Benjamin Cathelineau

J’ai choisis les deux compétences suivantes :

3. Mettre en œuvre des processus de validation
4. Comprendre finement les implications des calculs faits par une machine

Mettre en œuvre des processus de validation

en mettant en place une infrastructure de test automatisée, extensible et réutilisable

C’est moi qui a développé la majorité des scripts de tests. Tout a été fait avec bash. J’ai essayé de faire ça de la manière la plus propre et maintenable possible en factorisant le code dans des fonctions. Certaines fonctions étant utilisées par plusieurs fichiers je les ais elle même mis dans des fichiers externes pour les récupérer ensuite avec la commande source. Il est bien connu que Bash n’est pas le language de script avec la meilleur synthaxe du monde donc ce ne fut pas trivial. Je suis arrivé à approximativement 300-400 lignes de shell script en tout.

en concevant une base de tests pertinente

La base de test a été principalement conçu par Lucie. J’ai participé au début à faire quelques fichiers .deca. Cependant, c’est moi qui a suggéré et implémenté la comparaison des résultats des tests avec les résultats attendus(que ce soit pour les numéros d’erreur pour les tests contexts, la sortie de la machine ima pour les tests codegens…)

en proposant une démarche d’assurance qualité (méthodes de validation utilisées, processus pour corriger des bogues, validations à effectuer avant une deadline)

J’ai proposé de vérifier si le dépôt était “propre” avant le rendu (c’est à dire pas de fichiers non nécessaires ainsi que pas de commentaires avec des TODO dans les fichier sources). Toutefois cela était une erreur car, comme expliqué dans le bilan de gestion d’équipe et de projet, retirer les TODOs des fichiers a modifié la sortie de référence et donc nos tests ne passaient plus. Nous avons réussi à régler ce problème moins de 10 minutes avant le rendu. J’ai appris de cette erreur et à l’avenir j’essayerais d’éviter les modifications de dernière minute

en gérant les contraintes de temps et de coût.

Nous avons failli manquer de temps pour finir l’extension (surtout la partie range reduction qui a été finalisée le 25 janvier) mais nous avons réussi à la finir à temps quand même.

Benjamin Cathelineau

Comprendre finement les implications des calculs faits par une machine

en maîtrisant les mécanismes d'un langage à objets (héritage, polymorphisme, liaison dynamique)

Je connaissais déjà ces mécanismes grâce au cours Java du premier semestre. Toutefois, deca, bien que également un language objet est assez différent de java sur certain points. Par exemple, il est impossible de faire des constructeurs avec des paramètres et donc d'initialiser les attributs d'un objet avec les valeurs que l'on veut. Il était donc nécessaire de créer une méthode supplémentaire pour assigner les attributs d'un objet. De plus il semble qu'il n'y ait pas de garbage collection en deca, les objets créés restent en mémoire pour toute l'execution du programme, donc cela nous a posé des problèmes de pile pleine (que nous avons réussi à régler).

en maîtrisant les transformations effectuées par un compilateur (traduction d'un langage de haut niveau vers un langage de bas niveau)

C'est Baudouin qui a fait la majorité de la partie C du compilateur. Pour l'implémentation en deca de l'extension trigo, nous avions besoin de calculer les puissances de 2, j'ai suggéré l'utilisation de shift afin d'optimiser l'execution. C'est Baudoin qui a créer la fonction permettant les puissances de 2 positives avec des shifts. Il a également fait l'implémentation de la fonction _fma qui a augmentée la précision de l'extension. Quand à moi, j'ai fait une seule fonction en assembleur : celle qui génère une erreure d'exécution si la valeur entrée dans asin est au dessus de 1 ou en dessous de -1.

en prenant en compte les limitations induites par la représentation et les calculs avec des flottants.

C'est moi et nathan qui avons implémenté l'extension TRIGO en deca. Nous avons tout de suite eu des problèmes de précision car notre algorithme de range reduction n'était pas de très bonne qualité. En effet nous utilisions le nombre pi un très grand nombre de fois (l'algorithme était juste soustraction répété de pi jusqu'à que nous arrivions dans l'interval $[-\pi, \pi]$). Cela était à la fois très lent et très imprécis. La valeur de pi stockée dans un flottant sur 32 bits n'étant pas la valeur exacte de pi, nous accumulions une erreur de plus en plus grande à chaque soustraction. Toutefois sur les conseils de Lucie et de Baudoin nous avons trouvé un autre algorithme, qui n'utilise pi qu'une ou deux fois grâce à une division et un fma, bien plus efficace (beaucoup plus rapide et plus précis).