Rapport de progrès - Semaine 3

Philippe Caron

15 février 2017

1 Résumé de l'avancement

Au cours de la première semaine le projet a avancé à grands pas. Malheureusement, l'ordinateur sur lequel le projet était développé a connu quelques problèmes logiciels, la résolution de ces problèmes a occupé la semaine suivante. Malgré tout, la progrès de la première semaine étant suffisemment important, le projet demeure sur un échéancier raisonnable. Jusqu'a maintenant, le compilateur peut compter sur un préprocesseur fonctionnel, et un compilateur intermédiaire presque terminé. Il est possible de produire une représentation intermédiaire pour n'importe quel code lua ne comprenant ni fonction ni boucle.

2 Objectifs immédiats (2 semaines)

La progression des prochaines semaines risque d'être ralentie par la période d'examen. Néanmoins, l'objectif principal est de terminer le compilateur intermédiaire et être bien avancé dans le compilateur. Pour ce faire, le compilateur intermédiaire devra pouvoir gérer les boucles et les fonctions, ainsi que pouvoir détecter les fermetures. Quant au compilateur principal, son implémentation devrait être relativement simple une fois le langage IR bien établi. Le principal défi sera l'allocation des registre selon la norme utilisée par gcc pour l'appel de fonctions.

3 Description des composantes terminées

3.1 Préprocesseur

Un langage utilisant la notation infixe tel que lua nécessite d'emblée un préprocesseur. En effet, celui-ci s'assurera que le parenthésage de chaque expression est complet, de sorte qu'il sera facile pour le compilateur de gérer la priorité des opérations. De plus, on veut retirer le sucre syntaxique ainsi que les commentaires afin que le compilateur soit le plus simple possible.

3.1.1 Priorié des opérations

La priorité des opération en lua est très simple et ne comprend que huit niveau différents, tous sont associatif de la gauche à moins qu'il en soit spécifié autrement :

- Niveau 1 - Puissance : $\hat{}$ (associativité à droite)
- Niveau 2 - Unaire : - not
- Niveau 3 - Multiplication : * /
- Niveau 4 - Addition : + -
- Niveau 5 - Concatenation : .. (associativité à droite)
- Niveau 6 - Comparaison : == ~= < > <= >=
- Niveau 7 - Conjonction : and
- Niveau 8 - Disjonction : or

Le préprocesseur parcour chaque déclaration en ordre de niveau dans le sens inverse de l'associativité, et regroupe les termes requis par des parenthèse.

3.1.2 Mise en forme

Pour faciliter le travail du compilateur, le code est mis en forme de manière standard. Les commentaires sont retirés tout comme le sucre syntaxique. Par exemple «`str:sub(i,i)`» devient «`str.sub(str, i, i)`» qui devient à son tour «`str ["sub"] (str , i , i)`». Le résultat est donc très facile à lire pour le compilateur.

3.2 Compilateur intermédiaire

Le compilateur intermédiaire n'est pas fini, mais il peut déjà produire un peu de code IR. Tout ce qui est expression simple (sans boucle ou fonction) peut être générer, quoique le tout n'aie pas encore été testé. Le but est que le code produit comprenne un jeu d'instruction similaire à du code assembleur, avec une structure très près de celle du code machine.

3.2.1 Gestion de l'affectation multiple

Ceci est le seul cas où le langage IR ne représentera pas le code assembleur produit. Lors d'affectation multiples (ex : «`local x, y, z = 1, 2`») le compilateur utilisera une syntaxe spéciale de sorte à s'assurer que la pile est de la bonne taille une fois l'exécution de l'affectation terminée.

4 Problèmes et limites

Pour l'instant les chaînes de caractères multi-lignes délimitées par «[[...]]» ne sont pas prises en charge par le compilateur, ce n'est d'ailleurs pas une priorité car l'ajout de cette fonctionnalité n'a pas d'influence sur le déboggage du compilateur. De plus, son ajout est relativement rapide, à condition que tout fonctionne, ce qui est la priorité. Dans la même optique, les caractères spéciaux indiqués par un *backslash* (\) ne sont pas pris en charge non plus.

Étant donné que tout le code produit jusqu'à présent est une première expérience de programmation en Lua, beaucoup de fonction sont programmées de manière lourde et inefficace. Une des principales choses à faire une fois le compilateur terminé sera de réécrire ces fonctions. Également, les programmes ne sont pas encore liés entre-eux alors certaines fonctions sont présentes en double, ce qui devra être corrigé.

5 Conclusion

En bref, le projet avance bien, mais les problèmes techniques ayant ralenti le progrès il faudra s'assurer d'avoir un bon rythme au cours de prochaines semaines.