Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Compilation
Licence d'informatique 3e année
Éric Laporte

Arbres abstraits
Tables des symboles

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

`Nasm` pour le projet

Une liste de registres et d'instructions suffisante pour le projet de compilation

Registres

Registres de travail

`rbx, r12-r15`

`rax, rdi, rsi, rdx, rcx, r8-r11`

Registres de pile `rsp, rbp`

`rsp` pointe sur la plus petite adresse dans la pile

`rbp` pointe sur la plus petite adresse hors du bloc d'activation

et leurs sous-registres

`Nasm` pour le projet

Opérations

add
sub
imul
idiv
cmp
je
jne
jg
jng
and
or
xor

} bit à bit

Instructions de copie

mov
movsx
push
pop

`Nasm` pour le projet

jmp
syscall
call
ret

Instructions de saut

`syscall` permet aussi de lire et écrire

Code pour les fichiers `stdin`: 0

`stdout`: 1

db
dd
dq

Déclaration de données statiques

Afficher la pile pour comprendre ce qui ne va pas

Bibliothèque `cpu2.0`, fonction `stackview`

Traduire en **nasm**

On suppose que les 8 octets en sommet de pile contiennent un des opérandes

a := b
(b : sommet de pile)

```
pop qword [a]
```

a := a+4
(a : sommet de pile)

```
add [rsp], qword 4
```

b := b - *c
(c : sommet de pile)

```
pop rax
mov rax, [rax]
sub [b], rax
```

b := b * (*(c+base))
(c : sommet de pile)

```
pop rax
add rax, rbp
mov rax, [rax]
imul qword [b], rax
```

Traduire en `nasm`

a=12/*b
(b : sommet de pile)

```
pop rcx
mov  rax, 12
mov  rdx, 0
idiv qword [rcx]
mov  [a], rax
```

if a = 0 goto loop3
(a : sommet de pile)

```
pop rax
cmp rax, 0
je loop3
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Une façon plus systématique de traduire en **nasm**

```
mov eax, dword [a]
push rax
mov ecx, dword [x]
push rcx
pop rdx          ; x
pop rax          ; a
imul eax, edx
push rax         ; ax
push rcx         ; x
pop rdx          ; x
pop rax          ; ax
imul eax, edx
push rax         ; ax²
mov eax, dword [b]
push rax         ; b
push ecx         ; x
pop rdx          ; x
pop rax          ; b
imul eax, edx
push rax         ; bx
```

```
pop rdx          ; bx
pop rax          ; ax²
add eax, edx
push rax         ; ax²+bx
mov eax, dword [c]
push rax
pop rdx          ; c
pop rax          ; ax²+bx
add eax, edx
```

Calculer ax^2+bx+c dans la pile

Méthode

Empiler 2 opérandes

Dépiler 2 opérandes

Calculer le résultat

Calcul d'adresses

La mémoire pour les variables peut être la pile de données de **`nasm`**

Variables globales

Adresses absolues statiques ou dans la pile

a:=0

mov [a], qword 0

Variables locales

Adresses relatives au pointeur de base **`rbp`**

b:=0

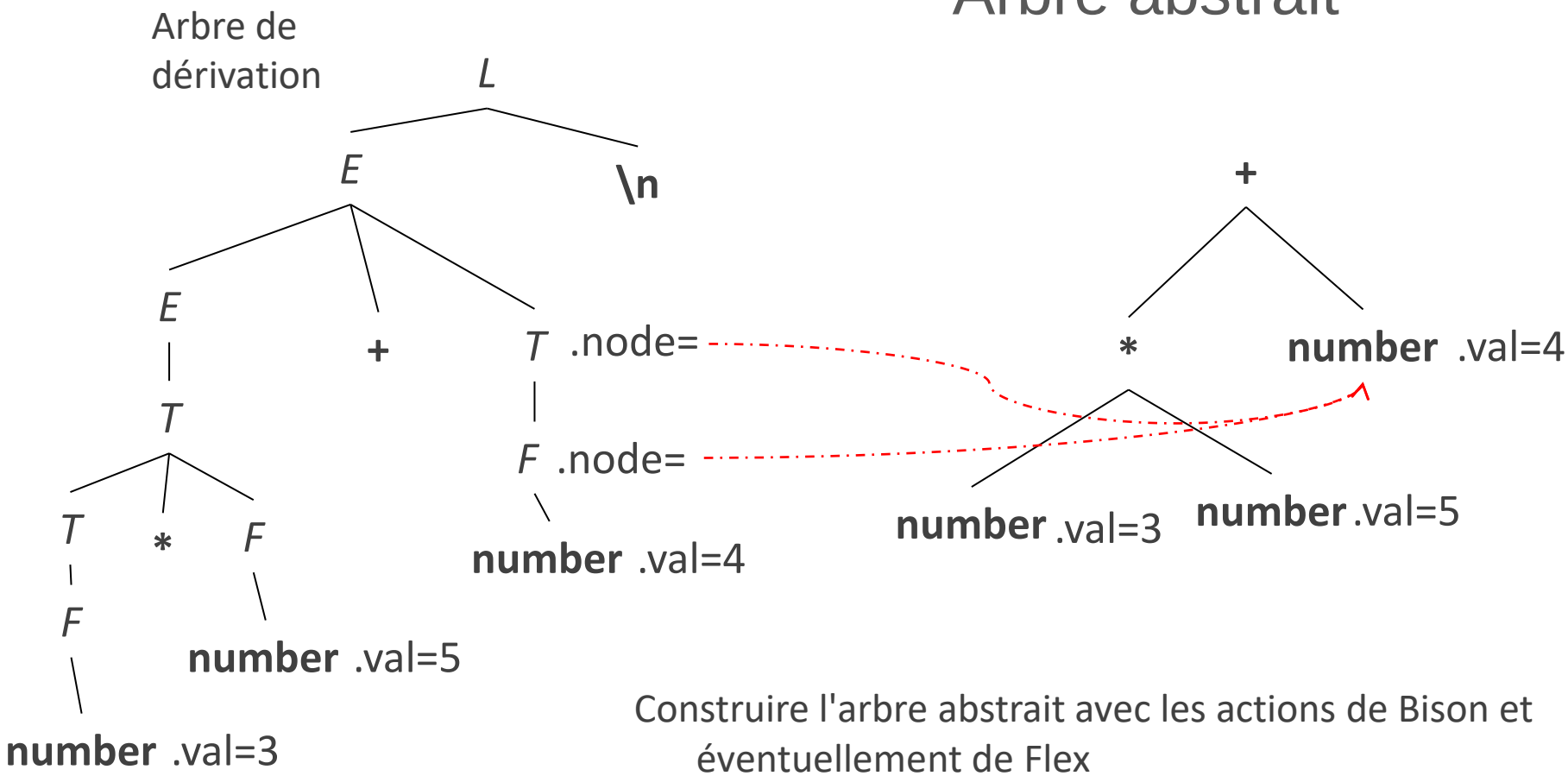
mov [rbp+b], qword 0

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

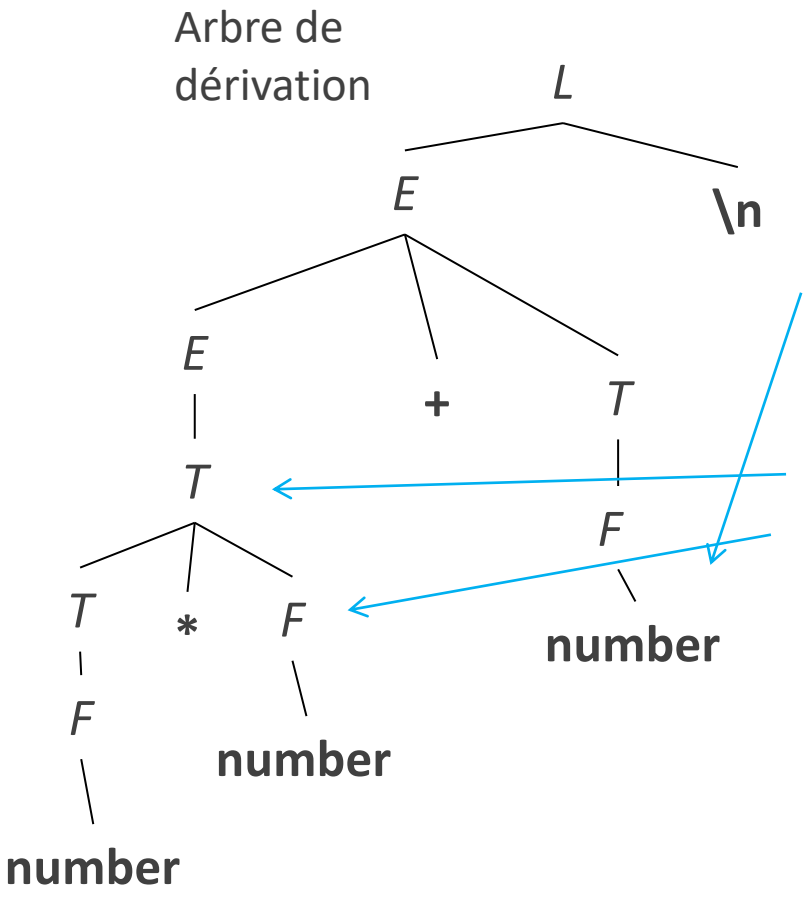
Arbre abstrait



Construire l'arbre abstrait avec les actions de Bison et éventuellement de Flex

Les attributs des nœuds de l'arbre de dérivation sont des pointeurs sur les nœuds de l'arbre abstrait

De l'arbre de dérivation à un arbre abstrait



Feuilles de l'arbre de dérivation

Conservées sauf les délimiteurs

Nœuds pour les expressions

remplacés par les opérateurs

disparaissent s'il n'y a pas d'opérateur

Nœuds pour les listes

on en garde un par liste

on aplatit la liste

Arbres d'arité libre

```
typedef struct Node {
  Kind kind;
  struct Node *firstChild, *nextSibling;
} Node;
```

Fonctions utilisées

`Node *makeNode(Kind kind)`	crée un nœud
`void addSibling(Node *node, Node *sibling)`	ajoute à un nœud un frère `sibling`
`void addChild(Node *node, Node *child)`	ajoute à un nœud un fils `child`

De l'arbre de dérivation à un arbre abstrait

Exercices

1) remplacer un non-terminal *T* par l'opérateur qui est dessous

2) sauter un non-terminal *T* qui est au-dessus d'un *F*

3) aplatir une liste (*SuiteInstr*)

Nœuds feuilles

Conservés sauf les délimiteurs

Non-terminaux pour les expressions

remplacés par les opérateurs

disparaissent s'il n'y a pas d'opérateur

Non-terminaux pour les listes

on en garde un par liste

on aplatit la liste

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Table des symboles

tab	
a	
x	
input_array	
i	
n	

Mémorise les informations sur les identificateurs contenus dans le programme

Entrées de la table des symboles

Correspondent à des déclarations

Si un même nom est déclaré plusieurs fois, il a plusieurs entrées dans la même table ou dans plusieurs tables des symboles

Table des symboles

tab	
a	
x	
input_array	
i	
n	

Le contenu de la table des symboles évolue pendant la compilation

Quand on entre dans un bloc, on ajoute des entrées pour les nouvelles variables locales

À la fin de la compilation, la table des symboles disparait

Opérations sur la table des symboles

- lire l'entrée correspondant à un identificateur
- insérer une nouvelle entrée
- initialiser la table des symboles

Insérer une nouvelle entrée pour une déclaration

tab	
a	
x	
input_array	
i	
n	

Indiquer ce qu'on sait sur la variable déclarée

Décider où le programme réservera de la mémoire pour la variable à l'exécution

Type

Les variables d'un même type occupent la même taille mémoire

Adresse relative

Indiquer dans une variable **`deplct`** l'adresse du prochain emplacement mémoire disponible

Une table des symboles ou plusieurs ?

tab	
a	
x	
input_array	

i	
n	

Si une fonction utilise une variable locale déclarée dans une autre, c'est une erreur sémantique

Comment détecter l'erreur ?

Solution 1 : plusieurs tables des symboles
Une pour chaque fonction
plus une pour les variables globales

Solution 2 : une seule table des symboles
Compiler chaque fonction séparément
Quand on a fini de compiler une fonction, retirer de la table les entrées de ses variables locales

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Table des symboles

tab	
a	
x	
input_array	
i	
n	

Implémentation avec table de hachage ouverte

N seaux

Une liste chainée dans chaque seau

Quand on trouve un identificateur, on le met en tête de la liste pour le cas où on le retrouverait bientôt

Exemple de fonction de hachage

Identificateur : $c_1 c_2 ... c_n$

$h_0 = 0$

$h_i = k\, h_{i-1} + c_i$

$H = (h_n \bmod 2^{30}) \bmod N$

$k = 613$

$N = 1008$

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Amorçage des compilateurs

Comment écrire un compilateur d'un nouveau langage source ?

En quel langage on l'écrit ?

compilateur auto-hébergé (*self-hosting, self-compiling*)

Comment écrire un compilateur d'un langage dans ce même langage ?

Comment compiler ce compilateur ?

Comment écrire un compilateur pour un nouveau langage cible ?

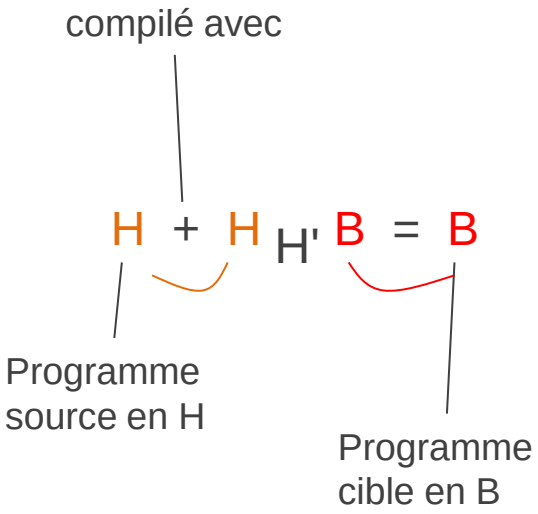
Sur quel processeur on l'exécute ?

compilateur natif

Sur un processeur qui exécute ce même langage ?

Quelques notations

$H_{H'}B$



H : langage source, de haut niveau

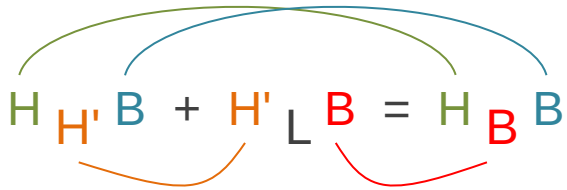
B : langage cible, de bas niveau

Un compilateur est un programme écrit dans un langage

On l'écrit dans un langage de haut niveau (ici H')

Compiler un compilateur

$H_{H'}\,B$



$H_{H}\,B$

$H_{B}\,B$

$H_{H}\,B + H_{L}\,B = H_{B}\,B$

Comment compiler un compilateur $H_{H'}\,B$?

Il faut par exemple un compilateur de H' en B

$H'_{L}\,B$

Si L est un langage de haut niveau, il a fallu compiler $H'_{L}\,B$ d'abord

Compilateur auto-hébergé (*self-hosting compiler*)

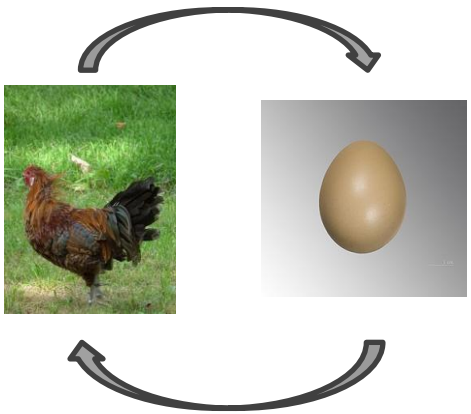
H'=H

Compilateur natif

H'=B

Compiler un compilateur

$$H_{H'}B + H'_{L}B = H_{B}B$$



By 4028mdk09 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

By Didier Descouens (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

$H_{B}B$

By Unknown (Smithsonian Institution) (Flickr: Grace Hopper and UNIVAC) [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)], via Wikimedia Commons

- Si L est un langage de haut niveau, il a fallu compiler $H'_{L}B$ d'abord
- Comment a-t-on créé le premier compilateur ?
- En langage machine, à la main, 1952
- Alick Glennie, à l'Université de Manchester
- Grace Hopper, chez un fabricant de machines à écrire américain

Écrire un compilateur

$H\ {}_{H_0}\ B\ +\ H_0\ {}_{H_0}\ B\ =\ H\ {}_{B}\ B$

$H\ {}_{H}\ B_0 \qquad\qquad H\ {}_{H}\ B$

Comment écrire un compilateur de H en B ?

Solution 1 : auto-amorçage (*bootstrapping*)

Partir d'un compilateur de H_0 en B

Seule la partie frontale change

Solution 2 : migration

Partir d'un compilateur de H en B_0

Seule la partie arrière change

Auto-amorçage des compilateurs

$K_{H}\,B + H_{H}\,B = K_{B}\,B$

$K_{0\,H}\,B + H_{H}\,B = K_{0\,B}\,B$
$K_{1\,K_0}\,B + K_{0\,B}\,B = K_{1\,B}\,B$
...
$K_{K_{n-1}}\,B + K_{n-1\,B}\,B = K_{B}\,B$

Écrire le premier compilateur d'un nouveau langage K

On ne peut pas l'écrire en K dès le départ

- version minimale K_0 écrite en H
- version plus étendue K_1 écrite en K_0

...

- version finale K écrite en K_{n-1}

Auto-amorçage

Versions du compilateur
Versions du langage

Faire migrer un compilateur d'une architecture vers une autre

$H_{H}B_0$ $\quad$ $H_{H}B$

$H_{H}B + H_{H}B_0 = H_{B_0}B$

$H_{H}B + H_{B_0}B = H_{B}B$

Partir d'un compilateur de H en B_0
Écrire un compilateur de H en B
Il faut le compiler deux fois

Première compilation
On obtient un compilateur croisé (*cross compiler*) :
- exécutable sur une architecture,
- produit du code exécutable sur une autre

Deuxième compilation
Toujours sur l'architecture B_0 mais avec le compilateur croisé
Copier le compilateur obtenu sur la machine cible