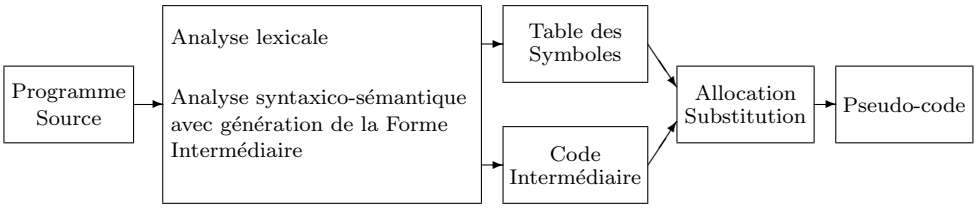
Chapitre 3

Allocation - Substitution



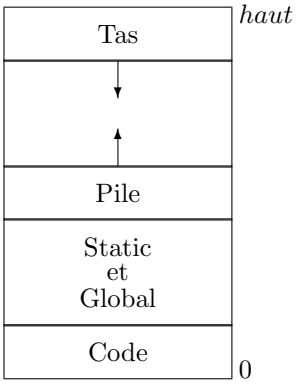
L'analyse syntaxico-sémantique a permis d'engendrer deux textes :
- Le dictionnaire qui contient les noms des objets manipulés par le programme.

- Le texte intermédiaire correspondant aux instructions du programme.

L'allocation a pour objet d'attribuer une place, i.e, une adresse en mémoire ou, d'une façon générale un mécanisme d'adressage, aux objets que le programme manipule. L'allocation opère donc sur la table des symboles et elle produit la TS ≪ allouée ≫, où à chaque objet, est associé une information d'adressage. L'allocation peut effectuer certaines vérifications sémantiques touchant par exemple, à la cohérence des déclarations des variables.

La substitution a pour objet de remplacer, dans les instructions, chaque occurrence d'un objet par son adresse, ou encore le mécanisme d'évaluation de son adresse. La substitution ≪ travaille ≫ sur le code intermédiaire en s'aidant du dictionnaire alloué produit par l'allocation. Elle produit un pseudo-code qui sera repris par la phase génération.
Schéma classique d'allocation mémoire

Espace logique d'adressage unique du programme (mémoire virtuelle) alloué par l'OS



- Les zones contenant le code, les données statiques et globales ont des tailles connues (par le compilateur par ex).
- Le programme source les accède implicitement avec des étiquettes (noms).
- Le tas et la pile s'accroissent et rétrécissent durant le temps d'exécution du programme.
- Meilleure utilisation de la mémoire si la pile et le tas s'accroissent chacun vers l'autre (Knuth). - Codes et données séparés ou entrelacés - On utilise l'espace d'adressage virtuel géré par le proc : on n'alloue pas dynamiquement cet espace.

Tas : heap
Pile : stack
zone « Code » : contient le programme binaire (zone où l'on n'écrit pas par défaut).
zone « statique » : contient les données globales du programmes, i.e., celles qui sont en vie durant tout le temps d'exécution.
zone « tas » : utilisée pour stocker les données allouées dynamiquement.
zone « pile » : passage de paramètres et de contextes lors d'appels de fonctions, variables locales, gestion de la récursivité, etc.

Vue globale de la mémoire

Un compilateur voit l'unique espace d'adressage virtuel du programme qu'il est entrain de compiler. Le système d'exploitation voit tous les espaces d'adressages des programmes et leur affectation à l'espace physique. Le processeur lui ne voit que des adresses mémoire : un long tableau uni-dimensionnel.

3.1 Allocation

Variables simples

A une variable, on associe en général du point de vue de l'allocation, une longueur et un cadrage. La longueur d'une variable est le nombre de cellules de mémoire servant à sa représentation. Cette longueur peut être constante ou variable. Le cadrage d'une variable est une notion que beaucoup d'ordinateurs obligent à introduire. En effet, sur ces ordinateurs, pour qu'une variable soit directement utilisable pour un calcul, il est nécessaire qu'elle soit alignée sur une limite spécifique (de demi-mot, de mot,de double mot, ...). Ainsi, nous

avons pour chaque variable, deux informations : sa longueur et son cadrage. Pour procéder donc à l'allocation de place mémoire, et puisqu'il s'agit de scalaires, donc d'entités indépendantes, on peut regrouper les variables en fonction de leur longueur et leur cadrage, de façon à perdre le moins d'espace mémoire possible. Les éléments de longueur variable posent un problème quant à leur représentation. Les schémas les plus réalistes, dans les langages de programmation, précisent que la longueur d'un élément est variable, mais ne peut pas dépasser une certaine valeur (i.e , la longueur maximale). En fonction de cette longueur maximale, on se pose le problème de l'implémentation de ces éléments. Il existe deux types de méthodes :
- Les méthodes qui consistent à allouer systématiquement la longueur maximale.
- Les méthodes qui consistent à allouer uniquement la place nécessaire.

Le second type de méthodes pose le problème de la gestion de la mémoire, il oblige quelquefois à utiliser le ramasse-miettes, ainsi est-il préférable de se tourner vers la 1ère à chaque fois que cela est possible.

Groupement Homogène

On entend par groupement homogène, tout groupement au niveau du langage d'objets de nature (et de longueur) identique.

Une première distinction doit être faite :
- Dimensions du groupement connues à la compilation (e.g Fortran).
- Dimensions du groupement non connues à la compilation (e.g Algol, Pascal, C, ...).

Cas des tableaux

L'adressage d'un élément d'un tableau à une seule dimension (vecteur) ne pose guère de problème, puisque les éléments d'un vecteur sont stockés séquentiellement en mémoire. Dans certains langages, on utilise les bornes pour le calcul de l'adresse de l'élément et le test de validité de l'indice. Le problème se complique dans le cas des tableaux multidimensionnels. Lorsque les bornes du tableau sont connues à la compilation, on peut directement bâtir le mécanisme d'adressage qui évite un grand nombre de calculs. Nous devons choisir avant tout un mode de représentation du tableau en mémoire. En effet, la mémoire est un tableau à une seule dimension. On parlera alors de rangement ligne/ligne ou colonne /colonne.

Par exemple :
- par ligne (langage C) : les éléments d'une ligne de la matrice sont consécutifs en mémoire.
- par colonne (Fortran) : les éléments d'une colonne de la matrice sont consécutifs en mémoire.

Rangement ligne par ligne

Exemple : Soit le tableau T[-2 :1 ; 1 :2], Donner le rangement des éléments du tableau ligne par ligne.
Nous remarquons que c'est l'indice 2 qui varie le plus vite.
Calcul de l'adresse de l'élément T[i,j] ?

@T[i,j] = @baseT + (i-(-2)) * (2-1+1) * taille d'un élément + (j-1) * taille d'un élément.

Généralisation à plusieurs dimensions
Soit $T[U_1 : L_1; U_2 : L_2; ...; U_n : L_n]$
$@T(i_1, i_2, ...i_n) = @baseT + (i_1 - U_1) * (L_2 - U_2 + 1)... * (L_n - U_n + 1) *$ taille d'un élément $+(i_2 - U_2) * (L_3 - U_3 + 1) * ... * (L_n - Un + 1) *$ taille d'un élément $+... + (i_n - U_n) *$ taille d'un élément

Soit

$$\begin{cases} d_j = L_j - U_j + 1 \\ d_{n+1} = 1 \end{cases}$$

$$@T[i_1, i_2, \ldots, i_n] = @baseT + (\sum_{j=1}^{n}(i_j - U_j) \prod_{k=j+1}^{n+1} d_k) \times taille\ d'un\ elt \quad (3.1)$$

$$= @baseT + (\sum_{j=1}^{n} i_j \prod_{k=j+1}^{n+1} d_k) \times taille\ d'un\ elt - (\sum_{j=1}^{n} U_j \prod_{k=j+1}^{n+1} d_k) \times taille\ d'un\ elt \quad (3.2)$$

Cette formule met en évidence deux parties :
. Une partie variable = $(\sum_{j=1}^{n} i_j \prod_{k=j+1}^{n+1} d_k) \times taille\ d'un\ elt$
. Une partie constante = $(\sum_{j=1}^{n} U_j \prod_{k=j+1}^{n+1} d_k) \times taille\ d'un\ elt$

Afin de référencer un élément de tableau, nous avons besoin de vérifier que les indices appartiennent bien au domaine autorisé, il sera donc nécessaire de mémoriser les informations U_i, L_i, d_i et n (nombre de dimensions).

n		
U_1	L_1	d_1
U_2	L_2	d_2
$\vdots$	$\vdots$	$\vdots$
U_n	L_n	d_n
$ConstPart$		@base

Ces descriptifs d'un tableau constituent ce que l'on appelle le vecteur de renseignements.
Dans le cas des langages, à allocation statique, c'est à dire pour lesquels, les bornes sont connues à la compilation et ne sont pas variables, on peut engendrer le code correspondant au calcul de l'adresse d'un élément ou même effectuer le calcul sans avoir recours à des vecteurs de renseignements (TS).
Dans le cas de l'allocation dynamique, les bornes sont inconnues, elles peuvent varier d'une exécution à une autre.
Le compilateur intervient donc pour :
- générer un code permettant le calcul des paramètres d'allocation U_i, L_i, d_i et de la taille maximale.

- insérer dans le code, la macro-instruction d'allocation alloc (taille, adrbase).

La déclaration d'un tableau se traduit par :
Soit un tableau $T[U_1 : L_1; U_2 : L_2; ...; U_n : L_n]$:

Algorithme de calcul de la taille d'un tableau
debut
$i := 1$; (* pour compter le nombre de dimensions du tableau*)
$taille := 1$;
$Conspart := 0$;
Pour chaque dimension i du tableau
faire Calculer U_i ;
Calculer L_i ;
si $U_i > L_i$ alors ('Erreur sémantique : borne inférieure supérieure à la borne supérieure')
sinon debut calculer d_i
conspart := conspart * $d_i + U_i$
taille := taille * d_i
fin
fsi
fsi
fait
taille := taille * taille d'un élément
conspart := -conspart
alloc (taille, adrbase)
fin

Remarque : Le vecteur de renseignement est dans la zone de données statique.

Représentation colonne/colonne

Soit un tableau M [-2 :1 ; 1 :3]
$@M[i_1, i_2] := @baseM + [(i_2 - U_2) * (L_1 - U_1 + 1) + (i_1 - U_1)] * tailled'unelt$

L'indice variant le moins vite est l'indice colonne.

Généralisation
$@M[i_1, i_2, ..., i_n] := @baseM + (i_n - U_n) * d_1 * d_2 * ... * d_{n-1} + (i_{n-1} - U_{n-1}) * d_1 * d_2 * ... d_n - 2 + ... + (i_2 - U_2) * d_1 + (i_1 - U_1) := @baseM + (\sum_{j=1}^{n} (i_j - U_j) \prod_{k=0}^{j-1} d_k)$

avec

$$\begin{cases} d_0 = 1 \\ constpart = -\sum_{j=1}^{n} U_j \prod_{k=0}^{j-1} d_k) \end{cases}$$

Représentation des tableaux creux

On ne représente en mémoire que les éléments non nuls. Si le tableau possède une symétrie (e.g matrice triangulaire), on peut définir une fonction spécifique

d'accès à ce tableau. On doit alors construire la fonction d'adresse.

Par contre, si l'on a un tableau creux non symétrique, on peut utiliser l'une des deux techniques suivantes :
- Application d'une fonction de hashcoding sur les indices de la référence au tableau et recherche de l'adresse de l'élément.
- Description de l'absence ou présence d'éléments dans un tableau par un vecteur associé. Ce vecteur associé est considéré comme une suite de bits, un bit à 1, spécifiant que l'élément associé est présent.

Le choix de l'une des méthodes est déliquat, il dépend en fait des possibilités qu'offre la machine pour la manipulation des vecteurs de bits, ou pour les accès par hashcoding.

Groupements hétérogènes

On entend par groupement hétérogène, tout groupement, au niveau du langage, d'objets de natures différentes en général. Dans un groupement hétérogène, on doit allouer les éléments dans l'ordre qui a été spécifié par le programmeur. En d'autres termes, on ne peut pas réorganiser les éléments de façon à minimiser les pertes de place dues au cadrage éventuel. Pour un groupement hétérogène, lors de l'allocation, on doit disposer d'une description du groupement. Celle-ci est composée d'une suite d'éléments. Un élément contient les informations tels que :
- son nombre d'occurrences (qui peut être variable ou constant)
- sa longueur
- ...

Substitution

L'objet de la substitution est de remplacer, dans les instructions, chaque occurrence d'un objet par son adresse ou encore le mécanisme d'évaluation de son adresse. La substitution travaille sur le texte intermédiaire (instructions) d'une part et sur le dictionnaire "alloué" d'autre part. Donc, cette phase, à partir de dictionnaires alloués et d'un texte intermédiaire crée un pseudo-code, où les objets sont remplacés par leur adresse ou par un mécanisme d'évaluation de son adresse. Ce pseudo-code sert d'entrée pour la phase génération de code.