Langages de programmation et compilation

Un Compilateur Petit Scala

Auteurs :
Nicolas ASSOUAD

15 janvier 2016

1 Rapport du premier rendu du projet de Compilation

Le travail demandé lors de cette première partie du projet était d'implémenter un analyseur syntaxique et un typeur pour un sous langage du langage Scala, appelé Petit Scala.

Ces différentes fonctionnalités ont toutes été implémentées. Notre compilateur pscala passe notamment tout les tests fournis sur le site internet du cours : https ://www.lri.fr/ filliatr/ens/compil/ (ceux de la partie 1 et 2).

L'usage de la commande pscala suit les spécifications du sujet.

Pour l'analyseur lexical ou syntaxique, peu de choix personnels ont été réalisés, cela a principalement consisté à écrire la grammaire et les différentes règles de priorité.

Les différents types sont définis dans le fichier scala_spec.ml. Chaque élément de la syntaxe abstraite possède un type particulier, comme les classes (sc_class), les méthodes (sc_method), les variables (sc_var), les expressions (sc_expr),... Au départ, le parti pris était d'associer à chaque élément de la grammaire formelle un type de la syntaxe abstraite, mais devant la redondance que cela avait entrainé, une fusion de certains types comme par exemple l'utilisation d'un même type : sc_param_ty pour les paramètres de types des classes et des méthodes a beaucoup simplifié le code. La dernière chose à noter sur l'implémentation des types est que certains sont décorés de positions comme sc_expr_dec, cela permet au typeur de rapporter la plupart de ses messages d'erreur avec une position précise. Il n'y a pas eu pour l'instant de décoration plus aprofondie de l'arbre de syntaxe abstraite, cela est volontaire, le rajout des types de chaque expression dans l'arbre étant facile, ils seront rajoutés quand la manière d'implémenter la suite du compilateur sera mieux comprise.

L'implémentation du typeur suit l'algorithme décrit dans le sujet du projet. Le contexte d'exécution est représenté dans mon programme par un triplet :

- un dictionnaire (MAP.MAKE) qui associe des chaînes de caractères (identifiant de classe) à un élément de type sc_class (une classe dans la syntaxe abstraite)
- un dictionnaire (MAP.MAKE) qui associe des chaînes de caractères (identifiant de classe) à un couple d'une contrainte binaire (bin_constraint) et d'un type (type_system)
- une liste de variables de type sc_var (une variable dans la syntaxe abstraite). Ces variables sont ordonnées dans l'ordre de déclaration, si x se trouve avant

y dans cette liste, cela signifie que x a été déclaré après y

La liste de variables de l'environnement possède une variable spéciale dont l'identifiant est la chaîne vide (var_id = ""), il peut être présent en plusieurs exemplaires dans la liste. Il sert de délimitateur entre les blocs, pour séparer les "espaces de nommage" de chaque bloc.

A la fin du typage d'une classe, ses champs class_members et class_methods contiennent respectivement les différentes variables membres et les différentes méthodes de la dite classe, y compris celles qui sont héritées. Le champs class_members possède les paramètres du contructeur de la classe typée, mais pas ceux de ses parents.

Deux exceptions peuvent être lancées par le typeur :

— Typing_Error(str), la plus courante, elle indique que le fichier en cours de typage est mal typé.
— Typing_Exception(str), rare, indique que le typeur n'a pas pu effectuer certaines actions à cause d'un disfonctionnement. Ne devrait jamais être levé !

2 Rapport du second rendu du projet de Compilation

Le compilateur pscala fonctionne correctement sur tous les tests fournis dans le sujet. Il répond à toutes les spécifications demandées. On a ajouté une option supplémentaire au compilateur : "-v". Elle permet de lui demander de ne pas être complètement silencieux et de dire les étapes qu'il a réalisées (de manière très succincte).

A présent, le typeur fait plus que typer. Il décore l'arbre de syntaxe abstrait. Il rajoute entre autre les types de chaque expression typée dans l'arbre. Quelques ajouts ont été fait pour rendre la compilation plus aisée :

— Le typeur ordonne les méthodes des classes de manière préfixe. Les méthodes héritées se trouvent en première dans la liste des méthodes à la fin du typage. Il y a notamment une cohérence entre l'ordre des méthodes d'une classe et d'une autre classe qui en hérite.
— Le typeur ordonne les attributs. Ils suit d'abord un ordre préfixe, il y a d'abord les attributs de la classe mère, puis ceux de la classe courante. En parallèle de cela, les attributs sont rangés ainsi : les paramètres du constructeur d'abord, puis les véritables attributs. On a par exemple : param_cons_mere, attrib_mere1, attrib_mere2, param_cons_fille1, param_cons_fille2, attrib_fille1, attrib_fille2, attrib_fille3.

Tout cela facilite grandement le travaille de compilation étant donné que tout est bien ordonné.

Il n'y a pas grand chose à dire de spécifique sur la compilation en elle même. A noter la présence de compteurs en début de fichier qui permettent de gérer les étiquettes pour les sauts dans les fonctionnalités IF, OR, AND, IF, ou WHILE. Ils permettent de créer des étiquettes différentes tout au long de la compilation.

Notre manière de repérer les positions des variables locales change par rapport à celle proposée en TD. Au lieu de remplacer tout les noms de variables au début par des décalages, on le fait à la volée en gardant en mémoire une table des variables locales et de leur décalage. Cela implique que au début d'un bloc, il n'y a pas allocation par soustraction à rsp d'une plage pour tout les variables locales de celui-ci. Les variables sont calculés puis mis sur la pile individuellement au fur et à mesure qu'on les rencontre.

Cette manière de faire ce retrouve aussi dans notre manière de gérer les tableaux d'activation. Ils sont eux aussi non pas alloué en une seule fois par une soustraction à rsp, mais mis sur la pile au fur et à mesure.

Que cela soit pour l'allocation des variables d'un bloc ou l'allocation d'un tableau d'activation, une fois que les variables ne sont plus utiles (à la fin du bloc, ou à la fin de la fonction), le compilateur se charge de dépiler minutieusement toutes ces variables ainsi allouées.

Le compilateur gére correctement l'héritage, appelle bien le constructeur de la super classe. Les paramètres du constructeur d'une classe sont privées à celle-ci, ils ne peuvent être appelé directement par une sous classe.