Projet de programmation fonctionnelle et de traduction des langages

Année 2021/2022

HUC Alexia

GUEMIL Walid

Table des matières

I. Introduction

Le projet est l'implantation d'un compilateur du langage RAT vu en TP avec l'ajout d'autres constructions telles que les pointeurs, l'opérateur d'assignation, les types nommés et les enregistrements. Ce rapport détaille les modifications apportées par ces nouveaux traitements au compilateur RAT, notamment au niveau de l'AST et des différentes passes.

II. Pointeurs

II.1. Lexer et parser

Pour ajouter le traitement des pointeurs, nous avons tout d'abord modifié notre lexer et parser afin d'ajouter les non teminaux (**new**, * et **&**) et les règles de grammaires nécessaire au traitement :

- $A \rightarrow (* \ A)$: déréférencement : accès en lecture ou écriture à la valeur pointée par A ;
- $TYPE \rightarrow \ TYPE *$: type des pointeurs sur un type TYPE ;
- $E \rightarrow null$: pointeur null ;
- $E \rightarrow (new \ TYPE)$: initialisation d'un pointeur de type TYPE ;
- $E \rightarrow \& \ id$: accès à l'adresse d'une variable.

II.2. AST Syntax

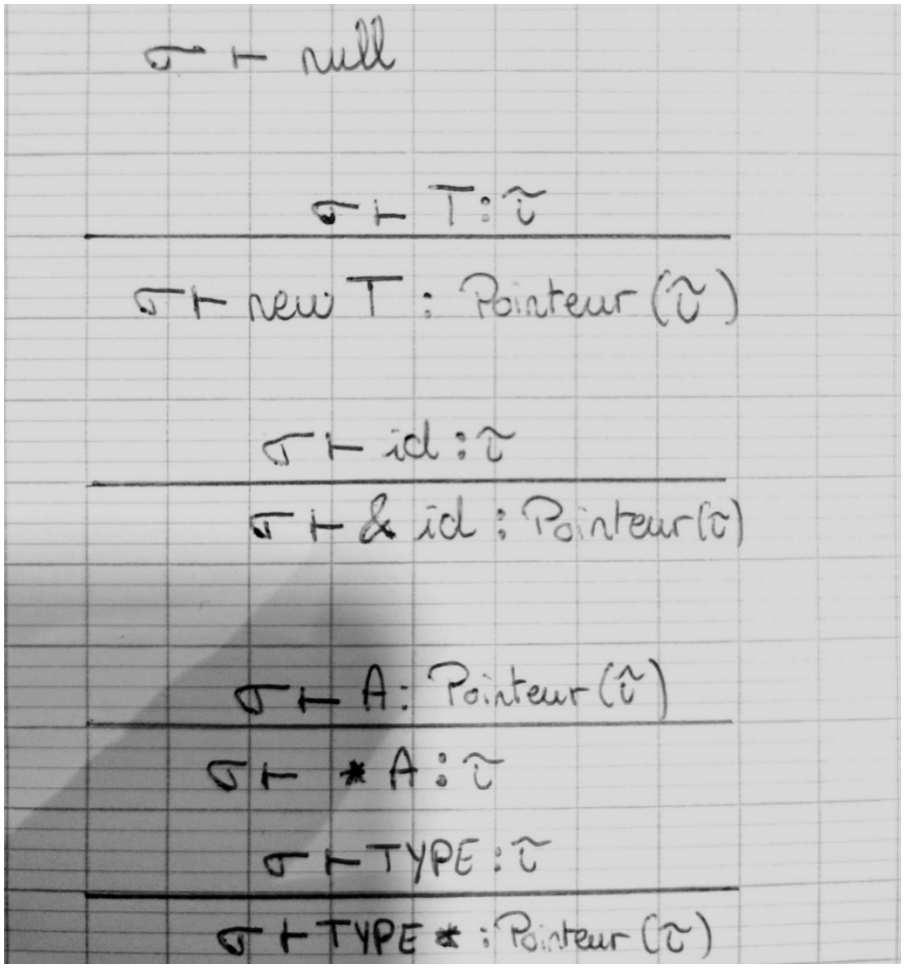
On a par la suite ajouté le type affectable à notre structure d'AST et modifié les instructions comme les affectations et assignations pour qu'elles traitent des éléments de type affectable * expression (auparavant string * expression).

II.3. Passes

La passe TDS traite l'ajout des différentes expressions et des affectables pour vérifier et ajouter les valeurs voulues à la TDS créer par le programme. La passe de typage permet de vérifier la compatibilité des types et de les sauvegarder dans la TDS, un pointeur étant compatible avec un pointeur sur le type Undefined ou sur un type qui est

compatible avec le type pointé. Pour le placement mémoire, un pointeur prend une place en mémoire de 1, ce qui est indépendant du type pointé. Pour la passe de code, on adapte l'accès à la mémoire en fonction des pointeurs en récupérant les adresses mémoires au lieu des valeurs des variables lorsque cela est nécessaire.

II.4. Jugements



III. Assignations

II.1. Lexer et parser

Pour ajouter le traitement de l'assignation d'affectation, nous avons modifié notre lexer et parser afin d'ajouter la règle de grammaires nécessaire au traitement :

$$— \; I \rightarrow \; A \; += \; E \; ;$$

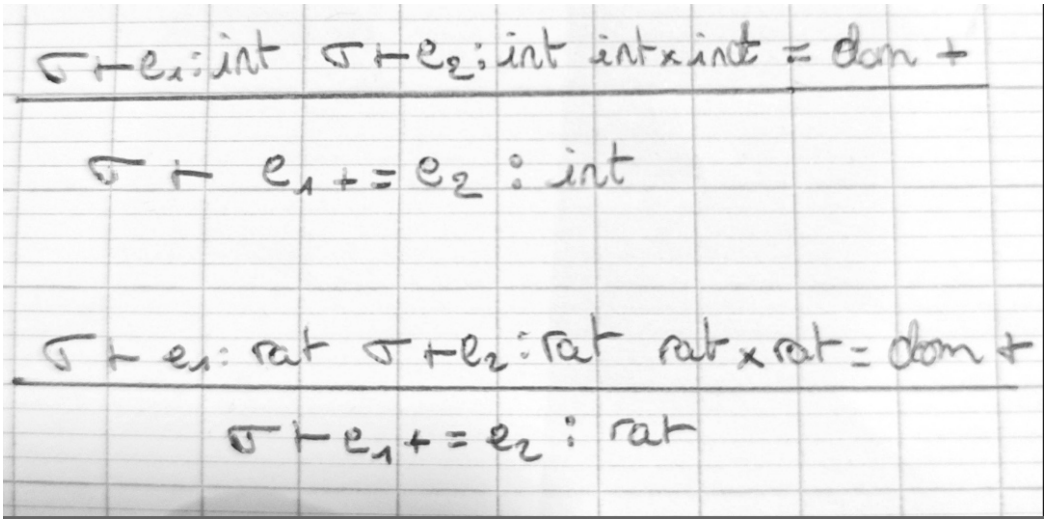
II.2. AST Syntax

L'instruction correspondante est nommée AssignationAdd et prend en compte le traitement d'élément de type affectable * expression.

II.3. Passes

L'addition ne prenant en compte que les types entier et rationnel, à partir de la passe de typage notre instruction AssignationAdd devient AssignationAddInt et AssignationAddRat en fonction du type des expressions traitées. Le code est à la fin une affectation à l'affectable donné de l'addition de l'affectable et de l'expression entrés.

II.4. Jugements



IV. Types Nommés

II.1. Lexer et parser

Pour ajouter le traitement de la définition de type, nous avons modifié notre lexer et parser afin d'ajouter le type des type nommés et les règles de grammaires nécessaires au traitement :

- $PROG \rightarrow TD\ FUN\ PROG$: définition globale (au programme) d'un type nommé ;
- $TD \rightarrow$
- $TD \rightarrow typedef\ tid\ =\ TYPE\ ;\ TD$
- $I \rightarrow typedef\ tid\ =\ TYPE\ ;$: définition locale (à un bloc) d'un type nommé ;
- $TYPE \rightarrow tid$: utilisation d'un type nommé.

II.2. AST Syntax

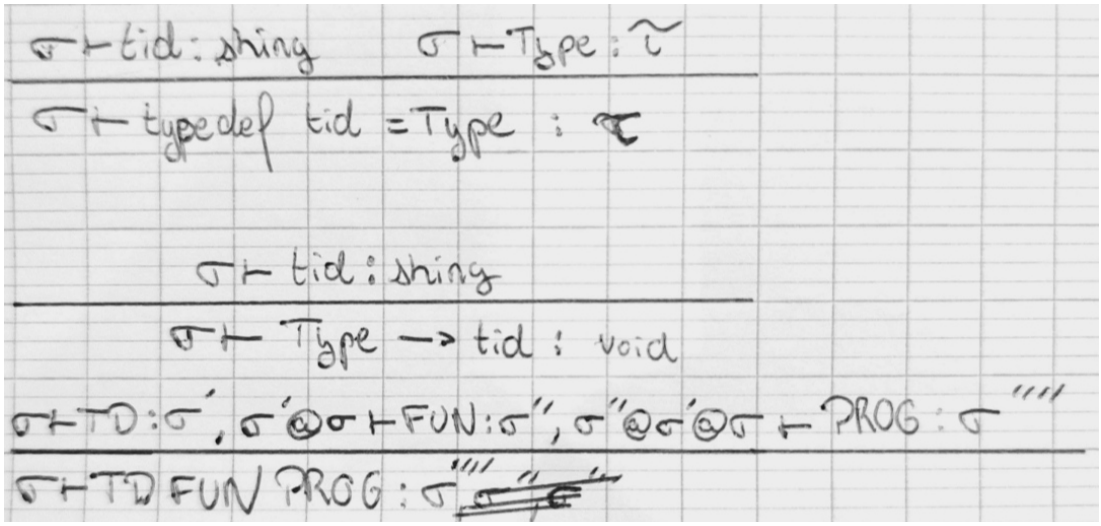
Nous avons modifié l'AST de manière que celui-ci prenne en compte la définition des types nommés ainsi que leur utilisation dans le reste du programme. Les types nommés pouvant être définis en amont du reste du programme, nous avons ajoutés au type Programme un paramètre qui est une liste de définition de types nommés.

II.3. Passes

Nous avons ajouté une fonction analyser_typedef qui est appelée lors d'une instruction ou lors de l'analyse du programme si on a des définitions de types nommés.

Lors de la passe TDS, une InfoVar est ajoutée contenant le nom du type nommé et le type réel qui lui correspond. Ensuite, dans cette même passe, lorsqu'une variable est déclarée avec un type nommé, celui-ci est remplacé par son type réel en parcourant la TDS (avec la fonction nom_to_type), afin que lors de la passe de typage les types puissent être correctement compatibles entre eux.

II.4. Jugements



V. Enregistrement

II.1. Lexer et parser

Pour ajouter le traitement des enregistrements, nous avons modifié notre lexer et parser afin d'ajouter le type des enregistrements et les règles de grammaires nécessaires au traitement :

- $TYPE \rightarrow struct\ \{\ DP\ \}$: définition d'un enregistrement (DP est une liste des types et noms des champs) ;
- $A \rightarrow (A.id)$: accès à un champ de l'enregistrement
- $E \rightarrow \{\ CP\ \}$: création d'un enregistrement avec la liste des valeurs de ses champs

II.2. AST Syntax

Les règles ont été ajouté à l'AST Syntax afin de traiter les données voulues, DefEnregistrement prenant une liste d'expression pour les expressions, afin de pouvoir affecter un enregistrement à une variable, et Acces dans les affectables pour pouvoir accéder aux champs contenus dans une variable de type enregistrement.

II.3. Passes

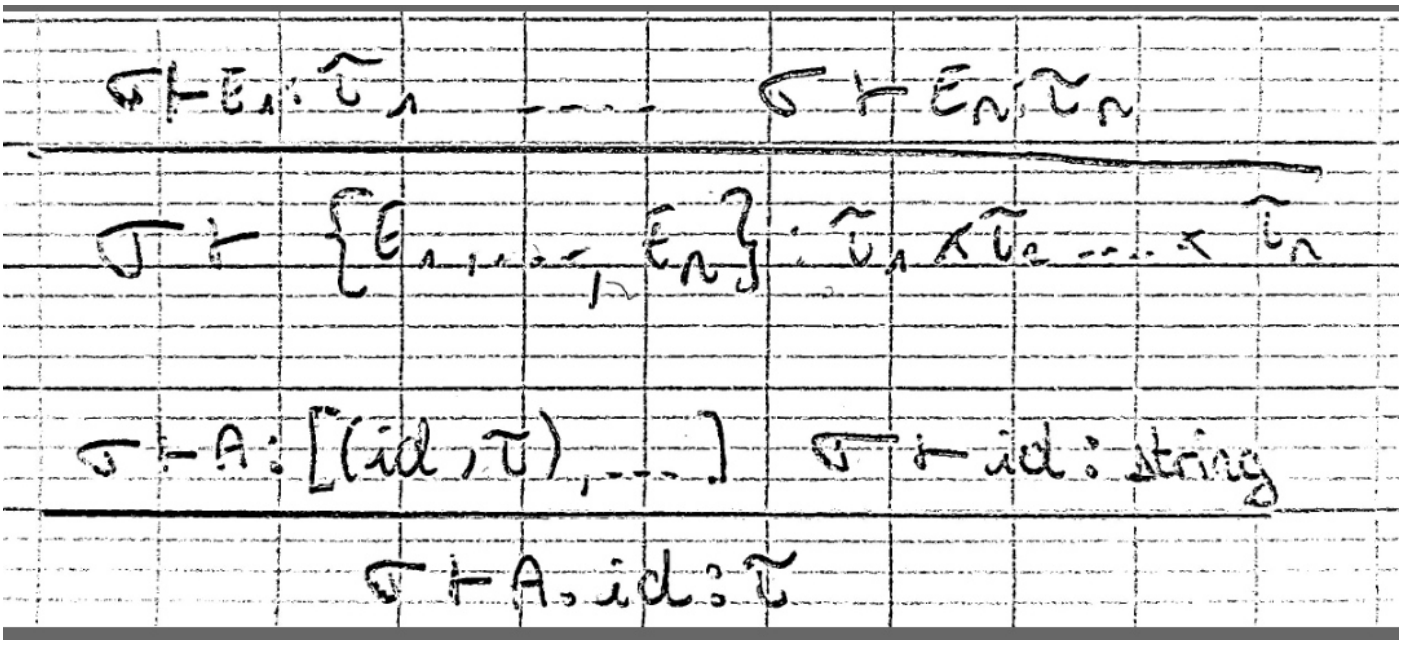
Lors de la passe TDS, chaque type enregistrement est ajouté à la TDS ainsi que les champs définis dans l'enregistrement, de manière à pouvoir les récupérer ensuite lors des accès ou des modifications des éléments d'un affectable de type enregistrement.

Pour pouvoir gérer les conflits de masquage des variables comme dans l'exemple ci-dessous, nous avons ajouté au module TDS une méthode de recherche d'un élément dans la TDS mère de la TDS entrée en paramètre de la fonction (chercherMere). Cette fonction nous permet de vérifier si une variable locale masque une variable globale d'enregistrement, et ce qui permet ou non l'accès aux éléments de la variable de type enregistrement.

```
main{
  struct {int x int y} p = {2 3};
  if ((p.x) < 4) {
    int x = 3;
    print (p.x);
  } else {
    int x = 4;
    print (p.x);
  }
}
```

Dans la passe de typage, on vérifie la compatibilité des types en comparant les types des enregistrements donnés en paramètres dans la liste des types et noms de chacun. Comme pour les pointeurs, un enregistrement requière une place en mémoire de 1, et son accès en mémoire reste le même que pour les autres variables, car chaque champ est enregistré dans la mémoire.

II.4. Jugements



VI. Conclusion

La récursivité des types enregistrements n'a pas pu être traitée, nous avons d'abord ajouté à la TDS le nom du type défini puis traité les éléments, mais cela ne suffit pas à les prendre en compte. Nous avons aussi une erreur que nous ne savons pas résoudre au niveau du typage, durant la phase de typage des types ne sont pas définis pour certains exemples du sujet, comme le troisième exemple du sujet sur les enregistrements. Nous n'avons pas traité la passe de code pour les enregistrements. Des tests unitaires ont été réalisé sur les fonctions ajoutés, et pour les différentes passes nous avons ajouté des tests pour confirmer le bon fonctionnement des passes, exceptés pour les passes de passe au code TAM, les tests automatisés ne fonctionnaient pas à cause de problèmes concernant java sur nos sessions. Nous avons donc effectué des tests à la main pour chacune des passes en utilisant le TAM fourni.