springer (2023)
16. **Klai, K.**, Bennani, M.T., **Arias, J.**, Ochi, H., Elouni, H. : Optimizing label coverage using regular expression-based linear programming. In : VECoS. pp. 16–31. LNCS, Springer (2024)
17. **Klai, K.**, Petrucci, L. : j construction of the symbolic observation graph. In : 2008 8th International Conference on Application of Concurrency to System Design. pp. 88–97. IEEE (2008)
18. **Klai, K.**, Poitrenaud, D. : Mc-sog : An ltl model checker based on symbolic observation graphs. In : International Conference on Applications and Theory of Petri Nets. pp. 288–306. Springer (2008)