Analyse Sémantique

Arthur Garnier

1 Résumé

- source -> Analyse Syntaxique (AST) -> Analyse sémantique (AST + TDS) -> Optimisations -> Prod. de code -> Assembleur
- Une table des symboles par région (fonction, bloc anonyme, ...)

La TDS se rempli au fur et à mesure que l'on rencontre des déclarations. Il faut se repérer dans le code avec une pile des régions ouvertes (PRO).

La TDS se rempli à partir de la table des déclaration, la table sémantique à partir de la table des instructions de l'AST.

1.1 Gestion dynamique de la mémoire

Mémoire	Pile
Tas ↓ ——— ↑ Pile	Code généré ——— Variables implémantées (Zone de données)

2 La zone de données

2.1 Objets statiques

Ce sont des objets qui ne changent jamais de place dans cette zone. Ils sont créés au début de l'éxécution du programme, et qui disparaissent à la fin.

2.2 Objets dynamiques

2.2.1 Gérés de manière automatique

Ce sont des objets créés au début de l'execution d'un bloc (fonction, procédure, anonyme). Ce sont typiquement des objets attachés aux blocs. Ils ne changent jamais de place dans l'environnement.

2.2.2 Gérés de manière explicite

Ce sont des objets créés et détruits à la demande (malloc, calloc, free, new, ...), ils sont rangés dans le tas.

3 Environnement - Gestion dynamique de la mémoire

Environnement = Bloc d'activation ou enregistrement d'activation (ie : frame)

Un environnement contient les informations nécessaires à l'exécution d'une fonction, procédure ou bloc anonyme.

3.1 Premières idées d'infos nécessaires

- L'ensemble des variables locales et paramètres
- Une place pour le résultat de fonciton (s'il y a lieu)
- Sauvegarder les registres
- Sauvegarder l'adresse de retour
- Sauvegarde de la base de l'appelant
- Chaînage statique

On parlera de gestion dynamique de la mémoire :

- A chaque appel de routine, est alloué un environnement dans la pile
- A un instant, il peut exister plusieurs exemplaires d'une variable ou paramètre.
  - L'adresse d'une variable n'est plus fixe
    - $\Rightarrow$ Le calcul se fait dynamiquement
    - $\Rightarrow$ Utiliser un mécanisme d'adressage variable : Adressage basé

$@Variable = \frac{\text{Base d'une routine}}{DYN} + \frac{\text{Déplacement}}{STAT}$

- A chaque sortie de routine (=fin de son execution)
  - On supprime son environnement de la pile (=on libère la place)
  - On revient à l'environnement appelant en restaurant la valeur de la base de cet envirionnement appelant.
    - $\Rightarrow$ Cela suppose que lors de chaque appel de routine courante.

4 Systèmes à l'éxécution

4.1 Accès variable locales et paramètres

@objet local = Base courage = déplacement (TDS)

4.2 Accès aux objets globaux

@objet Local = Base zone globale + deplacement

4.3 Accès aux objets (ni locaux ni globaux)

Ce sont des objets déclarés dans un bloc englobant. Donc l'environnement est forcément déjà dans la pile.

@objet non local = Base (de l'environnement de déclaration de l'objet) + déplacement

Pb : Comment retrouver cette base du bloc de définition de la variable non locale ?

3 solutions :

1. Ajout du n° de région dans la pile

Problème, on va parcourir des régions que l'on ne devrait pas consulter (problème de portée).

2. Utiliser l'information **d'imbrication** des régions

De la base de l'appelant la sauvegarde de l'ancienne base est appelée chaînage dynamique.

On trouve l'adresse d'une variable en partant de la base + le déplacement (déplacement présent dans la table des symboles)

Retrouver la base du bloc de déclaration d'une variable (non locale).

- Les règles de portée des déclarations garantissent que :
- Si I est déclarée dans un bloc X de n° d'imbrication N_X
- Si I est visible dans un bloc d'imbrication Y N_Y (Alors $N_x \leq N_y$) alors remonter (N_y - N_x) chaînages statique pour trouver la base du bloc qui déclare I.

4.4 Les paramètres

Différents modes de passage :

- Par valeur

Le paramètre formel prend la valeur de l'effectif

- Par adresse

On copie dans le paramètre formel l'adresse du paramètre effectif

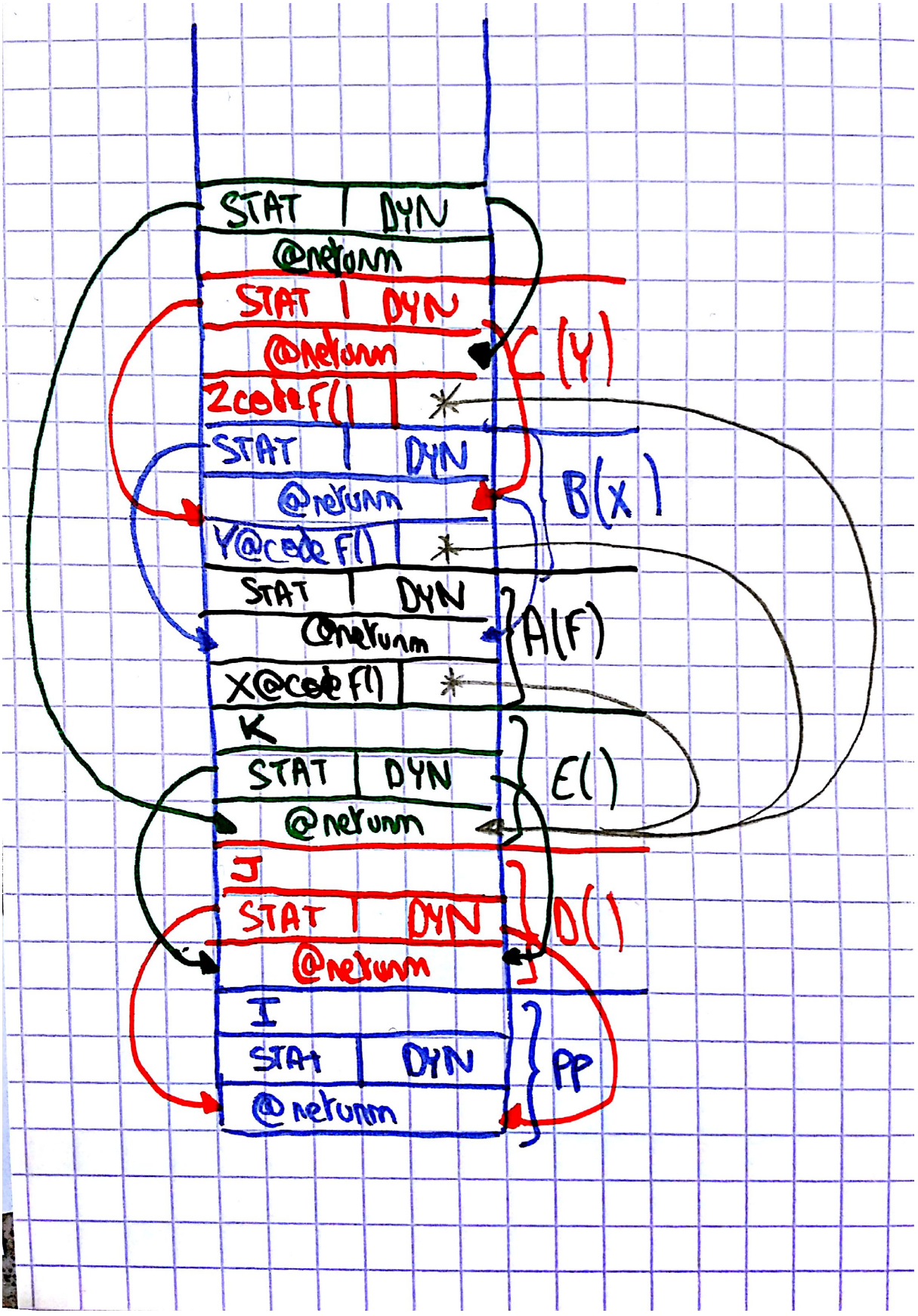
3. Utilisation d'un adresseur ou DISPLAY

C'est un tableau d'adresse de taille = le nombre maximal d'imbrication du programme. A tout moment, DISPLAY[i] contient la base du dernier bloc d'imbrication i actif

Avantage : Accès direct au bloc (plus de boucle assembleur à générer)

4.5 Les paramètres procéduraux

```
Programme PP
    I entier
    Procedure A (X : Proc)
        Procedure B(Y : Proc)
            Procedure C (Z: Proc)
                Z()
            C(Y)
        B(X)
    Procedure D ()
        I : Entier
        Procedure E()
            K : Entier
            Procedure F()
                I=10
                J=20
                K=30
            A(F)
        E()
D()
fin()
```


STAT | DYN
@return
STAT | DYN
@return
C(Y)
2code F()
STAT | DYN
@return
B(X)
Y@code F()
STAT | DYN
@return
A(F)
X@code F()
K
STAT | DYN
@return
E()
J
STAT | DYN
@return
D()
I
STAT | DYN
@return
PP

Pour un param procédural, mettre dans la pile, depuis l'appel concerné (ici A(F))

Deux information :

- @Code procédure (ici @code F)
- chainage Statique du paramètre procédural (ici Ch STAT de F)

5 Exercice Feuille