**Prouve-moi que ta méthode marche !**

*Ecrire un programme pour réaliser des tâches, c’est bien. Encore faudrait-il être sûr qu’on ne calcule pas n’importe quoi. Afin d’éviter des catastrophes monumentales, il est souvent nécessaire de s’assurer que les résultats d’un programme sont cohérents avec ce qu’on attend de lui. Si enlever les bugs « à la main » est en général long, laborieux et sans garantie, il existe des langages informatiques permettant de vérifier automatiquement si un programme est correct.*

Je suis à Paris et je prévois de prendre des vacances à Rome. Assis dans ma voiture, je lance mon GPS pour qu’il trouve le plus court chemin afin de rejoindre l’Italie. Celui-ci me donne alors une série d’instructions. Si par erreur, une seule des ces instructions est manquante, mais que je respecte tout de même les suivantes, je risque de terminer ma route très loin de ma destination initiale. Ce petit exemple illustre à quel point les algorithmes, qui sont les séries d’instructions que l’on donne aux ordinateurs, sont sensibles aux erreurs, même petites. C’est d’autant plus problématique que ces erreurs peuvent être très difficilement décelables par l’homme. Comment en effet remarquer qu’il manque l’instruction 452 « tournez à droite » dans un algorithme qui en comporte plus de 10 000 ? En particulier, si le programme a des répercussions sérieuses, comme c’est le cas des algorithmes qui régissent les décollages d’avion dans les tours de contrôle des aéroports, on ne peut pas se permettre de se tromper. C’est tout l’intérêt de la vérification automatique, qui fait faire ce travail à l’ordinateur.

En général, enlever les bugs manuellement consiste premièrement à constater une erreur, c’est-à-dire à remarquer que l’on obtient un résultat différent de celui attendu après l’exécution du programme, puis deuxièmement à la localiser, c’est-à-dire repérer l’instruction où le code est incorrect. Deux défauts majeurs rendent cette méthode pénible : d’une part, elle requiert que le programme tourne au moins une fois avec des valeurs numériques, pour vérifier qu’il est effectivement incorrect, et d’autre part, la localisation de l’erreur, si l’on n’a absolument aucune idée de son emplacement, impose de vérifier chaque ligne de code une par une jusqu’à ce que l’on ait trouvé la mauvaise.

A l’inverse, la vérification automatique propose des garanties de fonctionnement avant même que le programme ne tourne et sans vérifier les lignes une par une. Elle propose l’établissement : d’une part, d’un contrat signé par le programme du type « si vous me donnez tel type d’entrée, je m’engage à donner tel résultat ». Par exemple, dans le cas du GPS, le contrat serait « je m’engage à vous amener à Rome si vous me donnez une position initiale en France ». Et d’autre part de balises à différents endroits du code, qui vont tester des propositions spécifiques. Si chaque balise permet de prouver la suivante, le programme est prouvé. On pourrait par exemple créer une balise B1 « je suis à la frontière italienne » et une balise B2 « je suis à Rome ». Si la position initiale et les instructions jusqu’à B1 permettent de prouver B1, puis si B1 et les instructions jusqu’à B2 permettent de prouver B2, alors le programme du GPS est prouvé. En revanche, si par exemple B1 ne permet pas de prouver B2, mais plutôt B3 « je suis à Bruxelles », le programme est faux. Pas besoin de faire tout le chemin et constater seulement à l’arrivée que l’on n’est pas à Rome. En outre, on sait déjà que l’erreur est localisée entre B1 et B2.

Pourquoi alors ne pas simplement poser une balise à la fin du programme, qui testerait juste si le résultat est correct ? Simplement car il peut être très difficile de vérifier si une balise peut en prouver une autre. Cela peut même être impossible, auquel cas le programme est déclaré non prouvable avec ces balises particulières. En particulier, il peut être impossible de prouver que les données initiales permettent d’aboutir au résultat, mais tout à fait possible de prouver qu’elles conduisent à certaines balises intermédiaires, qui, elles, peuvent conduire au résultat. Le travail du vérificateur consiste donc à trouver les bonnes balises et à les mettre aux bons endroits.

Enfin, comme il existe déjà aujourd’hui des outils de vérifications automatiques (comme WHY3 ou COQ) utilisés par l’industrie, il devient intéressant de créer des vérifications automatiques de vérificateurs, c’est-à-dire des balises permettant de prouver qu’un vérificateur est effectivement capable de prouver un programme sans se tromper. On imagine bien à quel point cela peut être important, notamment pour être sûr qu’on ne prouve jamais (et donc qu’on n’utilise jamais) un programme incorrect. Et plus précisément, pour être sûr que notre GPS nous emmènera toujours là où on veut pour les vacances !