Méthodes formelles pour la conception, la programmation et la vérification de systèmes critiques émergents

Sylvain Conchon, Aurélie Hurault *
Alexandre Chapoutot, Pierre-Loïc Garoche*, Akram Idani*, Marc Pouzet, Matthieu Martel

Les méthodes formelles ont été élaborées depuis de nombreuses années afin d'assurer un niveau aussi élevé que possible en matière de précision et de fiabilité et ont ainsi montré que l'objectif du zéro-faute est réalisable pour des systèmes dits "fermés" (qui fonctionnent dans un environnement complètement maîtrisé). Cependant, les systèmes informatiques qui émergent aujourd'hui sont de plus en plus ouverts sur des environnements plus incertains, à temps continus ou basés sur de l'apprentissage de jeux de données difficilement prédictible. En outre, grâce aux progrès réalisés dans la miniaturisation des circuits intégrés et leur autonomie, de plus en plus de systèmes ouverts sont implémentés par des programmes complexes s'exécutant sur des circuits programmables qui s'apparentent à de véritables mini-ordinateurs (microcontrôleurs, FPGA, System on Chip). On retrouve ainsi ces programmes embarqués un peu partout dans les objets de notre quotidien, dans les *smartphones* ou l'électroménager, mais également dans les appareils médicaux, les voitures, les avions, etc.

La complexité de ces systèmes est évidente et la maîtrise des risques inhérents à leur utilisation est ainsi de plus en plus pressante. Il est clair que pour garantir leur sûreté et leur sécurité, l'adoption de moyens d'investigation indubitables et des techniques sûres et fiables reposant sur des fondements mathématiques s'impose. Aussi, les méthodes formelles ont-elles vocation à jouer un rôle d'envergure dans ce cadre avec des bénéfices indéniables. Ce faisant, pour être efficaces, les méthodes formelles devront inévitablement s'adapter au caractère ouvert, imprécis et intelligent de ces systèmes. Ce défi vise à faire progresser la science informatique, en particulier les méthodes formelles, pour la conception, la programmation et la vérification de ces systèmes critiques émergents.

1 Évolution des systèmes embarqués critiques

Les systèmes informatiques qui émergent aujourd'hui sont à la fois plus ouverts sur des environnements incertains, à temps continus ou basés sur de l'apprentissage de jeux de données, mais également conçus pour être exécutés sur des circuits électroniques programmables autonomes (microcontrôleurs, FPGA, System on Chip).

Systèmes hybrides ou cyber-physiques. Un système cyber-physique est un système dont le comportement ou la sémantique combine un aspect physique décrit typiquement par des équations différentielles (ordinaires ou algébriques) avec un calculateur contrôlé par un programme informatique. La physique Newtonienne décrit l'évolution du système physique par des équations qui dépendent du temps. Certains paramètres du modèle, comme justement le temps, évoluent dans des domaines continus ou denses. Le composant informatique, bien que réalisé par un mécanisme physique à base de transistors, a un comportement qui peut être décrit dans un monde plus discret. Le temps n'est plus

*Pour le groupe MFDL

dense mais cadencé par l'horloge de l'ordinateur. Les autres variables ou paramètres manipulés sont codés par des séquences de bits.

Systèmes avec apprentissage sur jeux de données massifs. Il est difficile aujourd'hui d'ignorer l'engouement pour l'intelligence artificielle et sa diffusion dans tous les domaines applicatifs, y compris les domaines critiques. De nombreux systèmes informatiques critiques reposent maintenant sur des réseaux de neurones entraînés à partir de grands jeux de données. Directement exécutées par des composants dédiés ou par des instructions machines directement implémentées dans les microprocesseurs, ces modules de *Machine Learning* fournissent des informations cruciales au système pour fonctionner, comme la reconnaissance de panneaux de signalisation dans une voiture, la reconnaissance faciale dans les smartphones, etc.

Systèmes électroniques programmables. Les progrès réalisés dans la miniaturisation des circuits intégrés sont incroyables. En quelques années, le nombre de transistors par puce est ainsi passé de quelques milliers à quelques dizaines de milliards. Cette révolution (qui suit la loi de Moore) a complètement bouleversé la manière de construire des systèmes électroniques. Là où on passait du temps hier à concevoir un circuit électronique complet dédié à une certaine tâche, on programme désormais un microcontrôleur, un FPGA voire un des ces "mini ordinateurs", appelés *System on Chip*, qui sont composés de CPU multi-coeurs cadencés avec des horloges de plusieurs gigahertz, disposant de capacités mémoire de plusieurs gigaoctets et accompagnés d'un ensemble de co-processeurs (graphique, etc.) et de périphériques (modems, wifi, capteurs CCD, etc.). Ces circuits électroniques programmables d'une centaine de millimètres carrés se retrouvent partout dans notre quotidien (pour preuve, il se vend près de 30 milliards de microcontrôleurs par an) et le développement des logiciels embarqués dans de tels composants représente un coût de plus en plus important dans l'industrie et leur maintenance est parfois difficile voire impossible.

2 Applications

Véhicules autonomes. La composante logicielle domine le véhicule de demain, c'est pourquoi l'usage de méthodes formelles est un enjeu majeur pour leur sûreté. Les défis relevés par ces méthodes durant les années à venir porteront sur plusieurs axes de recherche : la conception de systèmes temps réels embarqués, la correction des algorithmes décisionnels, la tolérance aux fautes dans un environnement incertain, la sécurité, etc. Dans la suite nous focalisons notre réflexion principalement sur deux verrous scientifiques majeurs.

Systèmes électroniques programmés de contrôle-commande. Aujourd'hui de nombreux systèmes électroniques sont pilotés et synchronisés par du logiciel. Un exemple d'architecture consiste à implanter des boucles de commande dans des circuits logiques programmables et des modules chargés d'assurer la coopération entre ces boucles de contrôle. La méthode de développement de ces systèmes suit habituellement une approche *model-based design* se basant sur des modèles Matlab/Simulink ou Scade dont du code C (ou C++) est extrait (automatiquement) par un compilateur. Ce code est ensuite lié à des programmes écrits à la main avant d'être traduit vers un langage de description de matériel tel que VHDL ou Verilog pour être enfin exécuté par des microcontrôleurs ou FPGA.

Il est important de développer des techniques et des outils pour gagner en confiance sur ces modules de contrôle-commande ainsi que sur les modules de coopération. Bien que les approches existantes reposent sur une bonne méthodologie basée modèles et sur de la génération de code, il nous

semble que l'écriture de ces systèmes dans des langages synchrones hybrides avec une sémantique bien définie, ainsi que des outils de méthodes formelle pour vérifier certaines propriétés fonctionnelles, permettrait de trouver des bugs en phase amont et de mieux appréhender la complexité de ces architectures.

3 Verrous scientifiques et techniques à lever

Le cadre scientifique général de notre défi est résumé dans le schéma de la figure 1. À droite du schéma se trouve l'implémentation d'un système embarqué critique. On y trouve des plateformes matérielles qui peuvent inclure un système d'exploitation temps-réel ou des circuits intégrés programmables. Les programmes considérés à ce niveau sont principalement écrits en assembleur ou dans des langages de description de matériel comme VHDL ou Verilog. Ces programmes interagissent avec leur environnement, en entrée à travers des capteurs, et en sortie à l'aide d'actionneurs sur lesquels ils envoient des ordres de contrôle. Les nouveaux systèmes embarqués peuvent également reposer sur l'utilisation de processeurs spécialisés qui exécutent par exemple des réseaux de neurones pour de la reconnaissance d'image. Le cadre de gauche contient quelques langages, modèles et outils utilisés pour concevoir ces systèmes critiques. Certaines de ces méthodes de conception sont exécutables, c'est-à-dire qu'elles permettent de produire des programmes qui peuvent être donnés en entrée à des compilateurs. Ces derniers peuvent alors produire du code plus ou moins efficace pour les plateformes matérielles, avec plus ou moins de confiance dans leur correction et efficacité. D'autres méthodes de description ne permettent pas cette génération de code mais elles peuvent être plus adaptées pour mener certaines analyses de fiabilité du système. Qu'elles soient exécutables ou non, les méthodes utilisées pour décrire le comportement d'un système critique doivent inclure une description formelle et suffisamment détaillée de son environnement, qu'il soit à temps continu ou discret, ou qu'il implique une partie basée sur de l'apprentissage à partir de jeux de données massifs. Le cadre du haut de la figure rassemble les techniques en méthode formelle couramment utilisées dans la vérification formelle de systèmes critiques. On y trouve des techniques de preuve déductive, de *model checking*, de test ou d'interprétation abstraite. Pour garantir la plus grande fiabilité des systèmes critiques, il faut que ces techniques s'appliquent partout, aux modèles formels, aux codes cibles sur les plateformes matérielles et aux compilateurs.

Ce cadre général soulève de nombreux verrous scientifiques qu'il est nécessaire d'attaquer afin d'améliorer les méthodes formelles pour la conception, la programmation et la vérification de systèmes critiques émergents. Les domaines de recherche liés à ce défi sont multiples. Une liste non exhaustive de thématiques est donnée ci-dessous.

Systèmes hybrides. Les outils mathématiques pour la modélisation et la vérification des systèmes discrets sont très différents de ceux à temps continu. Les premiers s'appuient plutôt sur l'algèbre générale et la logique, avec des raisonnements basés sur le principe d'induction ; les seconds sur la topologie et l'analyse. Même lorsque les systèmes à temps continu sont discrétisés, les modèles obtenus sont souvent trop complexes et peu intuitifs pour être traités efficacement par les méthodes des systèmes discrets. Par exemple, la représentation des réels par des flottants ou des encodages à base d'entier, rend les analyses difficiles. La complexité entraîne des problèmes de passage à l'échelle des outils de vérification automatique et le côté peu intuitif de certains modèles ne permet pas aux utilisateurs des méthodes automatiques ou semi-automatiques de construire des preuves rapidement. L'étude des systèmes hybrides désigne parfois l'étude de systèmes physiques dont le comportement est décrit par une combinaison discrète (typiquement finie) de dynamiques à temps continu. Même

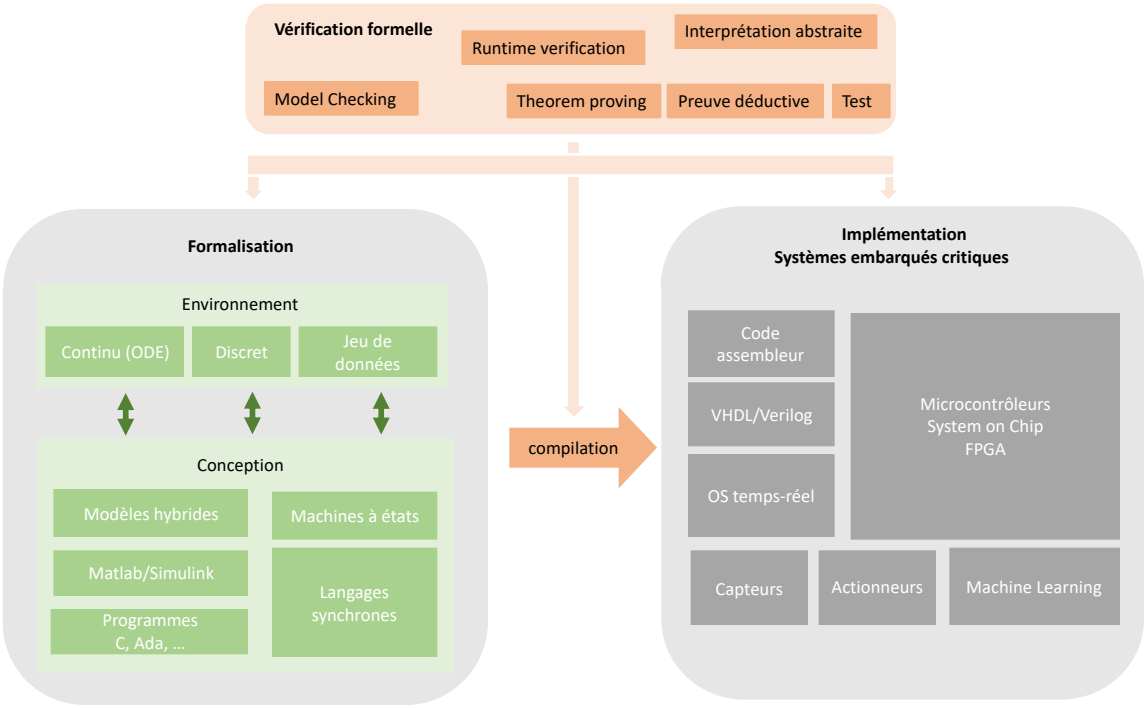

FIGURE 1 – Approche pour la formalisation et vérification de systèmes critiques embarqués vers de l'électronique programmée

si plusieurs travaux ont montré la faisabilité de développement formel de systèmes hybrides, les méthodes et outils actuellement disponibles manquent encore de maturité par rapport aux méthodes et outils dédiés aux systèmes exclusivement discrets ou continus. Ainsi il est difficile de plonger un système cyber-physique même basique, comme par exemple une boucle fermée entre un balancier inversé et un contrôleur linéaire avec saturations, dans ces formalismes.

Langages synchrones et leurs extensions aux systèmes hybrides Les langages synchrones comme Scade et leurs extensions aux systèmes hybrides comme Zélus pourraient être utilisés pour écrire des programmes qui ne sont pas facilement modélisables en Matlab/Simulink, comme des contraintes de synchronisation ou certaines machines à états. Leur expressivité permet d'exprimer des spécifications mathématiques exécutables très précises. Leur sémantique et leur compilation précisément définies assure que le code (pour l'exécution ou la simulation) reproduit fidèlement ce qui est écrit dans le modèle pour afin d'assurer que les propriétés vérifiées sur le modèles sont préservées sur l'implémentation.

Test de systèmes hybrides. Les techniques de test basées sur des propriétés (PBT), appelées également QuickCheck, semblent pertinentes dans ce contexte. Elles procèdent par génération aléatoire d'entrées filtrées par la propriété à tester. Pour les systèmes envisagés, cette génération devrait faire appel à un solveur de contraintes (par ex. SMT) qui saura traiter des équations différentielles et générer (et faire varier) des modèles. Une autre approche à développer consiste à utiliser les domaines abstraits de l'analyse statique pour construire des trajectoires abstraites. Ces trajectoires ensemblistes correspondent à des tests ou simulations à horizon borné mais capturent toutes les trajectoires faisables. Ces méthodes pourraient être

développées et appliquées dans ce contexte de systèmes hybrides complexes combinant des sémantiques à temps continu décrites par des EDO avec des programmes ; ces derniers pouvant décrire autant des composants logiciels que matériels.

Model Checking de systèmes hybrides. Un des enjeux important de ce défi est la vérification exhaustive de propriétés sur les modèles hybrides. Des approches comme dReach proposent de combiner satisfiabilité et résolution d'ODE pour raisonner sur l'atteignabilité de propriétés sur des systèmes hybrides. Pour résoudre les problèmes de précision et de passage à l'échelle, il pourrait être pertinent d'intégrer plus finement ces deux méthodes.

Analyse numérique. Les calculs décrits dans des outils comme Simulink sont exprimés dans le corps des réels, sans considérer les détails d'implémentation. Or, lors de la production du code, ces calculs doivent être implémentés en utilisant des nombres à virgule fixe ou flottante. Les difficultés concernent la précision des calculs du code objet et de son efficacité en termes d'espace utilisé sur les FPGA. Il nous semble important de travailler à des outils de synthèse de code numérique de bas niveau, en virgule fixe et flottante, qui pourront prendre en compte la précision et la contrainte d'espace en s'appuyant sur des techniques d'analyse statique et de transformation de programmes.

Systèmes avec apprentissage sur jeux de données massifs. Il est indispensable d'avoir des garanties sur les algorithmes d'intelligence artificielle (IA) pour pouvoir bénéficer des avancées de ce domaine en évitant ses dangers. Pour y arriver, plusieurs disciplines doivent cohabiter : économie, éthique, juridique, sécurité, méthodes formelles [2]. Il existe de nombreux verrous pour la vérification des algorithmes / systèmes d'IA liés aux caractéristiques de ceux-ci :

Spécification : les algorithmes / systèmes d'IA sont souvent peu ou mal spécifiés. Ils consistent à reconnaître des patterns dans les données d'entrée. L'algorithme d'IA est performant justement car on ne sait pas caractériser algorithmiquement / formellement les données d'entrée.

Explicabilité : bien souvent les spécialistes de l'IA eux-mêmes ont du mal à expliquer (en langage humain ou mathématiques) pourquoi les algorithmes donnent une réponse.

Reproductibilité : un même apprentissage répété plusieurs fois peut construire des IA ayant des comportements différents (part d'aléatoire dans l'apprentissage).

Prédictabilité et robustesse : face à une même (ou presque) configuration, il n'y a pas (toujours) de garanties d'avoir la même réponse en sortie.

La preuve de correction d'un algorithme est une preuve par rapport à une spécification. Sans spécification, pas de preuve. Il faudra donc trouver une façon / un langage / un formalisme /... pour décrire la spécification d'un algorithme d'IA. Il est déjà difficile d'assurer la correction de programmes que l'être humain écrit en se basant notamment sur des algorithmes. Comment garantir la correction de programmes qui ne sont plus écrits mais appris ? Pour contourner ce problème d'explicabilité, ainsi que ceux liés à la reproductibilité, la prédictabilité et la robustesse, une certification à postériori peut être plus adaptée : au lieu de certifier l'algorithme lui-même, chaque sortie est accompagnée d'un certificat de validité qui peut être rejoué.

Véhicules autonomes. Bien que les techniques de preuve (theorem proving et model-checking) et de vérification à l'exécution (runtime verification) - largement adressées par la communauté des méthodes formelles - soient incontournables, elles s'avèrent insuffisantes pour ces systèmes étant donné le côté imprévisible du comportement du véhicule. Ce dernier, ayant la responsabilité de reconnaître son environnement, est amené à engager en conséquence des actions qui se doivent d'être sûres. Ceci n'est pas sans failles car la reconnaissance de l'environnement se base sur des techniques qui

manquent parfois de précision tels que l'apprentissage, la prédiction ou encore l'approximation. À titre d'exemple, un rapport public dressé par le département des transports américain [1] désigne des failles de sûreté observées sur des véhicules Tesla et qui indiquent des comportements non conformes à ceux normalement attendus : "*The Automatic Emergency Braking or Autopilot systems may not function as designed, increasing the risk of a crash*". Il est important de distinguer la précision des informations issues du monde réel, et la justesse des actions engagées suite à ces informations. L'application d'une approche formelle donne aujourd'hui des solutions quant à la sûreté des actions entreprises par un automatisme ; en revanche, cela est souvent fondé sur l'hypothèse que les données en entrée sont bien précises. Ceci n'étant pas le cas du véhicule autonome, des techniques de supervision, elles-mêmes prouvées correctes, permettant d'évaluer la précision et la justesse des informations environnantes au système, sont devenues de mise.

La conduite est un processus qui se veut social car il implique des interactions parfois intenses et complexes avec des humains : autres conducteurs, cyclistes, piétons, etc. De nombreux travaux montrent qu'il est très difficile de remplacer les décisions de nature humaine par des décisions issues d'algorithmes vu que les humains s'appuient sur une intelligence généralisée et sur le bon sens. Dans un monde où le véhicule interagit très peu (voir pas du tout) avec des humains lors de la prise de décision, comme dans le cadre d'un train sans conducteur ou d'un pilote automatique d'avion, les techniques formelles ont montré leur efficacité. Néanmoins, dès lors que la prise de décision est collective et est le fruit d'interactions sociales, il devient nécessaire d'envisager une multitude de scénarios. Google par exemple, a mené des travaux pour que le véhicule puisse reconnaître un cycliste, interpréter ses signaux gestuels et prédire son intention (*e.g.* dans quel sens il envisage de tourner). Pour garantir la sûreté d'un tel système, les approches formelles se doivent de proposer des mécanismes qui couvrent la modélisation du comportement humain en plus de celui du système ainsi que les liens entre eux. Cela aura un impact direct sur les techniques de vérification actuelles (tel que le model-checking borné et/ou symbolique, la simulation, etc) qui devront être étendues non seulement pour exhiber les failles possibles, mais aussi pour élaborer des niveaux de confiance et/ou de tolérance, et identifier les responsabilités qui pourraient découler de ces failles. En effet, l'incertitude et l'hostilité de l'environnement dans lequel ces véhicules seront plongés, donnera une nouvelle vision à la cyber-sécurité et à la surveillance des usagers, et impactera de fait le cadre juridique et éthique de la société d'aujourd'hui.

4 Projet nationaux et internationaux sur le thème

Projets sur le thème des systèmes cyber-physiques

- Projet ANR DISCONT - Correct Integration of Discrete and Continuous Models
- Projet ANR EBRP PLus
- Projet ANR JCJC FEANICSES - Formal and Exhaustive Analysis of Numerical Intensive Control Software for Embedded Systems
- Projet IPL INRIA ModeliScale
- Projet ARTEMIS UnCoVerCPS - Unifying Control and Verification of Cyber-Physical Systems
- Projet ANR Cafein - Combinaison d'approches formelles pour l'étude d'invariants numériques

Projets sur le thème de l'IA et des méthodes formelles

- Au niveau national

    - Une recherche sur le site de l'ANR permet de se rendre compte de la difficulté du problème : aucun projet n'est financé sur ce sujet, probablement car le domaine n'est pas assez mûr.
    - Dans le cadre du Labex DigiCosme, une école d'été (ForMaL) a été organisée sur le thème des méthodes formelles et du machine learning pour aider à la création de synergies entre ces deux mondes : `https://formal-paris-saclay.fr/`
- Au niveau international
    - "Summit on Machine Learning Meets Formal Methods", 13 Juillet 2018, `http://www.floc2018.org/summit-on-machine-learning/`, en marge de la conférence FLoC.
    - En 2017, l'école d'été de Dagstuhl était sur le thème des méthodes formelles et du machine learning : `https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=17351`

Projets sur le thème des véhicule autonome Il existe une foultitude de projets sur le thème du véhicule autonome avec un intérêt manifeste de la part des secteurs publics et privés. Nous donnons quelques pointeurs vers des projets qui font usage de méthodes formelles dans le développement de ces systèmes :

- TrustMeIA (Toulouse) `https://www.laas.fr/projects/trustmeia/`
- Programmes "Train Autonome" de l'IRT Railenium et Tech4Rail de la SNCF
- SynC Contest : Automatic Driving Challenge using Model-based Design and Synchronous Programming
- TraCE-IT : Train Control Enhancement via Information Technology (`http://traceit.isti.cnr.it`)

Références

[1] National Highway Traffic Safety Administration. Automatic vehicle control systems - ODI Resume. Technical report, U.S. Department of Transportation, 2017. `https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.PDF`.

[2] Stuart J. Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4), 2015.