Sujet de thèse : Aide à la conception sûre de programmes PLC à travers l'analyse statique

Directeurs de thèse : Eric ZAMAÏ en co-encadrement avec Emil DUMITRESCU

Contacts : eric.zamai@insa-lyon.fr ou emil.dumitrescu@insa-lyon.fr

Laboratoire d'accutil : AMPERE – Lyon

Etablissement : INSA de Lyon

Ecole doctorale : EEA

Financement : CIFFRE

Date de début souhaitée : 1/10/2022

Déroulement souhaité de la thèse : Grenoble et Lyon (campus de l'INSA)

Profil recherché : candidat.e avec un profil en informatique industrielle/automatique, connaissance des Automates Programmables Industriel et de leur programmation. Connaissances en logique des prédicats, intérêt pour les approches formelles (SAT, SMT, logique temporelle), une bonne pratique en progammation.

Contexte

L'avènement de l'Industrie 4.0 repose sur l'exploitation croissante de systèmes de production distribués et communicants, tout en étant porteur d'une exigence fondamentale : la personnalisation de masse. Une telle exigence induit un besoin accru de flexibilité des systèmes automatisés de production, requérant une adaptation régulière des fonctions présentes dans les automatismes industriels. Le cycle requérant une telle adaptation se raccourcit, à cause de contraintes accrues sur les délais de fabrication, entrainant des contraintes de réactivité sur la reconfiguration. Les changements dans la logique de contrôle-commande sont assortis d'un besoin critique en termes de :

- validation des nouvelles fonctions vis-à-vis des concepteurs du nouveau procédé à implanter ;
- vérification de ces fonctions, afin de s'assurer de manière objective qu'elles remplissent leur spécification.

Il s'agit d'un processus de conception laborieux, car les fonctions de contrôle commande régissent de plus en plus des systèmes de production distribués et communicants. La concurrence assortie d'exigences de synchronisation représente un défi à la fois pour la conception et pour la vérification, lorsque l'on suit une approche de développement conventionnelle, qui s'appuie sur des cycles de codage et de test. En effet, dans la mesure où les jeux de tests véritablement intéressants sont très difficiles à trouver par les concepteurs, une telle approche de développement n'apporte pas les garanties nécessaires de correction. C'est là que les techniques formelles ont apporté une contribution cruciale, en incarnant un paradigme de vérification ayant désormais acquis une maturité incontestable [1][2][3][4][5].

D'un autre côté, dans le domaine des automatismes industriels, se pose la question de l'extraction de la propriété intellectuelle disséminée dans un programme automate (API) en vue de sa réutilisation. Il s'agit là d'une démarche non pas de conception, mais plutôt de rétro-ingénierie, visant à extraire la fonction de contrôle-commande dans son essence, et en faisant abstraction des détails d'implémentation. Une telle démarche a pour objet la re-implémentation d'une fonction de contrôle-

commande existante, préalablement validée et vérifiée, au sein d'un API plus récent, ou encore d'une technologie différente.

Le contexte de ce travail se situe à la croisée de ces deux besoins propres aux concepteurs de programmes automate (API) : l'ingénierie, et la rétro-ingénierie [6].

Motivations

Tout d'abord, les techniques formelles offrent au concepteur la possibilité d'exprimer une spécification de manière rigoureuse [7] . Elles permettent également de formaliser le comportement du système étudié, pour d'une part prouver la validité de sa spécification et d'autre part de prouver la correction de son implémentation. Cette rigueur est indispensable, mais elle constitue aussi un frein : les techniques de preuve les plus puissantes [8] ont un pouvoir expressif largement suffisant pour traiter des problèmes courants d'ingénierie, mais leur mise en pratique nécessite une expertise pointue en termes de formalisation et de mise en œuvre d'une preuve. En plus, l'établissement d'une preuve n'est pas toujours automatique. En effet, on peut constater que les techniques de preuve les plus expressives et les plus génériques sont aussi celles qui nécessitent le plus d'expertise ; à l'opposé, nous trouvons des techniques, une en particulier, ayant fait une percée spectaculaire dans le monde de la recherche jusque dans l'industrie : la vérification de modèles, connue plutôt sous son nom anglophone *model checking*, ou encore *property checking[9].*

La question du compromis entre expressivité et facilité d'utilisation d'une technique formelle est devenue prégnante, car les utilisateurs et bénéficiaires ciblés ne sont, la plupart du temps pas des experts. Leur besoin se résume en cinq points :

- exprimer de manière à la fois rigoureuse et intuitive une spécification. Le cas échéant, exprimer des référentiels de spécifications réutilisables sur plusieurs systèmes qui doivent se partager les mêmes fonctionnalités ;
- concevoir le système ciblé en s'appuyant sur les approches/techniques/langages habituels ;
- établir une preuve - un lien formel entre la spécification et la conception ainsi obtenue - de manière transparente, sans effort supplémentaire de formalisation ;
- obtenir un retour lisible, à travers la preuve établie : soit une *confirmation (trace témoin)* du bon fonctionnement attendu, soit un *contre-exemple* montrant que ce fonctionnement n'est pas toujours garanti [10];
- obtenir un lien systématique entre le comportement observé sur un contre-exemple et la ligne de code responsable de ce comportement.

Le model checking a apporté des réponses à la plupart de ces attentes : il est automatique, intuitif, fournissant à la fois des contre-exemples et des témoins, tout en permettant aux concepteurs une expression intuitive des spécifications. Les domaines d'application privilégiés sont la vérification de circuits, du logiciel embarqué, des pilotes logiciels, mais aussi du code d'automates programmables.

Malgré de nombreuses promesses tenues, cette technique s'est heurtée à la limite dure de l'explosion combinatoire. Basée sur des algorithmes à complexité spatiale exponentielle (en nombre de variables), la limite pratique de son utilisation est apparue pour des systèmes comportant environ 200 variables Booléennes. C'est à la fois beaucoup, quand on pense à la taille de l'espace d'états potentiel (2^{200}), et peu, quand on considère le nombre limite de variables, qui correspond somme toute à un système de petite taille.

Les approches compositionnelles [11][12] ont permis un contournement efficace de cette limite. Basées sur le principe du « diviser pour régner », elles préconisent une approche en deux pas : *prouver* des spécifications sur des sous-ensembles, puis tirer des conclusions sur la globalité du système étudié. Le potentiel de cette approche est a priori illimité. Cependant, ces approches nécessitent un effort accru de la part du concepteur. Il y a d'abord le raisonnement compositionnel à mettre en place, comme une stratégie de vérification. Cela nécessite à la fois de l'expertise et du temps. Ensuite, il est indispensable de construire (manuellement) des *modèles complémentaires d'environnement*, pour accompagner la preuve de chaque sous-ensemble. C'est une étape incontournable, car la plupart du temps, un sous-ensemble ne fonctionne correctement que dans l'environnement pour lequel il a été conçu, et par conséquent, sa preuve ne peut pas ignorer les spécificités de cet environnement. Il s'agit d'un travail fastidieux, nécessitant des efforts manuels de modélisation, qui rallongent considérablement la durée de la vérification.

Ces limitations ont été repoussées par l'utilisation de techniques plus avancées d'exploration. SAT [13] et SMT [14][12] ont posé la fondation d'un paradigme différent, celui de l'exploration bornée : *bounded model checking*. L'expressivité des spécifications est davantage restreinte, mais le potentiel de preuve est amélioré de plusieurs ordres de grandeurs, passant de quelques centaines à dix mille variables. L'utilisation de ces techniques est tournée vers la preuve d'invariants, ou d'atteignabilité.

L'ensemble de ces éléments montrent assez clairement une tendance, fondée sur un constat : il semble aujourd'hui impossible de vérifier formellement l'ensemble des spécifications qu'il serait intéressant de prouver pour un système de grande taille, tout en respectant des contraintes sur le temps total de développement. Il apparait donc intéressant d'exploiter le potentiel du model checking, tout en acceptant de se restreindre dans son usage, afin de retrouver une réponse à la plupart des attentes énoncées plus haut : une technique de preuve à la fois automatique, déterministe dans le temps de preuve, et surtout intuitive et lisible pour le concepteur.

Bien que simples en apparence, il est difficile de garantir une satisfaction systématique de ces attentes. Les principaux verrous se situent à quatre niveaux :

- l'identification des spécifications formelles à la fois « intuitives » , utiles pour le concepteur, et dont la preuve reste soluble dans des temps acceptables. C'est difficile, car cela doit tenir compte, pour une même spécification typique, de la variabilité des programmes pouvant y répondre ;
- la construction automatique d'un modèle [15][16] pour la preuve, à partir du code écrit par les concepteurs. Cela requiert la plupart du temps des restrictions sur les constructions algorithmiques utilisées, voire sur les types de données, restrictions à définir en fonction du domaine d'application. Ce point est particulièrement important, la qualité de ce modèle est déterminante afin de garantir que le système implémenté correspond en totalité à ce qui a été simulé et formellement vérifié ;
- l'abstraction automatique d'un programme API afin d'en obtenir un modèle de fonction de contrôle-commande ouvert pour une vérification et/ou une ré-implémentation ;
- la recontextualisation du résultat « brut » de la vérification formelle, pour tenir compte de la spécificité des systèmes vérifiés (structure, modes de fonctionnement) mais aussi des habitudes des concepteurs, afin de leur offrir un retour intuitif et exploitable apportant des réponses concrètes sur la qualité du code qu'ils ont conçu.

Définition du sujet de thèse

Le projet présenté ici, porté par un partenaire industriel de la région Auvergne Rhône-Alpes et l'INSA de Lyon, se situe en prolongement de cette vision : exploiter le potentiel des techniques formelles existantes pour aider à la conception sûre de systèmes de contrôle-commande basés sur des automates programmables [17].

Ce sujet vise à donner certaines capacités d'analyse formelle à la technologie interne, actuellement exploité par le partenaire industriel. Elle permet de développer des outils de génie logiciel dans le domaine des Automates Programmables Industriels (API) en passant au-delà des incompatibilités de langages de chaque modèle. Les outils déjà développés comportent des analyseurs de logique pour plusieurs gammes d'API différentes, des générateurs de code, un outil d'analyse statique, un outil de génération de flots de contrôle et de flots de données en vue de rétro-ingénierie d'applications existantes et des embryons d'outils dont du test automatique et de la génération de code informatique (langage C).

Donner des capacités d'analyse formelle à la technologie permet de répondre à plusieurs cas d'utilisation déjà identifiés et d'ouvrir la technologie vers des outils beaucoup plus puissants, dans le domaine de la rétro-ingénierie et de la validation d'application. La transposition d'un outil de développement depuis l'informatique vers l'automatisme demande un gros travail d'adaptation dans le sens où les formations, les compétences des développeurs d'automates ne sont pas compatibles avec la complexité des solutions informatiques. La taille des projets d'automatisme bien que de plus en plus importante n'est pas compatible avec la spécialisation d'une ressource (ingénieur) pour utiliser des outils formels pointus. Ainsi une contrainte importante dans le cadre de ce projet est de rendre simple l'utilisation d'outils formels pour la rétro-ingénierie et la validation.

En perspective de la réussite de ces travaux, l'ouverture de la technologie propriétaire sur la diversité des automates devrait permettre de valider plus largement la pertinence des solutions. En effet la majorité des études sur le sujet se focalisent sur un ou deux types d'automates et seulement sur un des langages supportés.

L'objectif de ce projet est de définir des algorithmes d'exploration et de vérification formelle répondant aux besoins spécifiques des concepteurs de code API, quel que soit le langage d'entrée utilisé :

- calculer formellement l'intervalle d'évolution des variables incluses dans le programme PLC ;
- vérifier formellement les invariants identifiés, et fournir le cas échéant un contre-exemple ou une trace témoin ;
- établir un lien lisible entre les résultats « bruts » (contre-exemple, trace témoin) obtenus formellement et les spécificités du programme PLC, tenant compte des habitudes de codage du concepteur ;
- extraire un modèle formel à partir d'une implémentation.

Ce travail nécessite dans un premier temps une restriction du langage de programmation PLC aux seules constructions et types de données exploitables par des algorithmes d'exploration formelle. Le langage propriétaire, de par ses qualités de langage « pivot » entre la conception et l'implémentation du code semble un bon point de départ pour cela.

Dans un second temps, définir une démarche de vérification prenant en compte la réalité des *modes de fonctionnement*. Cette réalité offre un cadre permettant de décomposer le fonctionnement du

système, en suivant les transitions entre les modes, avec à la clé la possibilité de maitriser la complexité de la vérification.

Enfin, en fonction des spécifications identifiées, il s'agit de cibler les techniques d'exploration formelle les plus adéquates, de cibler leur possibles synergies, et de proposer une démarche méthodologique d'exploitation combinant leurs qualités, et offrant au concepteur des retours lisibles et intuitifs.

Les résultats attendus sont :

- identification des invariants simples et à valeur ajoutée pour les concepteurs, en fonction des grandes classes de programmes API existantes ;
- proposition d'un sous-ensemble basé sur le format propriétaire et compatible avec les techniques d'exploration formelle existantes ;
- proposition et implémentation d'un/des algorithme(s) de vérification formelle adaptés pour les invariants identifiés plus haut ;
- proposition d'un algorithme générique permettant d'extraire la fonction de contrôle-commande à partir d'un programme implémenté ;
- proposition d'un algorithme permettant de calculer l'intervalle effectif d'évolution d'une variable donnée.

Bibliographie

[1] A. Zeller, N. Jazdi, et M. Weyrich, « Functional verification of distributed automation systems », *Int. J. Adv. Manuf. Technol.*, vol. 105, nº 9, p. 3991-4004, déc. 2019, doi: 10.1007/s00170-019-03791-2.

[2] M. Rausch et B. H. Krogh, « Formal verification of PLC programs », in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, Philadelphia, PA, USA, 1998, p. 234-238 vol.1. doi: 10.1109/ACC.1998.694666.

[3] J. O. Blech, P. Lindgren, D. Pereira, V. Vyatkin, et A. Zoitl, « A Comparison of Formal Verification Approaches for IEC 61499 », in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany, sept. 2016, p. 1-4. doi: 10.1109/ETFA.2016.7733636.

[4] D. Darvas, I. Majzik, et E. Blanco Viñuela, « Formal Verification of Safety PLC Based Control Software », in *Integrated Formal Methods*, vol. 9681, E. Ábrahám et M. Huisman, Éd. Cham: Springer International Publishing, 2016, p. 508-522. doi: 10.1007/978-3-319-33693-0_32.

[5] M. Weiß, P. Marks, B. Maschler, D. White, P. Kesseli, et M. Weyrich, « Towards establishing formal verification and inductive code synthesis in the PLC domain », 2021, doi: 10.48550/ARXIV.2106.15878.

[6] B. Vogel-Heuser *et al.*, « Challenges for Software Engineering in Automation », *J. Softw. Eng. Appl.*, vol. 07, nº 05, p. 440-451, 2014, doi: 10.4236/jsea.2014.75041.

[7] Haniel Moreira Barbosa, « Formal verification of PLC programs using the B Method », Universidade Federal do Rio Grande do Norte, 2012.

[8] A. Rashid, U. Siddique, et S. Tahar, « Formal Verification of Cyber-Physical Systems Using

Theorem Proving », in *Formal Techniques for Safety-Critical Systems*, vol. 1165, O. Hasan et F. Mallet, Éd. Cham: Springer International Publishing, 2020, p. 3-18. doi: 10.1007/978-3-030-46902-3_1.

[9] E. M. Clarke, « The Birth of Model Checking », in *25 Years of Model Checking*, vol. 5000, O. Grumberg et H. Veith, Éd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 1-26. doi: 10.1007/978-3-540-69850-0_1.

[10] P. Ovsiannikova et V. Vyatkin, « Towards user-friendly model checking of IEC 61499 systems with counterexample explanation », in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vasteras, Sweden, sept. 2021, p. 01-04. doi: 10.1109/ETFA45728.2021.9613491.

[11] A. Zeller et M. Weyrich, « Component based Verification of Distributed Automation Systems based on Model Composition », *Procedia CIRP*, vol. 72, p. 352-356, 2018, doi: 10.1016/j.procir.2018.03.183.

[12] D. Bohlender et S. Kowalewski, « Compositional Verification of PLC Software using Horn Clauses and Mode Abstraction », *IFAC-Pap.*, vol. 51, n° 7, p. 428-433, 2018, doi: 10.1016/j.ifacol.2018.06.336.

[13] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, et Y. Zhu, « Symbolic model checking using SAT procedures instead of BDDs », in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, p. 317-320.

[14] H. I. Ismail, I. V. Bessa, L. C. Cordeiro, E. B. de Lima Filho, et J. E. Chaves Filho, « DSVerifier: A Bounded Model Checking Tool for Digital Systems », in *Model Checking Software*, vol. 9232, B. Fischer et J. Geldenhuys, Éd. Cham: Springer International Publishing, 2015, p. 126-131. doi: 10.1007/978-3-319-23404-5_9.

[15] M. Xavier, S. Patil, et V. Vyatkin, « Cyber-physical automation systems modelling with IEC 61499 for their formal verification », in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, Palma de Mallorca, Spain, juill. 2021, p. 1-6. doi: 10.1109/INDIN45523.2021.9557416.

[16] D. Drozdov, V. Dubinin, S. Patil, et V. Vyatkin, « A Formal Model of IEC 61499-Based Industrial Automation Architecture Supporting Time-Aware Computations », *IEEE Open J. Ind. Electron. Soc.*, vol. 2, p. 169-183, 2021, doi: 10.1109/OJIES.2021.3056400.

[17] E. A. Lee et S. A. Seshia, *Introduction to embedded systems: a cyber-physical systems approach*, Second edition. Cambridge, Massachuetts: MIT Press, 2017.