Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

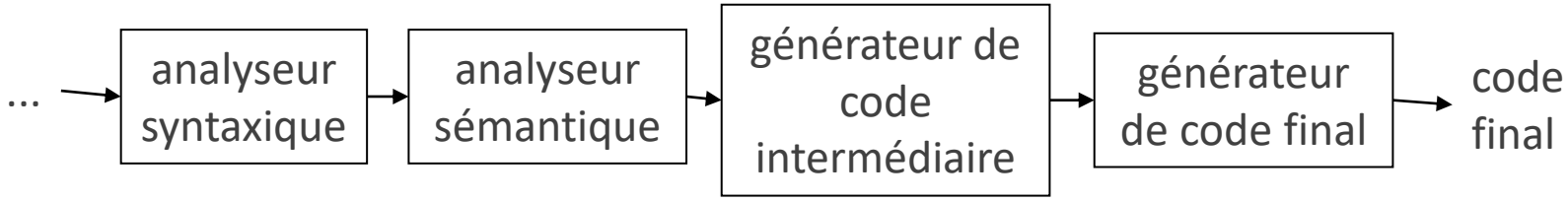
Contrôle de types

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Les types en programmation




```
int check( int tab[], int size);
(...)
   if (!check( values[i], MAX))
```

Les types sont une caractéristique des langages évolués

Représentation des structures de données

Le contrôle de types détecte des erreurs

- Exemple : comparer paramètre formel et paramètre effectif

Langages à objets : enrichissement des possibilités de typage

Contrôle de types statique (à la compilation)

Contrôle de types dynamique (à l'exécution)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Type d'une expression

```
typedef struct list{
   STEntry content;
   struct list *next;
} list;
(...)
   if(cell->next->next)
```

Le développeur déclare le type des variables

Le compilateur calcule le type des expressions

Projet 2020-2021

Types simples `int` (4 octets) et `char` (1 octet)

Structures

Vérifier la validité des expressions

Tenir compte des conversions implicites

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Représenter les types pendant la compilation

```
int size;
struct satellite gps;
int tab[MAX];
char *cursor;
char *cursors[MAX];
```

Comment indiquer le type d'une variable dans la table des symboles ?

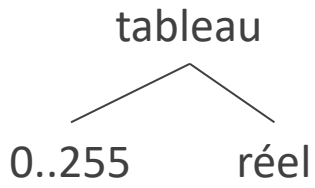
Types simples

Types complexes

Expressions de type

Expressions représentant les types des objets

Expressions de types



Types de base

booléen, caractère, entier, réel... et type vide

Constructeurs de types

tableaux, pointeurs...

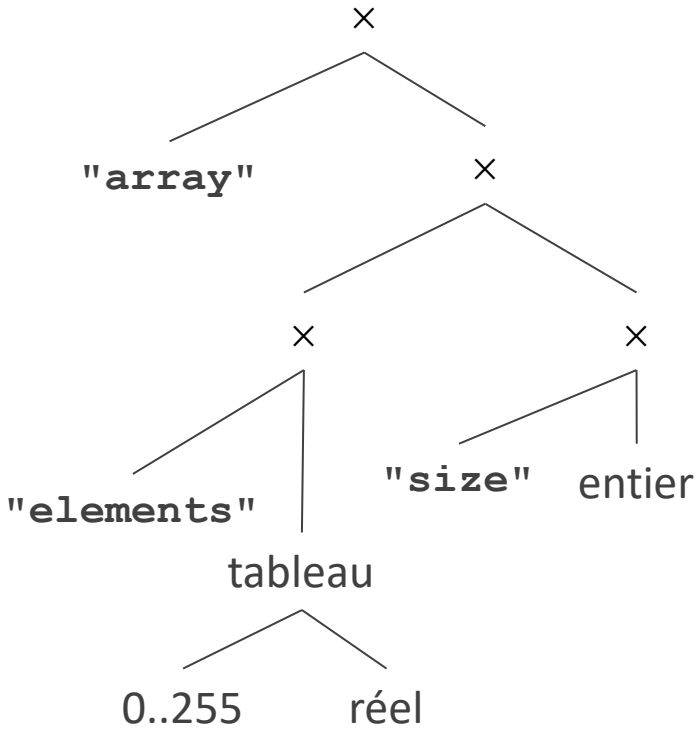
Les expressions de type peuvent être des arbres

Noms de types

nécessaires pour les types à définition récursive (exemple : liste chaînée)

Variables de types

nécessaires pour la généricité (fonctions sur les types)



Constructeurs de types

Tableau

tableau(*I*, *T*) : tableaux d'éléments de type *T* indicés par des indices de type *I*

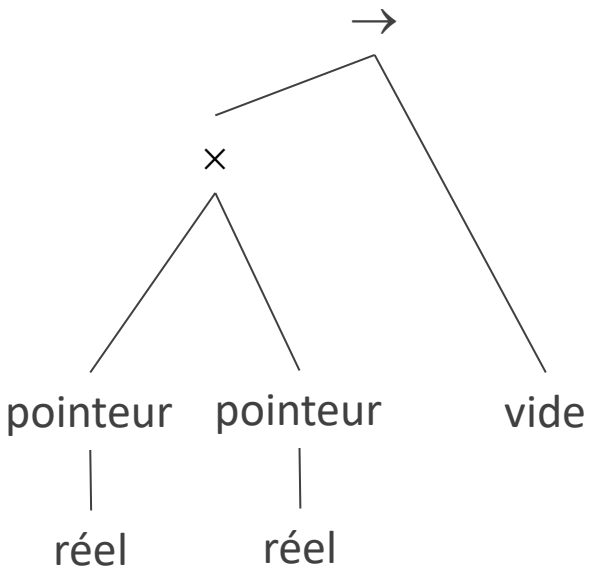
Produit

$T_1 \times T_2$: couples d'un élément de type T_1 et d'un de type T_2

Structure

De même plus des noms pour les champs et pour la structure

Constructeurs de types



Pointeur

pointeur(*T*) : pointeurs sur un objet de type *T*

Fonction

$D \rightarrow R$: fonctions de *D* (type du paramètre) dans *R* (type de la valeur de retour)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Contrôle de types

P --> *D E*

D --> *T* **id ;** *D*

| ε

T --> *B C*

B --> **char**

| **int**

C --> **[num]** *C*

| ε

E --> **num**

| **literal**

| **id**

| *E* **mod** *E*

| *E* **[** *E* **]**

Avec cette grammaire, on peut déclarer une variable comme **char** et l'utiliser comme **int**

Peut-on inclure le contrôle de types dans la syntaxe et dans la grammaire ?

type opérateur

A --> **t1 id1 ; op1 id1**

| **t2 id2 ; op2 id2**

| **t1 id2 ; op1 id2**

| **t2 id1 ; op2 id1**

Il faudrait des règles différentes pour chaque identificateur

On préfère garder l'analyse syntaxique et l'analyse lexicale séparées

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Contrôle de types

P --> *D E*

D --> *T* **id ;** *D*

| ε

T --> *B C*

B --> **char**

| **int**

C --> **[num]** *C*

| ε

E --> **num**

| **literal**

| **id**

| *E* **mod** *E*

| *E* **[** *E* **]**

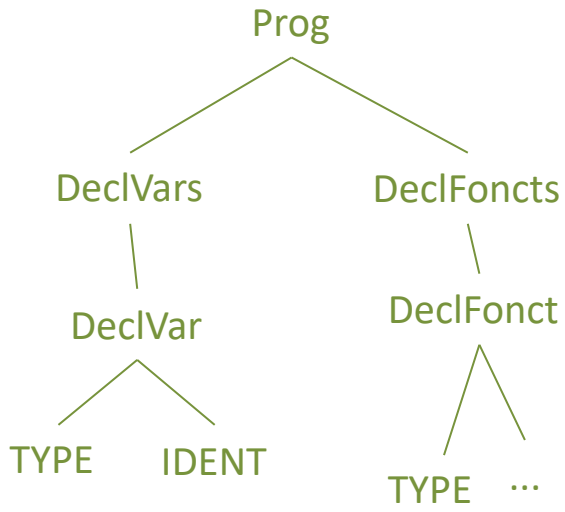
Pendant un parcours de l'arbre abstrait

Le contrôle de types fait partie de l'analyse sémantique

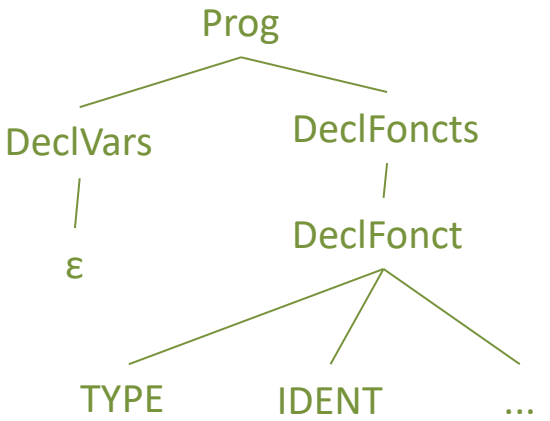
Décider - quelles instructions ajouter

- où les placer dans le parcours

Contrôle de types



arbre abstrait pour
`int count ; int (...)`



Le code source ressemble, mais les arbres abstraits ne se ressemblent pas

arbre abstrait pour
`int count(...)`

Contrôle de types

P --> *D E*

D --> *T* **id ;** *D*

| ε

T --> *B C*

B --> **char**

| **int**

C --> **[num]** *C*

| ε

E --> **num**

| **literal**

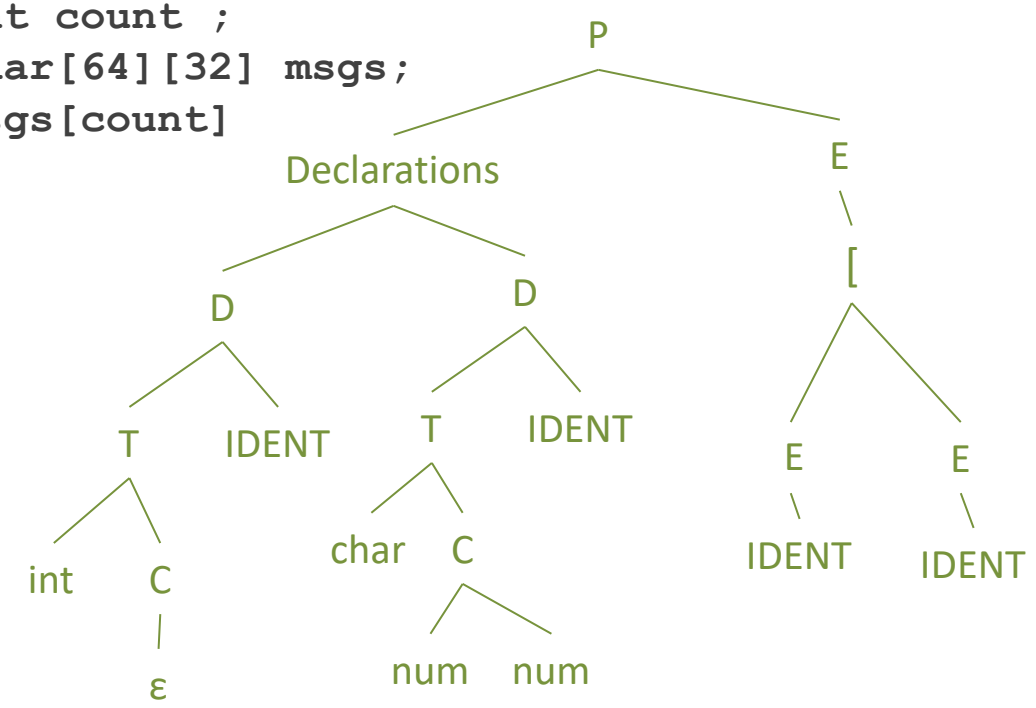
| **id**

| *E* **mod** *E*

| *E* **[** *E* **]**

arbre abstrait pour

```
int count ;
char[64][32] msgs;
msgs[count]
```



literal : constante de type caractère

Déclarations

arbre abstrait pour
`int count ;`
`char[64][32] msgs;`

P --> *D E*

D --> *T* **id ;** *D*

D --> ε

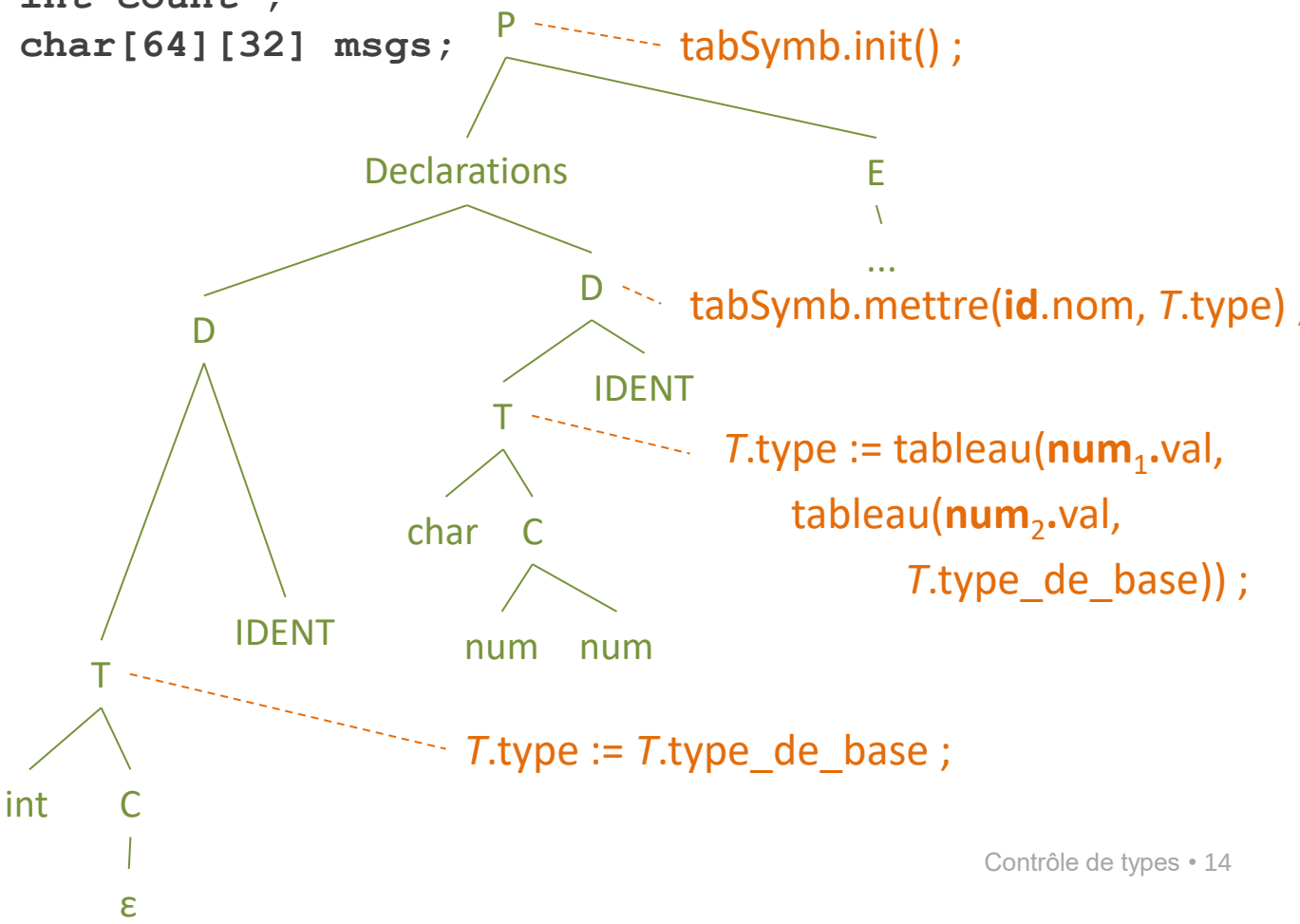
T --> *B C*

B --> **char**

B --> **int**

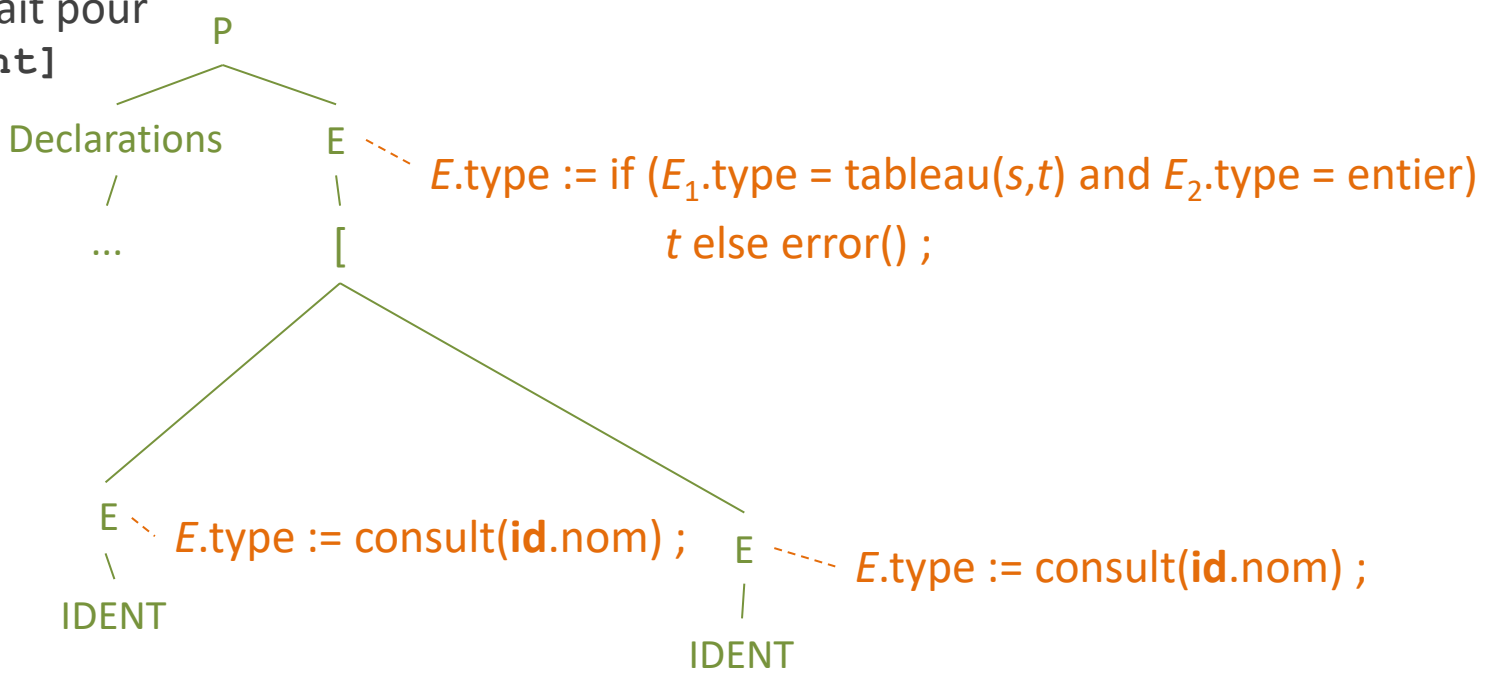
C --> ε

C --> **[num]** *C*



Expressions

arbre abstrait pour `msgs[count]`



E --> **literal**

E --> **num**

E --> **id**

E --> *E* **mod** *E*

E --> *E* **[** *E* **]**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Instructions

S --> **id =** *E*	if (**id**.type ≠ *E*.type) error() ;
S --> **if (** *E* **)** *S*	if (*E*.type ≠ booléen) error() ;
S --> **while (** *E* **)** *S*	if (*E*.type ≠ booléen) error() ;
S --> *S* **;** *S*	

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Fonctions

E --> E **(** E **)** E.type := if (E_1.type = $s \rightarrow t$ and E_2.type = s) t else error() ;

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Comparer deux types

Exemples

Comparer paramètre formel et paramètre effectif

if (E_1.type = $s \rightarrow t$ and E_2.type = s)

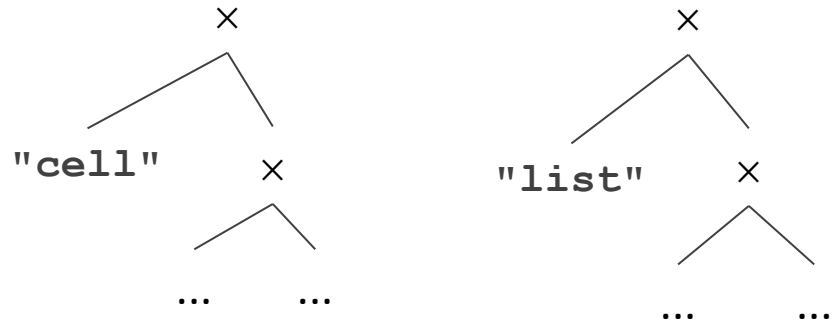
Vérifier le type d'un indice pour un tableau

if (E_1.type = tableau(s,t) and E_2.type = entier)

Algorithme d'unification

On compare des arbres qui peuvent contenir des variables (s et t)

La comparaison peut les initialiser



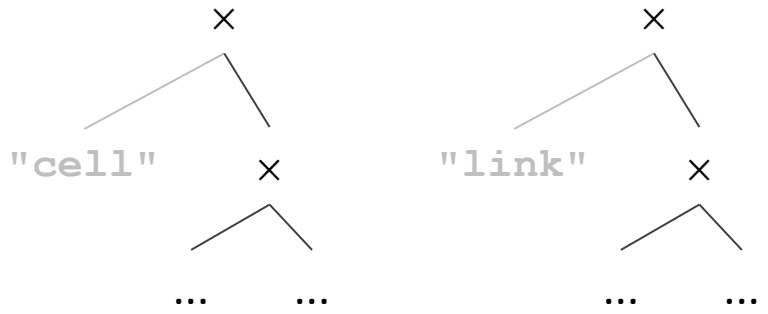
Types nommés

```
typedef struct cell {
    int identifier ;
    struct cell * link ;
} cell ;
typedef struct list {
    int content ;
    struct list * next ;
} list ;
(...)
cell cell0 ;
list list0 ;
cell0 = list0 ; /* compile error */
```

Équivalence nominale

Le nom fait partie du type, on compare seulement le nom

C'est le cas du C



Types nommés

```
typedef struct cell {
      int identifier ;
      struct cell * link ;
} cell ;
typedef cell list ;
(...)
cell cell0 ;
list list0 ;
cell0 = list0 ; /* OK */
```

Équivalence structurelle

Le nom ne fait pas partie du type, on compare seulement le reste de l'expression de type

Langages fonctionnels

Équivalence nominale large

En C, un typedef ne définit pas un type nouveau

Un nom introduit par typedef ne fait donc pas partie du type

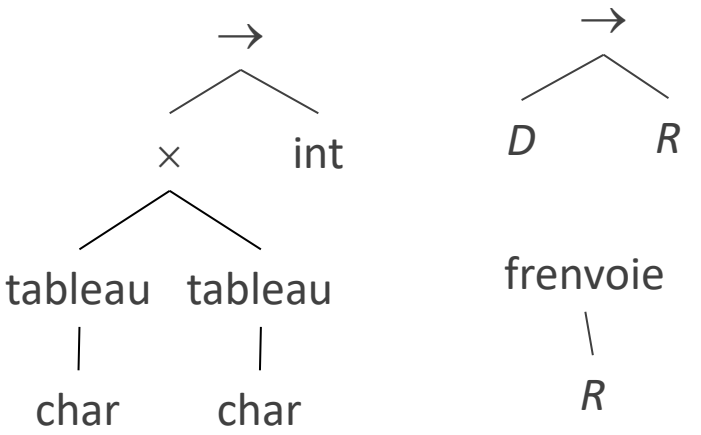
Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Déclarations de types en C

int compare(char s1[], char s2[]) ;



Type fonction

En C, un type fonction tient compte du type

- de la valeur de retour
- et des paramètres

Dans une déclaration de prototype, il est recommandé de préciser les paramètres (mais facultatif)

Déclarations de types en C

```
int (*myfunc)() ;
```

Type pointeur sur fonction
Sert à ce qu'une fonction soit
- élément de tableau
- paramètre d'une autre
- valeur de retour d'une autre
...

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Déclarations de types en C

```
int tab[ ]();
/*   non :
tableau(frenvoie(int)) impossible */

int (* tab[ ])():
/*   oui :
tableau(pointeur(frenvoie(int))) */
```

Type pointeur sur fonction

Pour pouvoir appliquer un constructeur de type à un **type fonction**, la syntaxe du C impose de construire d'abord un **type pointeur sur fonction**

Déclarations de types en C

```
int (* tab[])(), count;
```

`int`	type de base
`(* tab[])()`	déclarateur
`count`	déclarateur

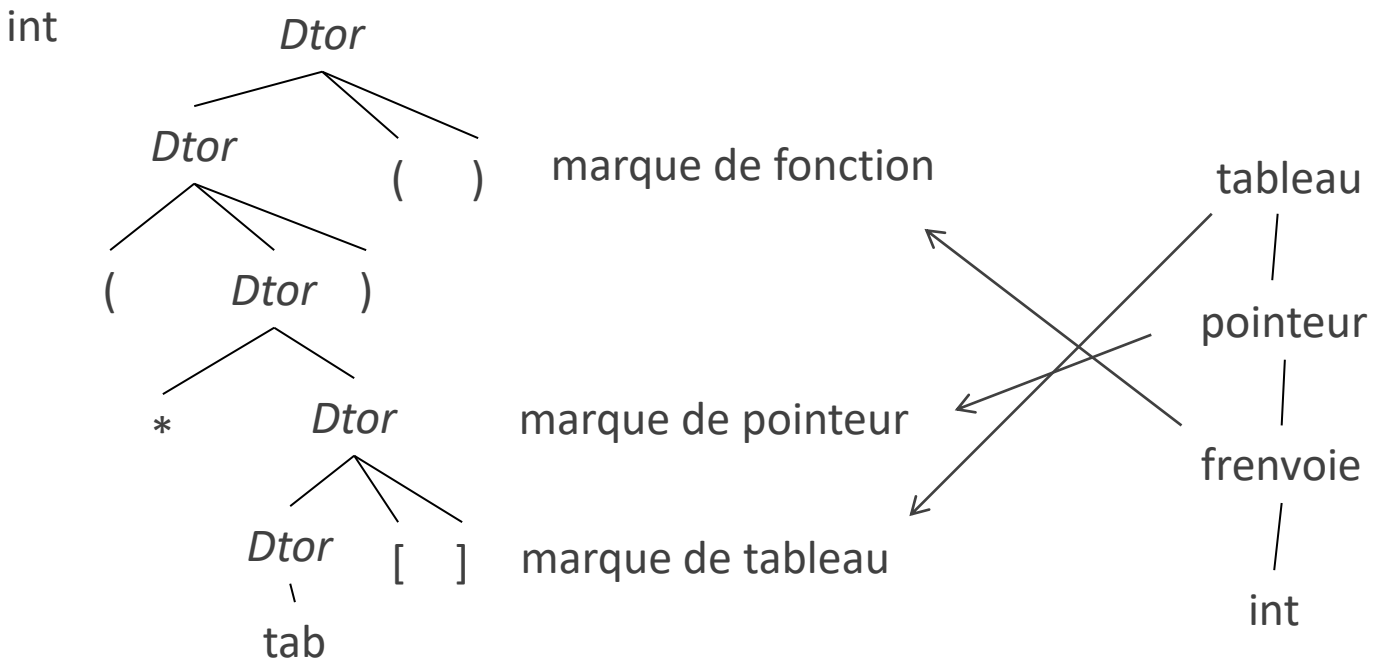
Déclarateur

```
int (* tab[])()
```

`int`	type de base
`(* tab[])()`	déclarateur

La déclaration se fait à l'envers par rapport à l'expression de type

Déclarations de types en C



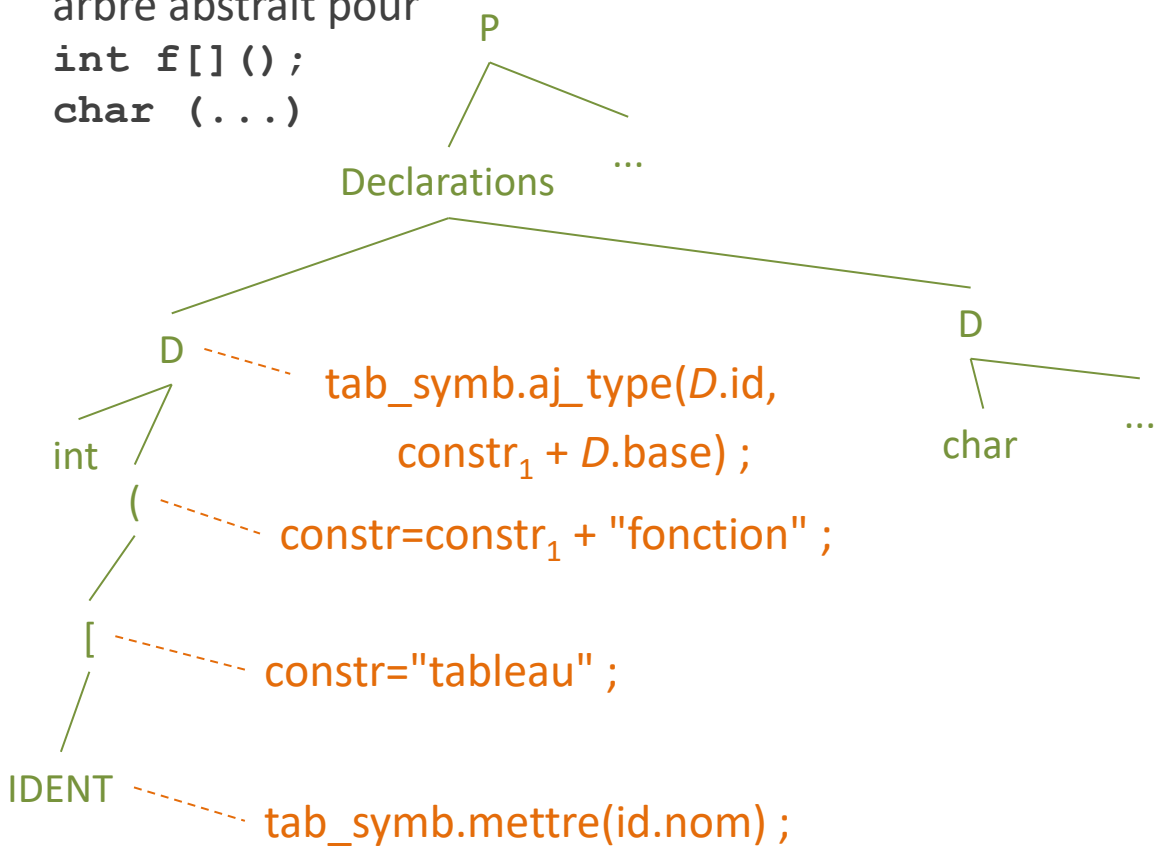
déclarateur

expression de type

Grammaire attribuée

P	--> *D* ; *E*
D	--> *D* ; *D*
D	--> *base Dtor*
base	--> **char**
base	--> **int**
Dtor	--> *Dtor* **[ ]**
Dtor	--> *Dtor* **( )**
Dtor	--> **id**
Dtor	--> ***** *Dtor*
Dtor	--> **(** *Dtor* **)**

arbre abstrait pour
```
int f[]();
char (...)
```



"+" représente la concaténation de chaines de caractères

Grammaire attribuée

Pour déclarer plusieurs variables avec le même type de base

D --> *base liste*

liste --> *Dtor*

liste --> *liste* , *Dtor*

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Conversions de types

Traduction en code intermédiaire :

tmp := intToReal i

tmp := real+ x tmp

où **real+** désigne l'addition des réels

L'expression **x + i** où **x** est un réel et **i** un entier nécessite une conversion

L'addition des réels et l'addition des entiers sont des opérations distinctes car la représentation des données est distincte

Conversions implicites (*coercion*)

Sans risque de perte d'informations

Faites par le compilateur

Conversions explicites (*casting*)

Exemples en C : **(float) 10**

(int) PI

Conversions implicites

E --> **num**	*E*.type := entier ;
E --> **num.num**	*E*.type := réel ;
E --> **id**	*E*.type := lookup(**id**.entry) ;
E --> *E* **op** *E*	*E*.type := if (E_1.type=entier and E_2.type=entier) entier ; else if (E_1.type=entier and E_2.type=réel) réel ; etc.

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Surcharge

Cas où un même symbole représente plusieurs opérateurs

L'opérateur à appliquer est choisi en fonction du contexte

Exemple : + désigne

- l'addition des entiers
- l'addition des matrices
- l'addition d'un entier à un caractère

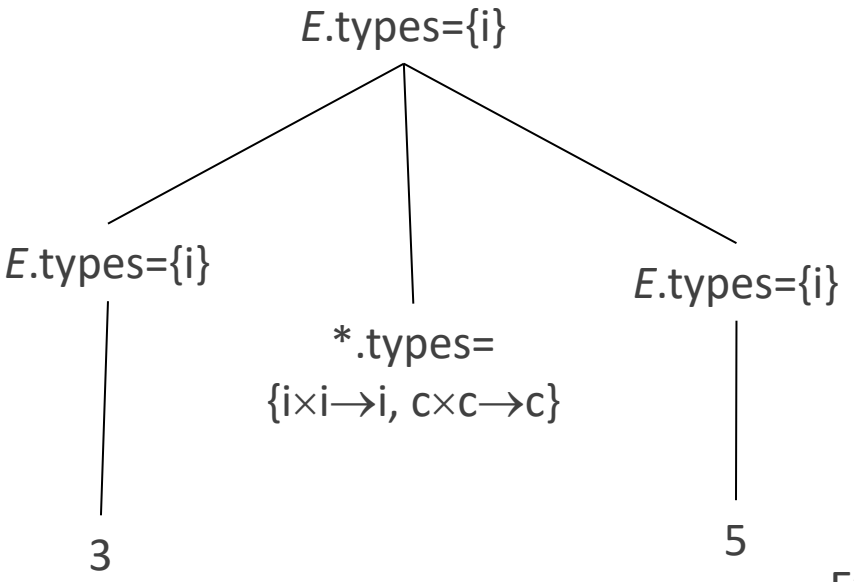
De même pour les fonctions

Grammaire attribuée pour calculer les types possibles d'une expression

E --> **id** *E*.types := lookup(**id**.entry)

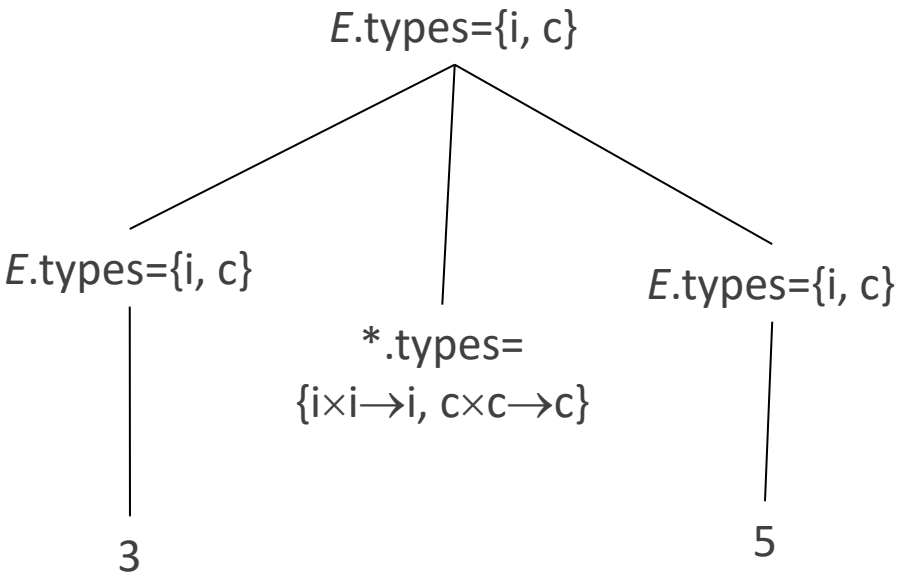
E --> *E* **(** *E* **)** *E*.types := { *t* | il existe un *s* dans E_2.types tel que $s \rightarrow t$ est dans E_1.types }

Surcharge des opérateurs



Exemple : l'opérateur * peut prendre des opérandes entiers ou complexes et le résultat peut être entier ou complexe

Surcharge des opérateurs



Surcharge des méthodes en Java

```
class A {
      int a, b;
      A(int a, int b){this.a = a; this.b = b; }
 }
 class B{
      private A couple;
      B(int a, int b){couple = new A(a,b); }
      B(A couple){this.couple = new A(couple.a, couple.b); }
      A getA( ){ return couple;}
      void ajouter(int inc){getA().a += inc; getA().b += inc; }
      void ajouter(B couple){
            this.getA().a += couple.getA().a;
            this.getA().b += couple.getA().b; }
 }
```

Surcharge des méthodes en Java

```
void ajouter(int inc){
void ajouter(B couple){
```

Définitions de plusieurs méthodes de même nom dans la même classe ou interface (ou par héritage)

Les méthodes surchargées diffèrent par leur signature

Signature d'une méthode : nombre, ordre et type des paramètres formels

Surcharge des méthodes en Java

```
B c1 = new B(3, 5);
B c2;
System.out.println("c1=" + c1);
c1.ajouter(5);
System.out.println("c1=" + c1);
c2 = new B(c1.getA());
c2.ajouter(c1);
```

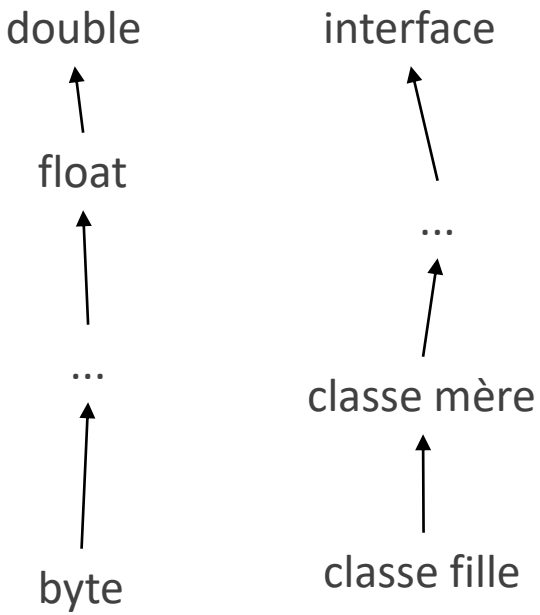
Sélection de la méthode pour un appel

Utilisation du type des paramètres effectifs

S'ils correspondent exactement à la signature d'une des méthodes, elle est sélectionnée

Le type de retour n'intervient pas

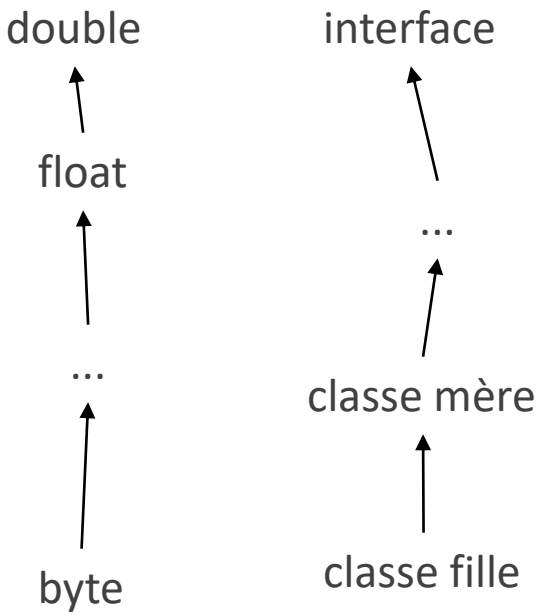
Sélection de méthode surchargée en Java



S'il n'y a pas de correspondance exacte avec une des signatures concurrentes, on utilise les conversions ascendantes

On sélectionne les signatures pour lesquelles on peut passer des paramètres effectifs à la signature en n'utilisant que des conversions ascendantes

Sélection de méthode surchargée



On compare les signatures sélectionnées

Une signature est plus spécifique qu'une autre si on peut passer de la première à la deuxième en n'utilisant que des conversions ascendantes

On recense les signatures maximalement spécifiques

S'il n'y en a qu'une, on la sélectionne

S'il y en a plusieurs, erreur de syntaxe

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Fonctions génériques

Fonctions dont les paramètres ou la valeur de retour sont de type non précisé

Exemples

L'opérateur **&** du C : si *E* est de type t, alors **&** *E* est de type pointer(t)

Les fonctions génériques du langage ADA

Certaines fonctions en langage CAML

Exemple non générique

```
typedef struct cell { int info ; struct cell * link ; } cell ;
int length(cell * list) {
        int len = 0 ;
        while (list) { len ++ ; list = list -> link ; }
        return len ; }
```

Calcul de la longueur d'une liste

On est obligé de connaître le type des éléments de la liste pour déclarer le type cell et la fonction

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Exemple générique

En ML

```
fun length(list) =
        if null(list) then 0
        else length(tl(list)) + 1 ;
```

null() et tl() sont prédéfinies

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Un langage avec généricité

P --> *D* **;** *E*

D --> *D* **;** *D* | **id :** *Q*

Q --> ∀ **typeVar .** *Q* | *T*

T --> **basicType**

| *unaryConstructor* **(** *T* **)**

| **typeVar**

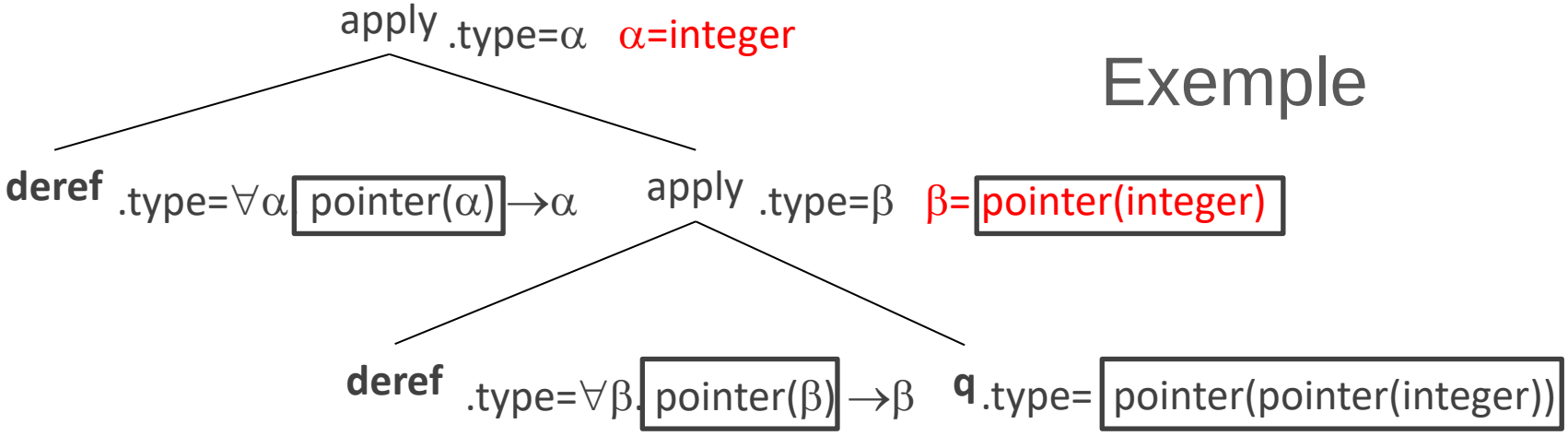
| $T \rightarrow T$

| $T \times T$

| **(** *T* **)**

E --> *E* **(** *E* **)** | *E* **,** *E* | **id**

Exemple



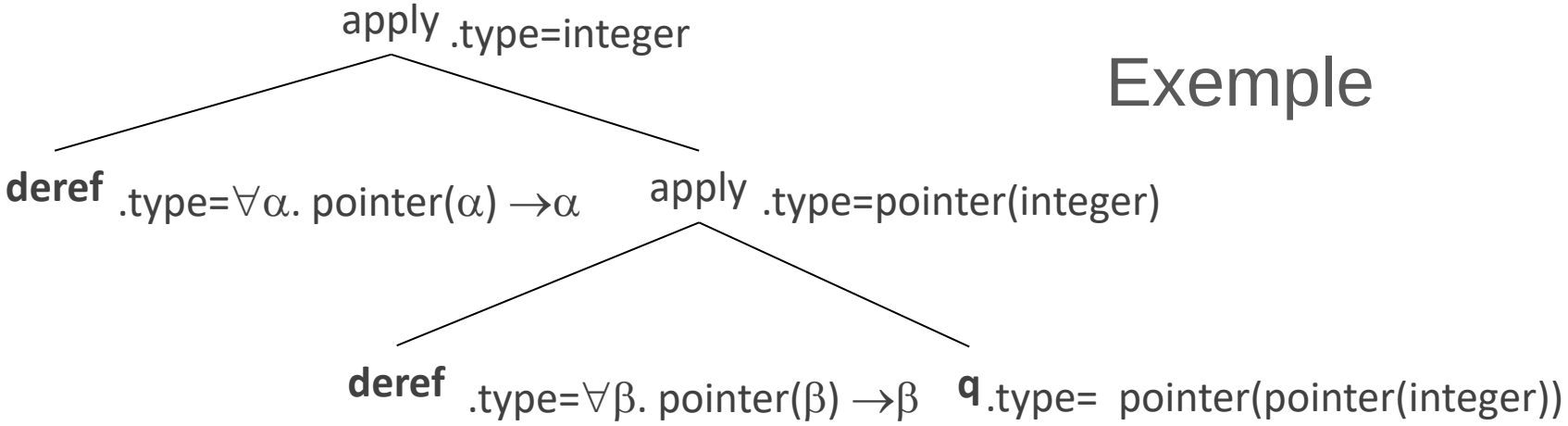
Arbre syntaxique de l'expression deref(deref(q))

Déclaration d'un pointeur

q : pointer(pointer(integer)) ;

Déclaration de la fonction de déréférencement

deref : $\forall\ \alpha\ .\ \text{pointer}(\alpha) \rightarrow \alpha$;



Exemple

Des occurrences distinctes d'une fonction générique n'ont pas nécessairement le même type

pointer(pointer(integer))→pointer(integer)

pointer(integer)→integer

Le calcul des types des variables fait appel à un algorithme d'**unification**