Langages

Traducteurs

Sémantique

Equipe pédagogique

Cours

- Jean-Claude Fernandez
  (VERIMAG av. de Vignate, GIERES 04 56 52 03 79)
- Yassine Lakhnech (VERIMAG)

TD

- Christian Ene (VERIMAG)
- Fabienne Lagnier (VERIMAG)
- Yassine Lakhnech (VERIMAG)
- Laurent Mounier (VERIMAG)

Les langages

Les langages informatiques

Leur description (syntaxe, sémantique)

L'exécution des programmes écrits dans un langage

Enseignements

Compilation : étude d'un compilateur "générique"

Sémantique des langages de programmation : formalismes pour la description de la sémantique d'un langage

Langages

- Langages impératifs e.g. Fortran, Algol-xx, Pascal, C, Ada, Java,... (Concepts : désignation d'emplacements mémoire, Structure de contrôle (PC) expression (PO))
- Langages fonctionnels e.g. ML, CAML, LISP (Concepts : réduction, évaluation de fonctions)
- Langages orienté objet Ada, Java,... (Concepts : objets, classes, types, héritage, polymorphisme, etc)
- Langages logiques Prolog (Concepts : Résolution dans le calcul des prédicats)
- Langages spécialisés VHDL, CSH, HTML, ...
- ...

Langages impératifs

- Structures de données :
  - Types de base (entiers, caractère, etc)
  - Types construits (énumération, produit, union, fonction ou tableaux)
- Expressions
- Structures de contrôle :
  - Affectation
  - itération, conditionnelle, séquence, structure de bloc, appel de fonctions, paramètres

Description des langages

Lexique Les mots du langage (le lexique) sont décrits par des expressions régulières

Syntaxe La syntaxe est décrite à l'aide de grammaire hors-contexte

Sémantique statique (type) Règles d'inférences ou grammaires attribuées

Sémantique dynamique Règles, ensemble de fonctions, ensemble d'équations

Exemple : un langage d'expressions

Lexique

- Identificateurs : `lettre.(lettre + chiffre)*`
- Constantes entières : `chiffre`+
- opérateur ‘+’
- opérateur ‘*’
- parenthèse ouvrante ‘(’
- parenthèse fermante ‘)’

Syntaxe

```
E : E ‘+’ T
E : T

T : T ‘*’ F
T : F

F : ID
F : NUM
F : ‘(’E‘)’
```


Sémantique : vérification de types

Γ décrit une fonction partielle des noms vers les types.

Exemples de règle :

$$\frac{\Gamma \vdash \mathrm{e}_1 : Int \quad \Gamma \vdash e_2 : Int}{\Gamma \vdash e_1 + e_2 : Int}$$

$$\frac{}{\Gamma \vdash \mathrm{NUM} : Int}$$

Sémantique dynamique

On décrit une fonction $\mathcal{A}$ qui interprète les expressions.

$$\begin{aligned}
\mathcal{N}(n_k \cdots n_0) &= \sum_{i=0}^{k} n_i * 10^i \\
\mathcal{A}(x)(\sigma) &= \sigma(x) \\
\mathcal{A}(n)(\sigma) &= \mathcal{N}(n) \\
\mathcal{A}(a_1 + a_2)(\sigma) &= \mathcal{A}(a_1)(\sigma) +_I \mathcal{A}(a_2)(\sigma)
\end{aligned}$$

Sémantique dynamique : règles

$$\frac{}{(n,\sigma) \mapsto \mathcal{N}(n)}$$

$$\frac{x \in \texttt{Dom}(\sigma)}{(x,\sigma) \mapsto \sigma(x)}$$

$$\frac{(e_1,\sigma) \mapsto (e_1',\sigma)}{(e_1 + e_2,\sigma) \mapsto (e_1' + e_2,\sigma)}$$

$$\frac{(e_2,\sigma) \mapsto (e_2',\sigma)}{(v + e_2,\sigma) \mapsto (v + e_2',\sigma)}$$

Langages

Traducteurs

Sémantique

Les traducteurs

– Les compilateurs

– Les interpréteurs

– ... et encore les préprocesseurs, les assembleurs, les éditeurs de liens

Un langage peut être interprété ou compilé : par exemple, Java

Quelques questions sur la compilation

1. Interactions entre Compilateurs, Langages et Architectures
2. Interprétation/Compilation
3. Architecture d'un compilateur
4. Quelques perspectives
5. Bibliographie

Architecture d'un compilateur

Analyse lexicale

Analyse syntaxique

Analyse sémantique

Génération de code intermédiaire

Optimisation de code

Génération de code

Analyse lexicale

Entrées : chaîne de caractères

Sorties : séquence de classes lexicales

Analyse la plus longue chaîne correspondant à une classe lexicale

Retourne à l'analyseur syntaxique

- La classe lexicale (le token)
- L'élément de cette classe (le lexème)

Analyse lexicale exemple :

Pour la chaîne de caractères

```
x1 + x2 +
x3
```

l'analyseur lexical
- reconnaît "`x1`" qui est un identificateur,
- il insère `x1` dans la table des symboles (en position 0)
- il retourne `(ID,0)` à l'analyseur syntaxique.

Il reste la chaîne à analyser.

```
 + x2 +
x3
```


Analyse lexicale exemple :

De la même façcon, il reconnaît et analyse `+`, puis `x2` qu'il insère dans la table des symboles en retournant `(ID,1)` etc.

A noter que les caractères séparateurs :
- ' '
- '\t'
- '\n'

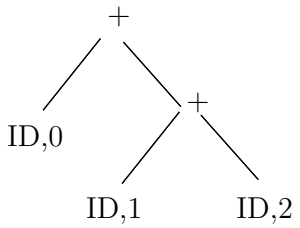
sont ignorés.

Analyse syntaxique

Entrées Un token, un attribut

Sortie un arbre abstrait, la table des symboles modifiée.

Analyse syntaxique exemple :



Analyseur syntaxique : fonctionnement

Le langage engendré par une grammaire hors-contexte $G = (V_T, V_N, S, P)$ est

$\mathcal{L}(G) = \{w \mid w \in V_T \text{ tel que } S \longrightarrow^* w\}$

où $\longrightarrow^*$ est la fermeture réflexive et transitive de $\longrightarrow$.

Exemple :

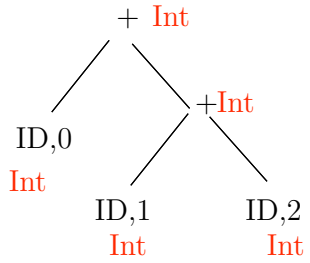
$$\texttt{E} \longrightarrow \texttt{E + T} \longrightarrow \texttt{E + T + T} \longrightarrow \texttt{T + T + T}$$
$$\longrightarrow \texttt{F + T + T} \longrightarrow \texttt{ID + T + T} \longrightarrow \texttt{ID + F + T}$$
$$\longrightarrow \texttt{ID + ID + T} \longrightarrow \texttt{ID + ID + F} \longrightarrow \texttt{ID + ID+ ID}$$

Remarque : dérivation gauche

Analyse sémantique : type

Entrée : arbre abstrait

Sortie : arbre abstrait enrichi par le calcul des types.



Considération sur le calcul des types : aspects statiques

+	Int	Réel
Int	Int	?
Réel	?	Réel

? peut être soit réel soit erreur

Considération sur le calcul des types : aspects dynamiques

Définition `int T[12]`

Utilisation `T[i]` ?

i=13 non valide

i=11 valide

Type : polymorphisme

```
let rec longueur = function
      [] -> 0
| (x::l) -> longueur(x)+1
  ;;
```

Une fonction qui calcule la longueur d'une liste indépendamment du type de ses éléments.

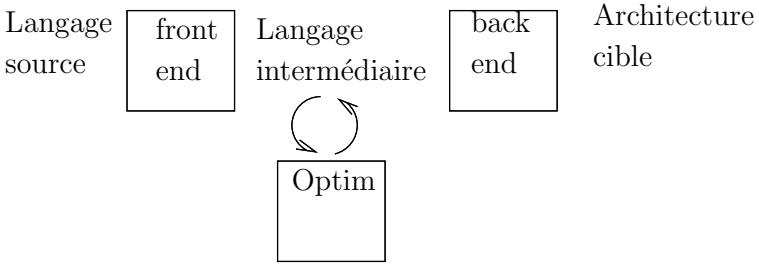
Génération de code intermédiaire

Un langage indépendant du langage source et de la machine cible

Exemple : Code à trois adresses :

- `x := y op x`
- `x := op y`
- `x := y`
- `goto L`
- `si x oprel y goto L`
- `*x := y, x := *y, x := &y`
- `x[i] := y, x := y[i]`

Génération de code intermédiaire



front end : analyse lexicale, syntaxique et sémantique.

back end : génération de code

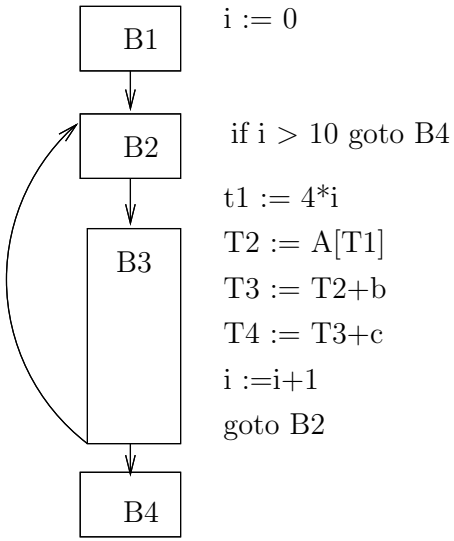
Génération de code intermédiaire exemple

Considérons le programme :

```
for (i=0 ; i < 10 ; i ++)
   A[i] := A[i] + b + c
```

La séquence de quadruplets (taille d'un entier = 4)

Génération de code intermédiaire exemple



i := 0

if i > 10 goto B4

t1 := 4*i
T2 := A[T1]
T3 := T2+b
T4 := T3+c
i :=i+1
goto B2

Optimisation de code : critères

Espace mémoire

Temps d'exécution

Consommation d'énergie

Optimisation de code : exemples

- Trouver et propager des valeurs constantes,
- déplacer du code à un endroit où il sera moins exécuté,
- trouver et supprimer des calculs inutiles
- supprimer du code inutile ou inaccessible,
- traduire des opérations complexes en opérations plus simples

Optimisation de code : déplacement de code

```
for (i=0;i< N;i++)
  for(j=0;j<N;j++)
    ...T[i,j]...
```


Optimisation de code : déplacement de code

On génère du code pour l'accès aux éléments du tableaux.

$a_{imp} + i * e_1 + j * e_2$

```
i := 0                    i:= 0
if (i> N) goto L1         t1 := taille*i
j := 0                    if (i> N) goto L1
if (j> N) goto L2         j := 0
 ....                     if (j> N) goto L2
t1 := taille*i                 ....
t2 := N*taille*j          t2 := N*taille*j
t3 := t1+t2               t3 := t1+t2
 ....                           ....
 T[t3]                     T[3]
```


Exemple

Code initial

```
a := x ** 2
b := 3
c := x
d := c * c
e := b * 2
f := a + d
g := e * f
```

Optimisation algébrique

```
a := x ** 2      a := x * x
b := 3           b := 3
c := x           c := x
d := c * c       d := c * c
e := b * 2       e := b << 1
f := a + d       f := a + d
g := e * f       g := e * f
```


Propagation des copies

```
a := x * x       a := x * x
b := 3           b := 3
c := x           c := x
d := c * c       d := x * x
e := b << 1      e := 3 << 1
f := a + d       f := a + d
g := e * f       g := e * f
```


Constant folding

```
a := x * x       a := x * x
b := 3           b := 3
c := x           c := x
d := x * x       d := x * x
e := 3 << 1      e := 6
f := a + d       f := a + d
g := e * f       g := e * f
```


Elimination des sous expressions communes

```
a := x * x       a := x * x
b := 3           b := 3
c := x           c := x
d := x * x       d := a
e := 6           e := 6
f := a + d       f := a + d
g := e * f       g := e * f
```

Propagation des copies

```
a := x * x        a := x * x
b := 3            b := 3
c := x            c := x
d := a            d := a
e := 6            e := 6
f := a + d        f := a + a
g := e * f        g := 6 * f
```


Elimination de code mort

```
a := x * x        a := x * x
b := 3
c := x
d := a
e := 6
f := a + a        f := a + a
g := 6 * f        g := 6 * f
```


Optimisation de code

Calcul de propriétés dans un espace, par calcul de point fixe de fonctions monotones induites par le programme

Transformation de programme Modification du programme en respectant la sémantique

Génération de code

(le back end)

Choix des instructions pour chaque opération du langage intermédiaire, on choisit une séquence d'instructions de la machine cible

Ordonnancement des instructions (instructions scheduling, software pipelining)

Allocation de registres

Langages

Traducteurs

Sémantique

Style de sémantique

Sémantique opérationnelle

Sémantique axiomatique

Sémantique dénotationnelle et calcul approché de propriétés

Langages considérés

Impératif

Fonctionnel

Agenda prévisionnel

- Mercredi 29/09C : JCF 3H de cours intro +analyse lexicale
- Mercredi 06/10 : JCF 3H de cours Analyse syntaxique
- Mercredi 13/10 : JCF 1H30 de cours Analyse syntaxique YL 1H30 de cours sémantique opérationnelle
- Mercredi 20/10 YL 3H de cours sémantique opérationnelle
- Mercredi 03/11 YL 3H de cours bloc+procédures
- Mercredi 10/11 JCF 3H typage + table + code
- Mercredi 17/11 JCF 3H + code
- Mercredi 24/11 JCF 1H30 optimsation de code YL 1H30 Hoare+denota
- Mercredi 01/12 YL 3H Hoare+dénotationnelle
- Mercredi 08/12 YL 3H Hoare+dénotationnelle

Références

[1] A. Aho, R. Sethi and J. Ullman Compilateurs : Principes, techniques et outils InterEditions, 1989

[2] H. R. Nielson and F. Nielson. Semantics with Applications : A Formal Introduction. Wiley Professional Computing, (240 pages, ISBN 0 471 92980 8), Wiley, 1992

[3] W. Waite and G. Goos. Compiler Construction Springer Verlag, 1984

[4] R. Wilhelm and D. Maurer. Théorie, construction, génération Masson 1994