PROJ2 : Rendu 2

Par François Pitois et Simon Fernandez

Description

Ce projet est un interprete, compilateur, machine à pile du langage Fouine.

Contenu

- main.ml : fichier d'entrée. Fait l'appel aux differents modules : Parsing, Lexing, Interprete, etc... Traite les options données pour appeler ce qu'il faut
- lexer.mll : le lexer
- parser.mly : le parser
- types.ml : contient les differents types dans lesquels seront stockées les instructions lues par lexer/parser
- interprete.ml : interprete de l'arborescence générée par lexer/parser, stockée dans les types décrits par types.ml
- interprete_aux.ml : fonctions utilitaires pour interprete. Sont juste utiles et rendent le code plus lisible.
- compiler.ml : compile l'arborescence en une suite de lexems qui seront en suite traités par la machine à pile
- machine.ml : machine à pile, prend une suite de lexems créés par compiler.ml, et les évalue, renvoit le haut de la pile apres l'execution
- machine_utils.ml : Un tas de fonctions utilitaires qui sont utilisées par la machine classique et par la ZINC
- machine_types.ml : Contient les lexems de la machine et ceux des différentes stacks
- zinc_compiler.ml : le compilateur de la version ZINC de la machine
- zinc_machine.ml : la machine de la version ZINC
- de_bruijn.ml : applique la transformation de De bruijn
- convert.ml : le convertisseur de Caml + extenstion vers Caml
- exception.fou.ml : Fichier fouine qui décrit les transformations de Caml + exception vers Caml
- Makefile : le Makefile du projet
- README.md : ce document
- exemples.fou.ml : Code fouine contenant de nombreux exemples sur les differentes possibilités de l'interprete et de la machine

Utilisation

Pour compiler les sources, le Makefile se charge de tout. Il faut juste taper make et tout est créé. L'executable s'appelle fouine

Le resultat de l'execution est soit un entier, soit une fonction. Dans ce dernier cas, l'interprete affichera le code CamL correspondant, et la machine à pile affichera la fonction sous la forme d'une suite de lexems correspondants à la version compilée de la fonction

Options

- -d ou –debug : Rend l'execution verbeuse. Seront affichés differentes informations intermediaires sur ce qu'il se passe lors de l'execution.
- -m ou –machine : Compile le programme donné et l'execute avec la machine à pile
- -z ou –zinc : Compile le programme et le lance sur la machine ZINC
- -i ou –interm toto.code : Compile le programme et stocke la suite de lexems dans toto.code sans essayer de l'executer
- -a ou –all : Lance l'interpreteur ET la machine à pile ET la ZINC sur la même entrée, pour comparer les sorties
- -E : Pour lancer la traduction vers fouine sans exception à partir de Fouine avec exception. (MAIS ne marche pas avec les fonction récursives)

Parties non traitées :

- Recursivité sur des variables dans la machine à pile
- Le moins unaire
- L'option -R et -RE

Options supplémentaires

- En plus du prInt qui affiche la valeur de l'int, nous avons ajouté feedback, qui affiche toute la valeur demandée (donc même si c'est une fonction ou une reference), sous le même format que ce qui est renvoyé pour le resultat
- Une option -debug pour encore plus de détail. Bien plus verbeux que -verbose

Options et langage de la Machine à pile

Possibilités

- Arithmétique classique
- If then else
- let ... in
- prInt et Feedback
- Fonctions
- References d'entiers et de fonctions
- Fonctions recursives
- Exceptions
- Indices de De Bruijn : OK sur la classique et sur ZINC

Reste à faire

- Tableaux
- Comparaison d'efficacité entre la classique et ZINC

Lexems

- CONST_INT(n) : Represente une constante entière.
- ACCESS(n) : Va chercher le n-ème terme en memoire et le met sur le haut de la pile
- GRAB : Prend le haut de la pile et le met sur le haut de l'environnement
- CLOS(code) : Representation d'une fonction : la suite de lexems à egffectuer pour appliquer la fonction. La variable est la première sur l'environnement
- OP(c) : Une opération arithmétique ou booléenne, sous la forme d'un seul caractere. Sont acceptés : +, -, *, /, =, & (le "et" logique), | (le "ou" logique), <, >
- APPLY : Appel la fonction qui est sur le dessus de la pile sur l'argiment qui est en 2e position sur la pile.
- PRINT : Affiche la valeur entière du dessus de la pile, sans la dépiler
- FEEDBACK : Affiche la valeur du dessus de la pile, sans la dépiler

- ENDLET : Supprime la valeur qui est sur le haut de l'environnement
- IF : Dépile le haut de la pile, si c'est true, execute jusqu'au ELSE et ensuite saute tout jusqu'au ENDIF. Si c'est false, on execute uniquement la portion entre ELSE et ENDIF
- RETURN : Pour revenir d'un appel de fonction. Va chercher sur la pile le code et l'environnement qui étaient en attente lors du APPLY pour continuer l'execution.
- REF : Comme GRAB mais stocke la valeur sous la forme d'une reference
- DEREF : Bang sur la référence qui est sur le haut de la pile
- ASSIGN(n) : Assigne la valeur qui est sur le haut de la pile à la reference x qui est déjà dans l'environnement.
- CLOS_R(code) : Comme CLOS, mais pour des fontions recursives. Si lors du APPLY c'est une fonction recursive qu'on se retrouve à appeler, on la push dans son propre environnement avant de l'executer. Pout les indices de De Bruijn, la variable est en position 2 sur la pile et la fonction en position 1
- SEQ : Pour délimiter des séquences : dépile le haut de la stack.
- TRY : Dépile la fonction sur la stack et l'ajoute sur la pile d'erreurs
- RAISE : Dépile la fonction sur la pile d'erreur, effectue son calcul et reprend le code là ou il faut
- ENDTRY : Dépile une fonction de la pile d'erreurs. Arrive quand rien n'a été RAISE

Pour ZINC - TAILAPPLY : Appel de apply avec recursivité terminale - PUSHMARK : Met une marque sur la pile pour délimiter les arguments passés à une fonction sur la pile d'arguments

Repartition du travail

- main : Simon
- lexer/parser : François
- types : François et Simon
- interprete : François
- compiler : Simon
- machine : Simon
- zinc : Simon
- De Bruijn : Simon
- convertisseur : François
- Exemples : François

- Readme : Simon