Traduction des langages

Projet

HPCBD - B2

Issam Habibi - Badr Sajid

2021

Contents

1 Introduction

Le compilateur du langage RAT a été construit dans une version basique lors des séances des TPs où on a travaillé sur les quatre passes fondamentaux : La gestion d'identifiants, le typage, le placement mémoire et la génération du code. Cependant, cette version ne traite pas les cas de plusieurs outils qu'un programmeur va sans doute vouloir intégrer dans son code : les pointeurs, la surcharge des fonctions, les types énumérés et la structure de contrôle switch/case. Ce projet a comme but donc d'étendre le compilateur du langage RAT déja réalisé afin de prendre en compte toutes les nouvelles constructions qu'on a déja cité.

2 Types et AST

L'ajout des nouvelles construction nécessite une évolution de la structure de notre AST :

2.1 Pointeurs

- L'ajout d'un nouveau cas de type : **Pointeur of type**
- L'ajout du nouveau type: **affectable = Ident of string | Valeur of affectable**
- Modifier un cas d'instruction : **Affectation of affectable * expression**
- L'ajout des cas aux expressions : **Affectable of affectable | Null | New of typ | Adresse of string**

2.2 Surcharge

Pas de modification à apporter à la structure de L'AST.

2.3 Enumeration

- L'ajout d'un nouveau cas de type : **Enumere of string**
- L'ajout du nouveau type enum : **Enumeration of string * string list**
- L'ajout d'un cas aux expressions représentant la valeur du type enumere : **ValEnum of string**.
- changer la structure d'un programme qui doit prendre en compte aussi les énumérations définies: **type programme = Programme of enum list * fonction list * bloc**.

2.4 Switch/Case

- L'ajout du nouveau type typecase qui couvre les valeurs possible des constantes avec lesquelles on va comparer successivement la valeur de l'expression du switch :
  typecase = TEnum of string | TEntier of int | TTrue | TFalse
- l'ajout d'un cas aux bloc qui est les case/default :
  case = Case of typecase * instruction list * bool | Default of instruction list * bool
- L'ajout d'un cas aux instructions pour prendre en compte les Switch/Case :
  Switch of expression * case list

3 Pointeurs

Nous avons tout d'abord ajouté des nouveaux tokens au lexer pour prendre en compte les modifications apportées sur la grammaire : notre compilateur doit reconnaître les terminaux : **new**, **&** et **null**. Ensuite, nous avons ajouté au parser les nouveaux règles de productions permettant la définition et la manipulation des pointeurs.

Pour la gestion des identifiants, il fallait évidemment tout d'abord ajouter les changements déja exhibés dans la partie AST à l'AstTds en changeant les string par des Tds.infoast .Nous avons ensuite ajouté une méthode pour analyser les affectables en prenant en compte les deux cas possibles : L'affectable est au gauche ou droite de l'affectation. Il fallait ensuite traiter les cas du type Affectable ajouté à l'AST dans les méthodes d'analyse des expressions et des instructions.

Pour le typage, et autre que la modification de l'AstType, il n'y a pas de traitement particulier pour les pointeurs : Il faut prendre en compte les types déja ajoutés dans les méthodes d'analyse de type des instructions et des expressions et ajouter une méthode pour analyser le type affectable .

Pour le placement mémoire, il fallait définir la taille d'un pointeur comme 1.

Finalement, pour la génération du code, nous avons défini 2 méthodes d'analyse du code afféctable selon son positionnement (à gauche ou à droite). Ensuite, nous avons traîté les cas des nouvelles expressiosns et instructions dans les méthodes d'analyse associées afin de générer le même code précisé dans le TD : Empiler les adresses avec LOADA d[r], réaliser une allocation de mémoire avec SUBR MAlloc et empiler/écrire les mots à partir de leurs adresses au sommet de la pile avec LOADI(n) et STOREI (n)

4 Surcharge

Pour la gestion des identifiants, nous avons tout d'abord changé l'approche avec laquelle en analysait les fonctons: si on trouve maintenant une fonction qui porte le même nom qu'une autre présente dans la TDS, nous vérifions si les deux fonctions n'ont pas les mêmes listes des paramètres et nous ajoutons la nouvelle fonction à la TDS le cas échéant. Sinon, nous levons une exception de double déclaration. Dans l'analyse des expressions, un appel de fonction nécessite maintenant la prise en compte de toutes les autres fonctions qui portent potentiellement le même nom. On a décidé donc d'apporter un changement à l'AstTds : le cas AppelFonction du type expression prend maintenant une liste des info_ast au lieu d'une seule, cette liste contient l'ensemble des fonctions avec un nom commun. La recherche globale dans la tds dans le cas d'un AppelFonction renvoie aussi une liste de fonctions aulieu d'une seule grâce à une nouvelle méthode de recherche définie.

Pour le typage, nous avons pris en compte la différence qui existe entre deux fonctions qui portent le même nom qui est la liste des paramètres. Nous avons défini donc une méthode **choixfunction** qui , étant donné le nom d'une fonction, la liste de ses paramètres et la liste des fonction portant le même nom, choisit celle contenant la liste des paramètres compatibles. Cette fonction nous permet dans l'analyse du type des expressions d'agir sur la liste des fonctions renvoyée lors du passeTDS pour récupérer et renvoyer la fonction exacte avec la bonne liste des paramètres.

Les autres passes demeurent inchangées.

5 Enumeration

Comme pour les pointeurs, nous avons ajouters de nouveux tokens aux lexer pour pouvoir reconnaîte les nouveaux terminaux : **enum** et **","** séparant les composants de l'énumération. Ensuite, nous avons ajouté au parser les nouvelles règles de productions permettant la définition et la manipulation des énumérations.

Pour la gestion des identifiants, nous avons tout d'abord ajouté un nouveau type d'information associé aux identifiants qui définissent une énumération : **InfoEnum of string * typ**, qui prend en compte le nom de cette énumération et le type de ses éléments. Ensuite, nous avons défini une méthode pour anaylser les énumérations dans la tds, traîter les nouveaux cas d'InfoEnum et ValEnum dans les méthodes d'anaylse déja définies et changer la signature du méthode **analyser** vu le changement de la structure d'un programme, qui devient maintenant (comme déja cité dans la partie AST) : un ensemble d'énumérations, un ensemble de fonctions et un bloc.

Pour le typage, nous avons fait un travail similaire en définissant une nouvelle fonction d'analyse des types énumérations, en résolvant la surchagre de l'opérateur = qui doit pouvoir comparer des énumérations aussi maintenant avec la définition d'un cas **EquEnum** dans le type binaire et le traitement du cas correspondant dans l'analyse des expressions.

Pur le placement mémoire, il fallait définir la taille d'une énumération qu'on a considéré 1 .

Pour la génération du code, pas de changement majeurs sauf la prise en compte le taille d'une énumération dans l'analyse du code bloc et l'ajout de l'évaluation d'une expression de type **ValEnum** à la tête de la pile grâce à la méthode d'analyse des expressions.

6 Switch/Case

Nous avons tout d'abord ajouter des nouveaux terminaux au lexer : **switch**, **case**, **default** et **break**. Ensuite, nous avons défini les nouvelles règles de production de la structure Switch/Case dans le parser.

Pour la gestion des identifiants, le travail fait s'intéressait globalement à l'analyse des instructions où on devrait prende en compte le cas d'une structure Switch/Case : il faut analyser l'expression passé en paramètre du Switch. Ensuite, il faut analyser la liste des cas traîtés (case) en analysant les instructions produite dans chaque case et pareil pour les instructions produite dans le cas Default.

Pour le typage, le travail était similaire : nous analysons pour la structure Switch/Case l'expression passée en paramètre du Switch. Ensuite, pour chaque cas (case) appartenant à la liste des cas il faut vérifier la conformité du type de la constante passé en paramètre avec le reste des constantes des autres cas et pareil avec le Default, sans oublier d'analyser le typage dans les bloc d'instructions introduite après chaque cas.

Nous n'avons pas apporté de changements au placement mémoire ou à la génération du code.

7 Conclusion

Ce projet était intéressant dans le sens où il permettait de découvrir le fonctionnement précis d'un compilateur et sa construction. Nous n'avons jamais imaginé, moi et mon binôme, qu'on allait un jour creusé autant dans les grammaires et les différentes passes pour construire, même dans une version basique, un compilateur pour un nouveau langage tel que RAT.

Nous avons rencontré beaucoup de difficultés le long du projet, la première était la familiarisation avec sa structure et les différentes définitions d'interfaces, de modules, de types..etc dans les différents fichiers. Mais ce n'était qu'une question de temps avant de devenir plus alaise avec cette structure. Les méthodes d'analyse n'étaient pas simple, surtout quand elles sont non (ou très peu) guidées et il fallait donc essayer de toujours s'inspirer de ce qui a déja était fait pour réaliser et adapter ces méthodes pour les nouveaux cas.

La surcharge a posé des problèmes de conceptualisation, la prise en compte d'une définition d'une nouvelle fonction avec le même nom mais des paramètres différents ouvre la porte pour plusieurs possibilités, nous jugeons que notre manque d'expérience nous ramène toujours à résoudre un tel problème avec des structures de données (comme les listes) même s'ils sont lourdes. Nous somme persuadés qu'il existe des méthode plus astucieuses qui ne nécessitent pas forcément de changer la structure de l'AstTds pour charger la liste de tout les fonctions portant le même nom à chaque fois où on appel une fonction.

Pour les types énumérés, il fallait pas oublier qu'on doit pouvoir définir une énumération à l'intérieur ou l'extérieur du programme principale et l'interdiction de la duplication d'une valeur de l'énumération au sein de l'énumération même. Nous pensons qu'il y a plus de travail à faire dans les passes placement de mémoire et génération de code et cette remarque est valable aussi pour la structure de contrôle Switch/Case.

Au final, nous avons beaucoup appris sur la structuration et la construction des compilateurs et leurs fonctionnements. Notre perception des compilateurs s'étend maintenant à plus qu'une petite ligne de commande qu'on passe à notre terminal pour compiler sans savoir ce qui passe derrière les coulisses de ce compilateur.