Ksass'Peuk

Zeste de savoir

La recherche en méthodes formelles, pour les 30 années à venir

7 mai 2019

Table des matières

% LA RECHERCHE EN MÉTHODES FORMELLES, POUR LES 30 ANNÉES À VENIR % Ksass'Peuk % 14 mai 2018

1. Introduction

Il y a quelques temps, je suis allé à la conférence NASA Formal Methods 2018 . Cette conférence, organisée par la NASA (National Aeronautics and Space Administration), regroupe chaque année des chercheurs des mondes académique et industriel autour de la thématique des méthodes formelles. Ces méthodes sont utilisées, en informatique, pour raisonner mathématiquement à propos des programmes et systèmes afin d'obtenir de très fortes garanties de fiabilité.

Dans les conférences scientifiques, il y a très généralement un ou plusieurs chercheurs qui sont invités pour parler d'un sujet qui leur importe particulièrement. Cette année, la présentation de Gilles Dowek, chercheur au Laboratoire Spécification et Vérification de l'ENS Cachan, a particulièrement retenu mon attention. Cette présentation tentait de répondre à la question "que va-t-il se passer pour les méthodes formelles dans les 30 années à venir ?".

Dans ce billet, je vous propose, après une mise en contexte, une retranscription de ce que j'ai retenu de cette présentation, agrémentée de quelques notes et références pour aller plus loin ou simplifier certaines idées.

Merci à **@gbdivers** pour la relecture.

2. Un peu de contexte

2.1. Méthodes formelles

Les méthodes formelles, en informatique, permettent de raisonner rigoureusement, mathématiquement, à propos de programmes ou plus généralement à propos de systèmes qui traitent automatiquement de l'information.

Par l'utilisation de tests, nous ne pouvons généralement pas montrer qu'un programme ne peut avoir que de "bons" comportements, dans la très vaste majorité des cas, nous ne pouvons tester qu'un nombre restreint d'entrées possibles et pas toutes les entrées possible d'un programme. À l'inverse, par l'utilisation de méthodes formelles, nous pouvons par exemple montrer mathématiquement qu'un programme écrit dans un langage donné ne peut avoir que de "bons" comportements et qu'aucun bug ne peut survenir à cause du programme que l'on a écrit.

Par exemple, si l'on considère un programme qui trie une liste de valeurs, on ne peut pas tester le programme sur toutes les listes possibles de valeurs qui existent : il y en a une infinité. Nous ne pourrons donc tester le programme que pour un certain échantillon de listes que nous devrons choisir avec attention. À l'inverse, par preuve mathématique, on sera capable de montrer que pour toute entrée possible, le programme que nous avons écrit produit nécessairement la bonne sortie (ce qui ne veut cependant pas dire que nous sommes sûr à 100% que le compilateur ne va pas faire d'erreur en traduisant le code, ou encore que le système d'exploitation ou les bibliothèques que nous utilisons n'en feront pas).

L'utilisation de ce type de méthode est revanche généralement assez coûteux à la fois en ressources humaines et en calculs. Par conséquent, on les réserve généralement à des domaines où les erreurs sont très critiques, comme par exemple en aéronautique, dans le ferroviaire, ou encore dans l'aérospatial.

Lorsqu'un programme comme un jeu vidéo a un bug, la majorité du temps, ce n'est pas très grave, même si cela peut agacer l'utilisateur. Et pour corriger le bug c'est facile : un patch de mise à jour et c'est terminé. À l'opposé, une fois que l'on a posé un rover sur Mars, il est un peu plus difficile de re-démarrer le système s'il crash, et faire une mise à jour est clairement une autre paire de manches que sur terre.

2.2. NASA Formal Methods

La NASA évolue dans un secteur qui a typiquement besoin de méthodes formelles. Elle investit donc depuis 30 ans dans ce domaine, même si ces financements ont d'abord été très difficiles à obtenir, puis à maintenir, car les premiers projets ont reçu un accueil assez circonspect, principalement à cause d'attentes trop ambitieuses pour un domaine qui n'en était qu'à ses débuts. Par la suite, les projets ont connus progressivement de plus en plus de succès, ce qui a entraîné qu'aujourd'hui, les méthodes formelles sont l'un des nombreux domaines de recherche de la NASA.

Depuis 10 ans maintenant, la NASA organise sa propre conférence à propos des méthodes formelles. Dans cette conférence, une place importante est faite aux diverses (et nombreuses) collaborations internationales de la NASA. Et cette année, Gilles Dowek, qui collabore sur

divers projets de la NASA depuis maintenant 15 ans, était l'un des présentateurs invités de la conférence.

2.3. Gilles Dowek

Gilles Dowek est un chercheur du Laboratoire Spécification et Vérification de l'ENS Cachan. Il s'intéresse à la formalisation des mathématiques (théorie des types, théorie des ensembles, etc), aux systèmes de calcul de preuve (vérification de preuve, preuve automatique de théorèmes, etc), et au domaine de la sûreté des systèmes aérospatiaux.

Le Dr Dowek s'intéresse également à l'enseignement et a notamment poussé le secteur de l'éducation (en France) à porter son attention sur le besoin d'introduire des notions basiques de science informatique dès le collège ou avant. Il a été membre du comité formé par le ministère de l'éducation pour produire un programme de science informatique au lycée, ce qui a mené au programme publié au Bulletin officiel le 13 octobre 2011. Il a participé au rapport de l'Académie des Sciences "L'enseignement de l'informatique - Il est urgent de ne plus attendre".

Il est membre du conseil scientifique de la Société Informatique de France et du CERNA, Commission de réflexion sur l'éthique de la recherche en sciences et technologies du numérique d'Allistene, l'Alliance des sciences et technologies du numérique. Par le passé, il a été responsable scientifique à Inria, en charge du domaine Algorithmique, Programmation, Logiciel et Architectures.

Gilles Dowek est l'auteur de plusieurs livres scientifiques populaires, il est également intéressé par la philosophie des sciences. La présentation retranscrite par la suite s'intéresse d'ailleurs de près aux questions éthiques auxquelles nous allons devoir faire face dans les années à venir en informatique.

3. A propos des méthodes formelles, pour les 30 prochaines années

Avant toute chose, ce que je propose ici est une retranscription de ce que j'ai noté et compris des idées exposées par Gilles Dowek pendant sa présentation. Ce n'est cependant pas une retranscription directe de la conférence, il contient quelques explications supplémentaires et références. J'ai également le biais d'être déjà convaincu que les méthodes formelles peuvent apporter beaucoup à tous les domaines de l'informatique.

La thématique de la présentation de Gilles Dowek est importante. Même si faire des prédictions est difficile (surtout quand elles concernent l'avenir *sic*), il est important de réfléchir à ces questions car elles servent de guide à la recherche.

3.1. Comprendre ce que font les ordinateurs

Avant de pouvoir faire une prévision sur l'avenir des méthodes formelles, il convient d'abord de se poser une autre question : pourquoi avons-nous besoin des méthodes formelles ?

La réponse proposée par Gilles est très simple : pour comprendre ce que font les ordinateurs. Cette réponse a retenu mon attention déjà bien éveillée, car c'est la raison pour laquelle j'ai voulu faire des méthodes formelles.

Nous sommes aujourd'hui entourés d'ordinateurs et nous interagissons en permanence avec ceux-ci. Les ordinateurs font des choses dont nous ne sommes pas capables. Ou plutôt dont nous ne sommes pas capables de manière aussi efficaces, ou avec une aussi bonne fiabilité. Et la raison est assez simple : si nous en étions capables aussi efficacement, nous n'utiliserions pas d'ordinateurs.

L'exemple avancé dans la présentation est le suivant : en tant qu'humains, si l'on nous demande quelle est la millième décimale de pi, nous ne savons pas répondre, et pourtant l'ordinateur le calcule de manière quasi-instantanée et l'on accepte ce résultat (si l'on a suffisamment confiance dans le programme que l'on utilise).

D'un côté nous voulons être surs que l'ordinateur nous transmet bien le bon résultat, mais en même temps nous ne voulons pas être capables de prévoir le résultat parce que sinon nous n'aurions pas besoin de l'ordinateur. Et c'est là tout l'objectif des méthodes formelles : être capable de comprendre sans que cela ne nécessite de nous de prévoir le comportement exact du système.

3.2. Sûreté et éthique des programmes : deux raisons pour l'usage de méthodes formelles

Pouvons-nous faire confiance à une boîte noire si l'on ne sait pas ce qu'elle fait ? Sur ce point les questions éthiques et de sûreté sont toutes les deux importantes.

Les questions de sûreté sont souvent admises comme de très bonnes raisons de savoir ce que font les machines (comme dit plus tôt dans cet article, les systèmes de transports, ou l'armement ont intégré ce besoin depuis longtemps), en revanche ce n'est pas encore si souvent le cas pour les questions éthiques.

Si l'on a une machine qui pose automatiquement divers verdicts (amendes, prêts, notes), nous avons le droit de savoir **pourquoi ?** : la loi qui gouverne l'algorithme doit être publique (*nul n'est censé ignorer la loi*), et cela n'impose pas que l'algorithme le soit. En effet, les explications nous permettent de repérer les décisions arbitraire (pour pouvoir nous défendre). Elles nous permettent aussi de progresser, de ne pas refaire les mêmes erreurs.

Dans les deux cas, nous avons plus de difficulté à avoir confiance dans une décision lorsqu'elle est prise par une machine que lorsqu'elle est prise un humain. Pour être vraiment accepté par un humain comme un bon système, la machine a souvent besoin d'être plus de 10 fois plus sure qu'un humain.

Par exemple, récemment, une voiture autonome a percuté et tué un piéton, et tous les journaux du monde ont titré cette nouvelle. Alors que le même jour, probablement beaucoup plus de piétons ont été tués par des humains. Et c'est le cas tous les jours, mais nous l'acceptons (en 2016, pour la France seule, 559 piétons sont morts dans un accident de la route[12]).

Un autre axe de réflexion est l'impartialité de l'algorithme. On veut d'un algorithme que ses décisions soient éthiques, et qu'elle ne soit pas arbitraires ou discriminatoires par exemple. Dans le même temps, une libération de prison prononcée par un juge humain avant le repas a moins

de 20% de chances d'être acceptée contre plus de 60% juste après le repas[3], ce qui est tout sauf impartial, donc un algorithme n'aura pas un mal fou à se montrer plus impartial que cela.

3.3. À propos des spécifications

Dans l'introduction, j'ai indiqué que les méthodes formelles nous permettent de montrer qu'un programme ne peut avoir que des "bons comportements". Il convient maintenant de préciser que le développeur doit par contre préciser ce qu'est un "bon comportement".
Dans le cas des tests, ce sont les sorties ou erreurs que l'on attend en fonction des entrées que l'on a choisies pour les tests. Par exemple, si je donne une liste $[3, 4, 2, 1]$ en entrée d'un tri, je veux une liste $[1, 2, 3, 4]$ en sortie.
Dans le cas des méthodes formelles, on fournit une spécification formelle, des formules mathématiques, qui décrivent ce qu'est une entrée valide, et ce qu'est une sortie valide. Par exemple, pour le tri, nous exprimerions mathématiquement : *pour toute liste d'entrée, je veux une liste de sortie qui contient toutes les valeurs de la liste d'entrée, telle que pour toute paire d'éléments consécutifs dans la liste, le premier élément est plus petit que son suivant.*

Comprendre ce que fait un ordinateur n'est pas être capable de prévoir ses résultats : on n'aurait pas besoin d'ordinateur dans ce cas. Ce n'est pas non plus être capable de dire comment le résultat est produit : cela ne nous dit pas ce que signifie le résultat.

Comprendre ce que fait l'ordinateur c'est être capable de prédire les propriétés du résultat.

Par exemple :

— "le résultat est la 1000ème décimale de pi"
— "dans mon contrôle de trafic aérien, il ne peut pas y avoir de conflits"
— "l'acceptation d'un prêt ne dépend pas de la couleur de peau du demandeur"

Écrire des spécifications (formelles) est très utile. En nous demandant ce que l'on cherche **exactement** à faire, nous obtenons un guide précis pour nous aider à le faire, le comprendre et le vérifier. Des spécifications formelles, mathématiques, présentent en plus la qualité d'être beaucoup moins sujettes à interprétation qu'un texte écrit en langue naturelle.

Cependant, cela peut déjà être complexe. Par exemple, définir précisément "ne pas y avoir de conflit" est nécessaire, car cette propriété en elle-même n'est pas suffisamment précise pour que l'on comprenne réellement ce qu'elle demande et implique. Pire, "ne pas faire de discrimination" implique de définir formellement la notion de "discrimination" ce qui est une tâche ardue.

3.4. Sûreté et éthique : des réponses formelles probablement différentes

Pour traiter cette question, Gilles Dowek explique d'abord la différence entre compilateur certifié et certifiant (qu'on peut étendre plus globalement aux programmes, de manière générale).

Dans le cas d'un compilateur certifié, on a une garantie formelle que pour tout programme transmis en entrée, la sémantique du programme de sortie est exactement celle que l'on a demandé. À savoir une preuve mathématique que pour toute entrée, on la compilera correctement.

Dans le cas d'un compilateur certifiant, nous avons la garantie que pour tout programme que l'on donne en entrée, un certificat sera produit pour nous justifier que la sémantique de la sortie est la même que celle de l'entrée. Et nous pouvons ensuite vérifier que ce certificat est correct.

Un compilateur certifié est certifiant (car l'on a la garantie que la sortie ne peut qu'être correcte). Un compilateur certifiant est presque certifié, car on peut vérifier au cas par cas lorsque l'on a besoin.

Des banquiers, des juges ou des professeurs automatiques devraient être certifiant. De cette manière, en cas de décision erronée, on pourrait pointer ce qui n'est pas acceptable dans le certificat.

Des systèmes autonomes de transports devraient être certifiés, car si une commande d'avion ne fait pas son travail correctement en plein vol, il sera sûrement trop tard pour aller voir ce qui ne va pas dans le certificat.

3.5. Expliquer un programme peut être difficile

3.5.1. Exemple : prévisions météorologiques

Quand un calcul nécessite des heures de calculs, des milliards d'opérations sont exécutées. Chacune d'elle est facile à expliquer, mais rassembler ces explications ne nous permet pas d'obtenir une explication globale de ce que fait le système.

Par exemple, si le système nous indique 32°C à une position géographique et pas 31°C, il est difficile de définir et d'expliquer exactement le pourquoi. Même si l'on est capable d'expliquer la propriété du résultat (c'est la température que l'on attend à cet endroit), il serait compliqué d'expliquer pourquoi ce résultat est correct.

Les programmes de calculs météorologiques peuvent être des boîtes noires. Le modèle du climat ne devrait pas l'être.

3.5.2. Exemple : un algorithme d'apprentissage

En tant qu'humains il nous est très facile d'apprendre très rapidement la différence de style entre deux artistes s'ils appartiennent à des courants bien distincts (et ce serait aussi le cas pour des machines aujourd'hui). En revanche définir ce qui nous permet ensuite de différencier ces deux artistes en regardant une œuvre est beaucoup plus difficile.

Nous pouvons définir des notions de distance entre les images, de température, comment l'œuvre transmet une émotion particulière, mais il est bien plus complexe de donner une formalisation à cela.

De plus, entraîner un algorithme avec une base de données dont les valeurs sont biaisés peut amener à des résultats eux aussi biaisés. Mais ces biais ne viennent pas tant de l'algorithme que des données qui ont été utilisées. On pourrait même donner une explication du biais, mais les résultats resteraient faux par rapport au besoin.

3.6. Quatre concepts en science informatique

Ces quatre concepts sont fixés empiriquement par le fait que chacun arrive à tomber dans un cas qui lui convient.

— Algorithme
— Machine
— Langage
— Donnée

Pour les trois premières notions, nous avons déjà de nombreux résultats formels, avec des programmes, des architectures de processeur ou encore des langages spécifiés et prouvés corrects. Mais ce n'est pas encore vraiment le cas pour les données, même pas d'un point de vue spécifications.

3.7. Conclusion de la présentation

Nous avons besoin de plus de méthodes formelles parce que nous avons besoin de plus d'explications à propos des systèmes qui nous entourent et prennent une grande place dans notre vie. Nous n'en avons pas seulement besoin pour la sûreté et la sécurité, mais aussi pour des questions d'éthiques. Nous n'en avons pas seulement besoin pour les algorithmes, les machines et les langages, mais aussi pour les données.

4. Conclusion

Les méthodes formelles sont de plus en plus utilisées et s'attaquent aujourd'hui à de nouveaux défis. Outre le niveau sûreté de plus en plus élevé que l'on parvient à atteindre dans les domaines critiques, ces méthodes sont également de plus en plus utilisées pour des tâches sembables mais pour des cas moins critiques (internet de objets, protocoles réseaux, ...).

Cette présentation aborde deux autres aspects de la mutation que subit le domaine. D'abord, avec l'arrivée de systèmes automatiques pouvant prendre des décisions où l'éthique est importante, nous voulons être capables de montrer que les décisions sont effectivement éthiques. D'autre part, si les recherches en méthodes formelles à propos de l'algorithmique, des langages et des systèmes ont bénéficié de nombreuses contributions, ce n'est pas encore le cas pour les données alors que celles-ci vont avoir un rôle important à jour dans les années à venir.

1. Plus d'informations par ici : http ://www.securite-routiere.gouv.fr/medias/espace-presse/publications-presse/bilan-definitif-de-l-accidentalite-routiere-2016 .
2. À noter que le nombre de véhicule autonomes en circulation comparé au nombre d'humain a aussi un impact certain.
3. Plus d'information par ici : http ://dx.doi.org/10.1073/pnas.1018033108 .