Analyse syntaxique

intro

- La structure grammaticale est l'ensemble des règles que l'on doit appliquer à partir de l'axiome.
- Cette structure est représentée par un arbre syntaxique.
- L'arbre syntaxique complet contient toutes les informations qui sont apportées du texte analysé.
- Il peut exister plusieurs arbres syntaxiques associés au même texte si la grammaire est ambigüe.
- L'analyse est ascendante ou descendante suivant que l'on crée l'arbre syntaxique en partant des feuilles ou de la racine.

analyse

- Analyse ascendante : on essaye de réduire la phrase analysée par application de règles pour retrouver la racine.
  - On examine la suite de symboles terminaux pour tenter de synthétiser une notion (un symbole non terminal).
  - Puis on recommence, en examinant les symboles terminaux non synthétisés et les notions déjà synthétisées jusqu'à atteindre l'axiome de la grammaire ou jusqu'à aboutir à une erreur.
- Analyse descendante :
  - On construit l'arbre syntaxique à partir de la racine. Pour essayer une règle, on parcourt son développement (partie droite) :
    - Si c'est un symbole terminal, on regarde s'il est présent dans la chaîne source
    - Sinon on essaye d'appliquer le procédé pour trouver la règle à appliquer.
    - Lorsqu'un symbole attendu est absent, alors on élimine la règle et on examine une autre règle définissant la même notion.
    - S'il n'en reste plus, on considère que la notion est absente.

dérivations

- Les méthodes ascendantes consistent à retrouver la dérivation la plus à droite par une suite de réductions.
- Les méthodes descendantes consistent à retrouver la dérivation la plus à gauche par une suite de dérivations.

Le retour arrière

- Les méthodes les plus générales prévoient la possibilité de reculer dans l'analyse : défaire des parties de l'arbre syntaxique que l'on vient de construire lorsque l'on s'aperçoit que la méthode employée n'est pas la bonne. Ce processus n'est pas nécessaire lorsque la grammaire possède certaines particularités.
- Si l'on peut affirmer que la reconnaissance d'un caractère ou d'une notion suffit à discriminer une règle, alors on doit savoir à tout moment quelle règle appliquer et donc savoir détecter une erreur sans avoir recours au retour arrière.
- On cherche toujours à supprimer les retours en arrière en remplaçant la grammaire initiale par une grammaire équivalente qui les évite.

Codage de la grammaire

Deux techniques de représentations :

- Codée dans une structure (table, …), l'analyseur la parcourt en suivant les différents liens (les alternants, les successeurs ou encore définitions de notions) : analyseur très simple à réaliser. En effet, c'est un automate à pile dirigé par une table.
- Grammaire incluse dans l'analyseur : l'algorithme de l'analyseur suit la structure de la grammaire, dans ce cas l'algorithme est spécifique à une grammaire

Les symboles de bases

Dans le cas des grammaires non contextuelles, les méthodes précédentes nécessitent l'utilisation de piles ou de la récursivité pour s'appliquer.

Si on se limite aux grammaires régulières, un automate à pile n'est plus nécessaire.

Associations syntaxe / sémantique

- Les méthodes précédentes effectuent l'analyse syntaxique dont le résultat est stocké.
- La méthode consiste à introduire dans le codage de la grammaire des appels de procédures qui réalisent la fabrication de structures de données représentant la structure syntaxique.
- Cette procédure porte un nom : une action sémantique.
- Les actions sémantiques engendrent parfois le code objet.

Limites des méthodes d'analyse classiques

- Les méthodes décrites jusqu'à présent ne permettent pas de décrire le langage naturel.
- Ambiguïté : lorsque l'analyse d'une phrase conduit à au moins 2 arbres syntaxiques différents, la grammaire est ambiguë.
- L'ambiguïté n'est pas un problème pour la compilation.

Comparaisons

- Une méthode avec retour arrière est toujours plus lente mais elle permet de traiter une gamme plus large de grammaires.
- Une méthode tabulée (représentée dans une table ou structure) occupe généralement moins de place mais elle sera plus lente que la méthode programmée.
- Il existe des outils adéquats tels que YACC pour construire des analyseurs descendants

Analyse syntaxique et automates

- Les grammaires contextuelles correspondent aux machines de Turing.
- Les grammaires hors-contexte correspondent aux automates à piles.
- Les grammaires régulières correspondent aux automates sans piles

Algorithme très intuitif pour la transformation d'un langage infixé vers un langage post-fixé

- On empile les parenthèses gauches. Ensuite, on va empiler aussi les opérateurs pour les sortir lorsque leur deuxième opérande sera sorti à la rencontre d'un opérateur identique ou moins prioritaire.
- L'automate a deux états principaux :
  - L'attente d'un opérande
  - L'attente d'un opérateur
  - Un état pour vider la pile à la fin
  - L'état final

Analyse syntaxique descendante

Algorithme général :

```
Fonction analyse(Notion N) : renvoie (booléen Trouvé)
Var posLoc : indice dans le tableau
// R : règle courante, E l'élément courant
DEBUT
   posLoc := posSource ;
   R := 1ère règle (N) ; // règle qui décrit N
   Trouvé := FAUX ;
   TANT QUE (NON Trouvé) ET (R ≠ Vide) FAIRE
      SI essai( R ) ALORS
         Trouvé := VRAI ;
      SINON
         R := suivant( R ) ;
         posSource := posLoc ;
      FIN SI
   FIN TANT QUE
   RETOUR Trouvé ;
FIN
```


Analyse descendante (suite)

```
Fonction essai (Règle R) : renvoie (booléen Bon)
DEBUT
   E := 1er élément de R ;
   Bon := VRAI ;
   TANT QUE Bon ET (E ≠ Vide) FAIRE
      SI E est une notion ALORS
         SI analyse(E) ALORS
            E := suivant(E) ;
         SINON
            Bon := FAUX ;
         FIN SI
      SINON
         SI (E = Source[posSource]) ALORS
            posSource := incrementer(posSource) ;
            E := suivant(E) ;
         SINON
            Bon := FAUX ;
         FIN SI
      FIN SI
   FIN TANT QUE
   RETOUR Bon ;
FIN
```

Analyse descendante

```
Programme Principal :
DEBUT
    posSource := début du texte source ;
    SI analyse(axiome) ALORS
        Ecrire(«  Texte reconnu ! ») ;
    SINON
        Ecrire(« Texte non reconnu ! ») ;
    FIN SI
FIN
```


Descente récursive

- Plutôt que de coder la grammaire dans une structure, on la code directement dans l'algorithme d'analyse.
- On associe une fonction booléenne à chaque notion. Cette fonction essaye successivement les différentes règles qui définissent la notion. Le booléen retourné, dit si la notion a été reconnue ou pas.
- Les grammaires doivent posséder certaines propriétés pour être traitées par cette méthode

Descente récursive

Exemple : E → E + T

```
Fonction E() : renvoie booléen
DEBUT
   posLoc := posSource ;
   SI T() ALORS
      SI (Source[posSource] = '+') ALORS
         posSource := incrémenter(posSource) ;
         SI E() ALORS
            RETOUR VRAI ;
         FIN SI
      FIN SI
   FIN SI
   posSource := posLoc ;
   RETOUR T() ;
FIN
```


Suppression de la récursivité gauche d'une grammaire

A → Bb → Aw

Lorsque l'on a des récursivités indirectes, on se ramène à la récursivité directe en remplaçant B par tous ses développements.

Exemple :

Nom → lettre | lettreEtChiffre

lettreEtChiffre → Nom lettre | Nom Chiffre

ce qui peut s'écrire :

Nom → lettre | Nom lettre | Nom Chiffre

Algorithme

- Factoriser pour se ramener à une règle de la forme suivante : A → a | Ab

  Nom → lettre | Nom alphaNum

- Transformer la règle de façon à supprimer la récursivité à gauche.

  A → a | aB

  B → b | bB

  Exemple

  Nom → lettre | Nom alphanumérique

  Nom → lettre | lettre B

  B → alphanumérique | alphanumérique B

  Alphanumérique → lettre | chiffre