Notes de cours

GIF 340 : ÉLÉMENTS DE COMPILATION

Chapitre 5

Introduction à l 'analyse syntaxique

Ahmed KHOUMSI

PLAN

But de l'analyse syntaxique

Types d'erreurs dans un langage de programmation

Traitement des erreurs syntaxiques

Stratégies de récupération sur erreur
- récupération en mode panique
- récupération au niveau du syntagme
- récupération par production d'erreurs
- récupération par correction globale

Contraintes sur les grammaires pour spécifier la syntaxe
- exemple de contrainte : grammaire non ambiguë
- exemple de contrainte : grammaire non récursive à gauche

Méthodes d'analyse :
- analyse descendante
- analyse ascendante

But d'un analyseur syntaxique

Travail essentiel effectué :
- regroupe les unités lexicales en structures grammaticales
- vérifie si la syntaxe est correcte
- construit un AST représentant la structure hiérarchique sur le flot des unités lexicales

Résultat essentiel en sortie :
- signale si la syntaxe est correcte
- Un AST

Types d'erreurs

Erreurs lexicales (détectées par le scanner)
tout ce qui n'est pas conforme à la grammaire (régulière) définissant le vocabulaire
Exemple : écriture erronée d'une unité lexicale telle que
un identificateur, un mot clé, un opérateur ...

Erreurs syntaxiques (détectées par le parser)
tout ce qui n'est pas conforme à la grammaire (hors-contexte) définissant la syntaxe
Exemples :
- expression arithmétique mal parenthésée
- mot clé if sans le then correspondant
- affectation sans partie droite a := ;
- affectation sans partie gauche := b;

C'est l'objet de cette partie du cours

Types d'erreurs (suite)

Erreurs sémantiques (détectées par l'analyseur sémantique)

Généralement, la sémantique d'un langage est spécifiée d'une manière informelle car une spécification formelle serait :

- complexe à réaliser
- et surtout trop complexe à analyser

Ceci est dû au fait que la grammaire nécessaire pour spécifier la sémantique d'un langage n'est pas hors-contexte

On détermine alors d'une manière informelle, un certain nombre d'erreurs sémantiques que l'on désire détecter

Exemples :

- opérateur appliqué à un opérande incompatible
- utilisation d'une procédure incompatible avec sa définition (déclaration) (sur nombre ou type d'arguments …)

Erreurs logiques

Exemple : un appel récursif infini

Traitement des erreurs syntaxiques

En général, plus grande part du traitement des erreurs est faite par le parser. Car :

- une grande partie des erreurs effectuées sont par nature syntaxiques
- il existe des méthodes performantes pour détecter les erreurs syntaxiques

Gestionnaire des erreurs dans un Parser

- indiquer présence d'erreurs de façon claire et précise (lieu, cause …)
- traiter chaque erreur rapidement
- ne pas ralentir de façon significative la compilation d'un programme correct

Dans la grande majorité des cas, les buts du gestionnaire sont facilement atteints
Car une grande partie des erreurs mettent en jeu une seule unité lexicale

Exemple : Une étude sur des programmes Pascal écrits par des étudiants montre que la plus grande partie des erreurs est due à l'usage incorrect du point-virgule

Buts du gestionnaire des erreurs deviennent difficiles à atteindre lorsque, par exemple, une erreur se produit longtemps avant sa détection.
Exemple : oubli de "}" en langage C

Traitement des erreurs syntaxiques (suite)

Il existe des méthodes d'analyse syntaxique permettant de détecter une erreur dès que possible
Exemples : Méthodes LL et méthodes LR (présentées ultérieurement)

Après détection d'une erreur, le gestionnaire des erreurs doit, si possible :

- indiquer emplacement (dans le programme source) où erreur produite
  exemple : visualiser ligne erronée, avec marqueur désignant position de l'erreur

- donner la cause de l'erreur
  exemples : - parenthèse fermante en plus
  - point-virgule manquant

Nous allons maintenant présenter les stratégies de récupération sur erreur qui spécifient ce qu'il faut faire après détection d'erreur

Contraintes sur grammaires

Vocabulaire d'un langage est spécifié par une grammaire régulière

Syntaxe est spécifiée par une grammaire :

- hors-contexte (non contextuelle)
- qui doit satisfaire des contraintes supplémentaires qui dépendent de la méthode d'analyse syntaxique adoptée

Premier exemple de contrainte : grammaire non ambiguë

Une grammaire qui produit plus d'un AST pour une phrase donnée est dite ambiguë

Exemple : Condition définie par règles

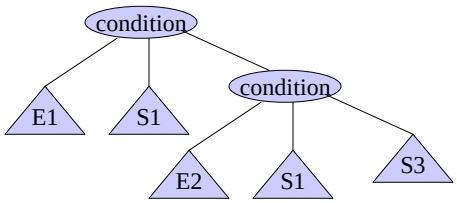
Inst ⟶ si *Expr* alors *Inst*

Inst ⟶ si *Expr* alors *Inst* sinon *Inst*

Exemple de phrase non ambiguë :

si *E1* alors *S1* sinon si *E2* alors *S2* sinon *S3*

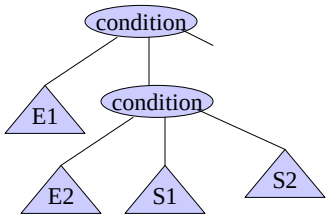
L'AST correspondant peut être schématisé comme suit



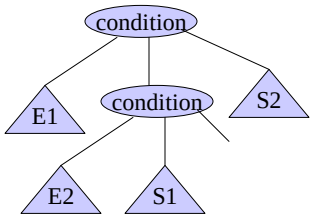
Premier exemple de contrainte : grammaire non ambiguë (suite)

Exemple de phrase ambiguë : si *E1* alors si *E2* alors *S1* sinon *S2*

Premier AST correspondant
(sinon associé au second si)



Second AST correspondant
(sinon associé au premier si)



Premier exemple de contrainte : grammaire non ambiguë (suite)

L'ambiguïté peut être résolue en modifiant la grammaire comme suit

Inst ⟶ *Inst1* | *Inst2*

Inst1 ⟶ si *Expr* alors *Inst1* sinon *Inst1*

Inst2 ⟶ si *Expr* alors *Inst* | si *Expr* alors *Inst1* sinon *Inst2*

Avec ces nouvelles règles, seul le premier AST est produit

Certaines méthodes d'analyse syntaxique acceptent des grammaires ambiguës. Ces méthodes utilisent des règles pour lever l'ambiguïté en ne conservant qu'un AST unique pour chaque phrase (c'est le cas du parser généré par YACC)

Second exemple de contrainte : grammaire non récursive à gauche

Une grammaire est récursive à gauche si elle possède une ou plusieurs règles de la forme $A \longrightarrow A\alpha$

où : A est un non-terminal

α est une chaîne quelconque de symboles (terminaux ou non-terminaux) ne commençant pas par A

Certaines (mais pas toutes) méthodes d'analyse syntaxique ne supportent pas les grammaires récursives à gauches

Suppression de la récursivité à gauche comme suit :

Soit la règle $A \longrightarrow A\alpha 1 \mid A\alpha 2 \mid \ldots \mid A\alpha n \mid \beta 1 \mid \beta 2 \mid \ldots \mid \beta m$

où : A est un non-terminal

αi et βj sont des chaînes de symboles (terminaux ou non-terminaux) ne commençant pas par A

Cette règle peut être remplacée par les deux règles suivantes

$A \longrightarrow \beta 1A' \mid \beta 2A' \mid \ldots \mid \beta mA'$

$A' \longrightarrow \varepsilon \mid \alpha 1A' \mid \alpha 2A' \mid \ldots \mid \alpha nA'$

Second exemple de contrainte : grammaire non récursive à gauche (suite)

Exemple de suppression de la récursivité à gauche

$A \longrightarrow A\,a \mid b$ où : A est un non-terminal
a et b sont des terminaux

Le langage engendré par cette règle est défini par l'expression régulière ba^*

La règle ci-dessus peut être remplacée par

$A \longrightarrow bA'$

$A' \longrightarrow aA' \mid \varepsilon$

Pour comprendre la nécessité de la non-récursivité (pour certaines méthodes), considérons un analyseur syntaxique qui, pour analyser une phrase donnée :

- part du symbole de départ
- parcourt la phrase à analyser de gauche à droite
- décide, à la lecture de chaque symbole, de la dérivation à appliquer

Si la phrase à analyser peut être obtenue après une séquence de dérivation, alors elle est jugée correcte. Sinon, on ne peut rien conclure.

Interprétation intuitive de la nécessité de la non-récursivité à gauche

Considérons, par exemple, la phrase baa à analyser

Analyse en utilisant les règles $A \longrightarrow A\,a$ et $A \longrightarrow b$

Premier caractère lu est b

Le parser ne peut pas décider laquelle des deux règles appliquer car il ne sait pas ce qui vient après b

Pour dériver la phrase à analyser, il faudrait appliquer la première règle deux fois, et ensuite la seconde règle

Analyse en utilisant les règles $A \longrightarrow bA'$ $A' \longrightarrow aA'$ et $A' \longrightarrow \varepsilon$

- Symbole de départ est A
- Première règle appliquée une fois au début, sur lecture de b
  Le symbole courant est bA'
- La seconde règle est appliquée à chaque lecture de a, donc deux fois
  À sa première application, le symbole courant devient baA'
  À sa seconde application, le symbole courant devient $baaA'$
- La troisième règle est appliquée à la lecture de fin de chaîne
  Le symbole courant devient baa

La phrase est donc correcte

Méthodes d'analyse

Soit une grammaire G définie par : Vt, Vn, S et P

Soit une phrase p constituée de symbole terminaux

Le but est alors de déterminer si on peut dériver p à partir de S

Autrement dit, il s'agit de déterminer s'il existe un arbre syntaxique correspondant à la phrase p

Deux approches sont généralement utilisées :

- analyse descendante
- analyse ascendante

Analyse descendante

- On commence par le symbole de départ S
- On applique différentes règles de l'ensemble P

Si on arrive à dériver p à partir de S alors p est syntaxiquement correcte

Autrement dit, il s'agit de construire l'arbre syntaxique :
- en commençant par la racine
- en descendant vers les feuilles

Exemple :

Vt = {*id*, *, +}

Vn = {*E, T, F*}

S = *E*

P = { $E \longrightarrow T$, $E \longrightarrow E+T$, $T \longrightarrow F$, $T \longrightarrow T*F$, $F \longrightarrow$ id }

La phrase p à analyser est *id + id * id*

Une séquence de dérivations permettant d'obtenir p est la suivante

$E \Longrightarrow E + T \Longrightarrow T + T \Longrightarrow F + T \Longrightarrow$ id $+ T \Longrightarrow id + T * F$
$\Longrightarrow id + F * F \Longrightarrow id + id * F \Longrightarrow id + id + id$

Analyse ascendante

On commence à partir de p et on essaye de remonter, par réductions (règles appliquées à l'envers) successives, au symbole de départ S

Si on arrive à obtenir S, alors p est correcte (syntaxiquement)
Autrement dit, il s'agit de construire l'arbre syntaxique :
- en commençant par les feuilles
- en remontant vers la racine

Exemple : on reprend le même exemple précédent

Voici une séquence de réductions permettant d'obtenir le symbole de départ *E*

id + id * id ⟸ *F* + id * id ⟸ *T* + id * id ⟸ *E* + id * id
⟸ *E* + *F* * id ⟸ *E* + *T* * id ⟸ *E* + *T* * *F* ⟸ *E* + *T* ⟸ *E*