

# ENSEEIH - SCIENCES DU NUMÉRIQUES



---

## Rapport Projet TDL

---

Baptiste Gomez  
Hermas Obou

Janvier 2025

# Contents

1	Introduction	2
2	Evolution des AST	2
3	Jugements de typage	2
4	Pointeurs	3
5	Variables Globales	4
6	Variables Statiques Locales	4
7	Paramètres par Défaut	4
8	Conclusion	5

## 1 Introduction

Dans ce projet, nous allons partir du compilateur RAT fait en TP et étendre ce langage aux pointeurs, variables globales, variables statiques locales et aux paramètres par défaut. Nous verrons dans ce rapport les choix de conceptions fait, les modifications de la grammaire de RAT ainsi que les difficultés que nous avons rencontrées.

## 2 Evolution des AST

- Lors de l'ajout des pointeurs, nous avons dû faire des modifications de la grammaire de RAT, notamment en ajoutant un nouveau symbole non terminal qui correspond aux affectables. Un affectable est un élément auquel on peut affecter une valeur. Cela engendre un tout nouveau type dans les AST qui va comprendre le déréférencement. De plus, une expression admet trois nouvelles entrées : "New", "Null" et "Adresse" pour la gestion des pointeurs.
- Pour l'ajout des variables globales et statiques locales, nous avons ajouté une "DeclarationStatic" et une "DeclarationGlobale" pour pouvoir différencier ces variables des variables classiques. En termes d'AST, une DeclarationStatic est la copie conforme d'une Declaration, idem pour DeclarationGlobale.
- Un nouveau type défaut était nécessaire pour l'ajout des paramètres par défaut pour pouvoir les séparer du reste et garder une trace de ceux-ci. Un paramètre par défaut est seulement caractérisé par une expression et disparaît après la passe de gestion d'identifiant en étant intégrés dans l'appel de fonction.

## 3 Jugements de typage

Expression

- $\sigma \vdash E : \tau$

$$\frac{}{\sigma \vdash A : \tau}$$

- $\sigma \vdash TYPE : \tau$

$$\frac{}{\sigma \vdash (new \ TYPE) : Pointeur(\tau)}$$

- $\sigma \vdash Null : Pointeur(Undefined)$

Déréférencement

- $\sigma \vdash A : Pointeur(\tau)$

$$\frac{}{\sigma \vdash (*A) : \tau}$$

Type

- $\sigma \vdash TYPE : \tau$

$$\frac{}{\sigma \vdash TYPE* : Pointeur(\tau)}$$

Affectation

- $\sigma \vdash A : \tau \quad \sigma \vdash E : \tau$

$$\frac{}{\sigma, \tau_r \vdash A = E : void, []}$$

Adresse

- $\sigma \vdash A : \tau$

$$\frac{}{\sigma \vdash \&A : Pointeur(\tau)}$$

## 4 Pointeurs

Un pointeur est maintenant un affectable qui peut être déréférencer. Nous avons créé une fonction analyser-affectable à chaque passe pour gérer ce nouveau type. Un affectable admet les mêmes restrictions aux niveaux des identifiants que dans le langage RAT simple. De plus un pointeur agit comme une variable classique avec une taille de 1 à la phase de placement en mémoire. Dans la passe de typage, il faut bien faire attention d'analyser le contenu de Deref récursivement pour des problèmes de typage avec des pointeurs de

pointeurs etc. En générant le code TAM, nous avons ajouté un paramètre "write" à la fonction qui analyse les affectable pour savoir si on doit faire un storei ou un loadi de la taille de l'affectable. En effet, \*px = ... et x = \*px sont différents du point de vue de px, dans le premier cas en écriture et dans le deuxième cas en lecture.

## 5 Variables Globales

Les variables globales, du fait de leur déclaration en haut du code source, ne nécessitent pas énormément de changement. En effet, les points importants sont seulement à la déclaration des variables (double déclaration) et à l'affectation (bonne modification de la variable globale). Le placement mémoire et la génération de code ne posent pas de difficultés particulières : on commence par analyser les variables globales dans SB, puis on traite les fonctions en fonction de ces variables globales analysées et il ne faut pas oublier de calculer le déplacement de SB après les variables globales et statiques locales.

## 6 Variables Statiques Locales

Nous avons ajouter en plus de la déclaration, la déclaration statique pour différencier les deux. La détection d'une déclaration statique dans le main se fait au niveau de la passe de placement et génère une erreur. Lors de la passe de placement en mémoire, nous avons différencié les blocs dans le main des blocs dans une fonction pour pouvoir intégrer la gestion des variables statiques locales. Lors d'une analyse de fonction dans le placement en mémoire, il ne faut pas oublier de séparer les déclarations statiques des instructions avant de construire l'AstPlacement.Fonction et de retourner les variables statiques ainsi que le déplacement suivant SB pour pouvoir continuer l'analyse des autres fonctions / blocs.

## 7 Paramètres par Défaut

Les paramètres par défaut ont nécessité la création d'un nouveau type "defaut" qui est en option dans parser.mly pour les garder en mémoire et faire le remplacement si nécessaire lors de AppelFonction dans la passe de gestion

d'identifiant. Dans cette même passe, on intègre aussi les paramètres par défaut dans `Fonction` et nous n'avons plus d'occurrence de ces paramètres par défaut dans les passes suivantes.

## 8 Conclusion

Les principales difficultés que nous avons rencontrés sont au niveau de `Menhir` et du `parser.mly`. Nous avons bloqué pendant longtemps sur des erreurs de syntaxe au niveau de `Menhir` lors de l'exécution du compilateur. Les variables statiques locales nécessitant beaucoup de changements nous ont aussi ralenti lors de la planification des modifications nécessaires.