Sur quelques obstacles inattendus et caprices de Julia

Daniel Porumbel[1]

Conservatoire National des Arts et Métiers
daniel.porumbel@cnam.fr

Mots-clés : *langages de programmation en RO, Julia, C++*

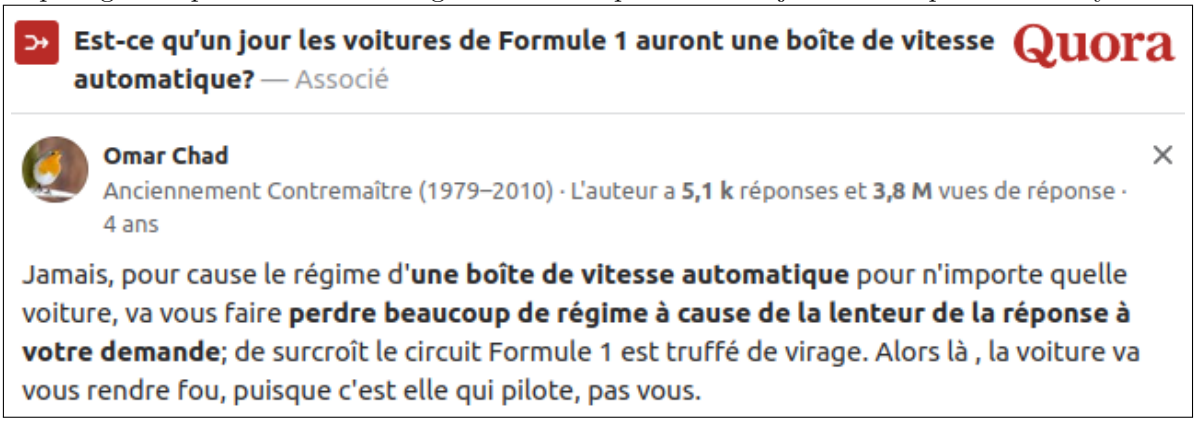
She looks so sweet as candy
But she can't fool me
I've seen her glancing at you
Working out a strategy

Run 4 Fun, Please Don't Talk to Julia

1 Introduction, synthèse et conclusion

Ma principale raison pour rejeter Julia, Matlab ou Python est la suivante : je veux savoir plus-ou-moins ce qui se passe lorsqu'un bloc de code est exécuté. Même le C++ peut me déplaire à cet égard, par exemple si on utilise trop d'héritage en cascade. Je me sens plus à l'aise avec un « C-like C++ ». Une telle préférence peut être défendue avec des arguments d'efficacité, mais c'est d'abord une question psychologique. S'il y a trop de couches intermédiaires dans la chaine de commande, je sens que je commence à perdre le contrôle de ma machine ; je ne la conduit plus tout seul. Je vous l'accorde : personne ne sait exactement ce qui se passe dans la tête d'une machine, car on ne peut pas la programmer en assembleur. Mais on doit lui limiter les caprices et les réactions inattendues, d'où un besoin d'ordre et de certitudes pour minimiser la confusion. Si vous avez peur de déclarer n comme variable globale pour ne pas plomber l'efficacité, vous avez peur des caprices de Julia. Les caprices d'efficacité ne seront jamais étonnants tant que travaillerez avec un mille-feuille logiciel. Ces milles couches logicielles seront toujours là, car ces langages très modernes feront le maximum pour rendre efficace même le code le plus superficiel, écrit à la légère sans trop connaître les structures utilisées derrière.

Il n'y a vraiment aucun problème si beaucoup sont conquis par ces nouveaux langages. C'est seulement la consommation massive et excessive qui risque de créer des désordres, en poussant le monde vers la légèreté. Par exemple, les profs de types ou de structures de données pourront bientôt avoir du mal à trouver des gens pour les écouter. Le principe « il faut savoir les utiliser mais c'est pas essentiel de savoir comment ça fonctionne à l'intérieur » prend de plus en plus de poids. Si on délègue trop de tâches à des machines, on risque de se retrouver un jour avec trop de gens dépersonnalisés – imaginez si ChatGpt codera un jour mieux que nous en Python.



Une autre raison d'antipathie vient de l'expérience personnelle. Imaginez un touriste qui visite trois fois une ville très appréciée par tout le monde et qui trouve à chaque fois plus d'obstacles que prévu. Est-il obligé de faire une étude très poussée de la ville avant de penser aller ailleurs ? L'industrie hôtelière devrait elle commencer à dresser une liste de vertus de la ville pour remettre le touriste perdu sur le bon chemin ? Si votre réponse à ces deux questions est « Oui », évitez cet exposé à tout prix. Je ne mets plus d'effort dans ce texte, car un lecteur pressé pourra penser que je m'acharne à faire un travail négatif. Je veux juste partager avec Monsieur Tout-le-Monde une autre vision sur la programmation en RO, présentée par un « lay man » – un homme plus ou moins capable de réaliser diverses tâches mais qui ne cherche pas à suivre les grandes politiques de programmation de telle ou telle école de pensée. Je n'ai pas écouté les discours des grandes sommités du domaine : j'ai dû apprendre sur le tas ; je n'ai pas eu le temps.