Rapport de Projet TdL et PF

Dino Gurnari
Valentin Lebrun

Département Sciences du Numérique
Deuxième année - Architecture, Systèmes et Réseaux
2021-2022

Table des matières

1 Introduction

Le but du projet de programmation fonctionnelle et de traduction des langages est d'étendre le compilateur du langage *RAT* réalisé en TP de traduction des langages pour traiter de nouvelles constructions : les pointeurs, l'opérateur d'assignation d'addition (+=), les enregistrements et les types nommés.

Pour cela nous devront modifier le lexer et le parser pour prendre en compte le nouveau lexique et la nouvelle syntaxe puis modifier les 4 différentes passes (Gestion des identifiants, Typage, Placement mémoire et Génération de code). Et enfin nous devront tester notre compilateur pour qu'il réponde à toutes les exigences.

2 Types

2.1 Modification de l'AST

Nous avons rajouté dans l'AST le type `affectable` qui permet d'accès aux identifiants enregistrés en passant par un *ident*. On utilise ce type également pour les déréférencements des pointeurs ou pour accéder à un champs d'un enregistrement. Le type `instruction` de l'Ast Syntax peut également servir pour l'assignation plus ou la déclaration de type. Le type `expression` est utilisé pour créer un enregistrement qui prend comme paramètre la liste des expressions de chaque champs de l'enregistrement, mais également pour définir un pointeur avec l'opération *New* qui prend un type ou `Null`. On utilise *Adr* afin d'accéder à l'adresse d'un variable.

2.2 Jugements de typage

Les nouveaux jugements de typage sont les suivants :

— n° 5 : $\frac{\sigma\vdash TYPE:\tau \quad \sigma\vdash Tid:\tau}{\sigma\vdash TD\rightarrow typedef Tid=TYPE:void,[]}$

— n° 12 : $\frac{\sigma\vdash A:\tau \quad \sigma\vdash E:\tau}{\sigma\vdash I\rightarrow A=E:void,[]}$

— n° 13 : $\frac{\sigma\vdash TYPE:Rat|Int \quad \sigma\vdash A:\tau \quad \sigma\vdash E:\tau}{\sigma\vdash I\rightarrow A+=E:void,[]}$

— n° 19 : $\frac{\sigma\vdash TYPE:\tau \quad \sigma\vdash tid:\tau}{\sigma\vdash I\rightarrow typedef tid=TYPE:void,[]}$

— n° 20 : $\frac{\sigma\vdash A:\tau \quad \sigma\vdash id:\tau}{\sigma\vdash I\rightarrow A=id:void,[]}$

— n° 21 : $\frac{\sigma\vdash A1:\tau \quad \sigma\vdash A2:ptr(\tau)}{\sigma\vdash I\rightarrow A1=(*A2):void,[]}$

— n° 22 : $\frac{\sigma\vdash A1:\tau \quad \sigma\vdash A2:Record \quad \sigma\vdash id:\tau}{\sigma\vdash I\rightarrow A1=(A2.id):void,[]}$

— n° 28 : $\frac{\sigma\vdash TYPE:\tau}{\sigma\vdash TYPE\rightarrow TYPE*:ptr(\tau),[]}$

— n° 29 : $\frac{\sigma\vdash Tid:}{\sigma\vdash TYPE\rightarrow Tid:\tau,[]}$

— n° 30 : $\frac{\sigma\vdash DP:\tau}{\sigma\vdash TYPE\rightarrow struct\{DP\}:Record(\tau),[]}$

— n° 45 : $\frac{\sigma\vdash A:\tau}{\sigma\vdash E\rightarrow A:\tau,[]}$

— n° 46 : $\frac{}{\sigma\vdash E\rightarrow null:ptr(null),[]}$

— n° 47 : $\frac{\sigma\vdash TYPE:\tau}{\sigma\vdash E\rightarrow (New\,TYPE):Ptr(\tau),[]}$

— n° 48 : $\frac{\sigma \vdash id:\tau}{\sigma \vdash E \to id:ptr(\tau),[]}$

— n° 49 : $\frac{\sigma \vdash CP:\tau}{\sigma \vdash E \to CP:Record(\tau),[]}$

3 Pointeurs

Pour définir les pointeurs nous avons dû repenser à la façon d'accéder au variable. Pour cela nous avons créé le type `affectable`. Dans la passe Tds nous vérifions que les identifiants ne sont pas créé en double si on cherche à la définir et qu'un accès se fait sur une variable déjà crée pour cela on utilise un paramètre "modifie" afin de savoir si on essaye de la créer ou de l'utiliser. Cette technique est aussi utilisé dans la génération de code *TAM* afin de savoir si on store ou si on doit "load" la variable. Pour la compatibilité des types on utilise la fonction `est_compatible` dans laquelle on rajoute les cas valables.

4 Opérateur d'assignation d'addition

Pour implanter l'opérateur d'assignation `+=`, nous avons rajouter un token correspondant lui correspondant ainsi que sa règle de grammaire dans le parser puis dans la passe TDS nous remplaçons l'opération d'assignation d'addition (`x =+ y`) par une addition et d'une assignation (`x = x + y`)

5 Types nommés

Le principe de notre implantation des types nommés est de remplacer dans la passe de typage les types nommés par leurs types réels équivalents. Pour cela il faut avoir préalablement vérifier la bonne définition et utilisation de ces types dans la passe de gestion des identifiants.

6 Enregistrements

Pour enregistrement, nous avons d'abord gérer comment créer un record, puis ensuite nous avons gérer comment accéder aux champs créer. Nous avons créer un nouveau type

7 Bonus

Extension VSCode

Nous avons réalisé en parallèle du projet une petite extension de coloration lexicale pour le langage RAT nommée RAT Highlights, elle est réalisée en suivant la grammaire `TextMate` en `JSON`. Les opérateurs, mots-clefs et fonctions de bases sont prisent en compte ainsi que les types et les commentaires.

8 Conclusion

Pour conclure ce rapport nous pouvons parler de ce que ce projet nous à apporter. Tout d'abord nous avons appris à mieux comprendre le fonctionnement d'un compilateur et les difficultés liées à sa conception, notamment pour l'implantation de fonctionnalités plus complexes telles que les pointeurs et les enregistrements. Cela nous a aussi permis de mettre en oeuvre une bonne stratégie de travail en binôme.

Pour aller plus loin dans la conception de ce compilateur nous aurions pu mettre en place les listes chaînées qui demande de pouvoir faire des définitions récursives d'enregistrements. La libérations de mémoire à l'aide d'une méthode `free` ou l'opérateur d'incrémentation (`++`) sont d'autres pistes d'amélioration.