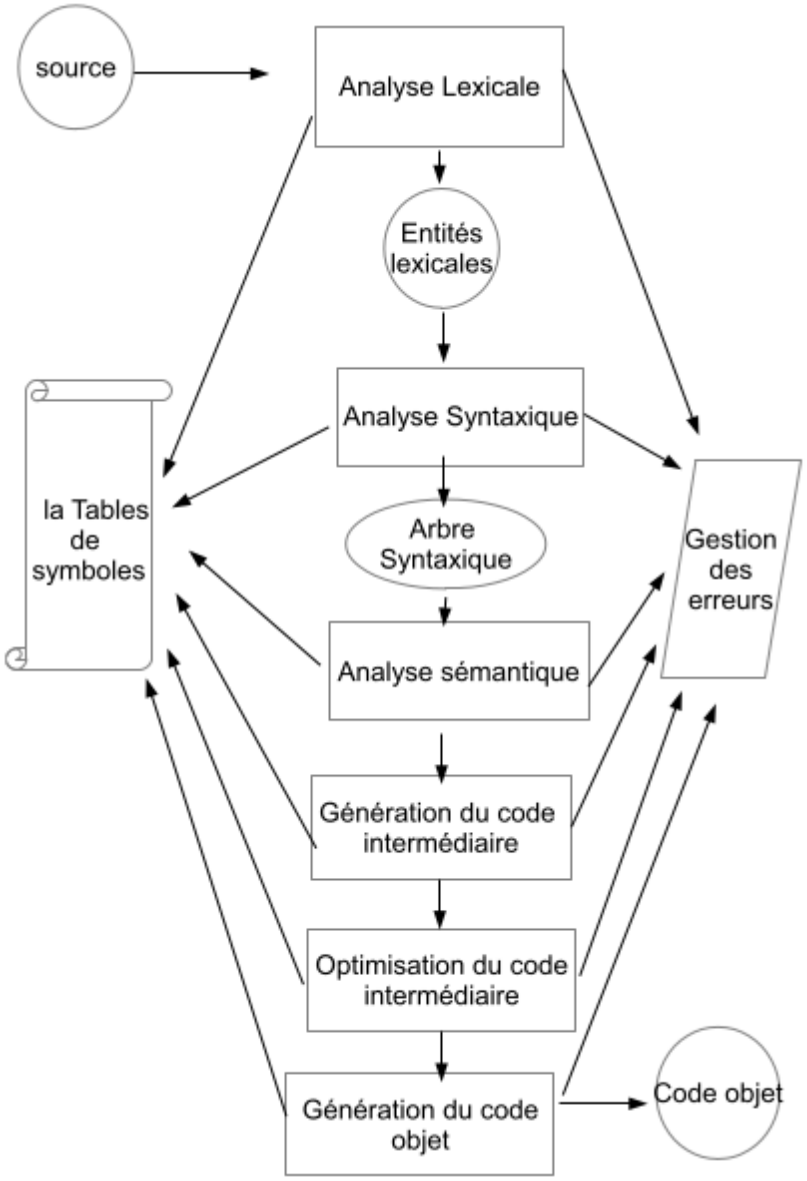
Chapitre I : Introduction à la compilation

Le processus permettant de traduire un programme écrit dans un langage source vers des instructions machines est appelé compilation.
Un compilateur est classiquement structuré en une chaîne de traitements, appelés phases de compilation, au travers desquelles passe le programme, sous des représentations diverses. Chaque phase reçoit de la précédente une représentation particulière du programme et émet vers la suivante une autre représentation, dont on peut imaginer qu'elle est proche du programme cible. De plus, toutes les phases consultent et mettent à jour une table des symboles, ou s'accumulent les données concernant les objets du programme.

Le schéma suivant donne les différentes phases d'un compilateur.

Le processus de compilation est complexe. On décompose cette traduction en traductions plus élémentaires. Tout ceci est idéalement représenté par le schéma suivant :



1. Analyse lexicale

La phase d'analyse lexicale, lit le programme source, caractère par caractère, et tente d'y reconnaître une suite d'éléments lexicaux. Cette phase se base sur les grammaires de type3, des expressions régulières et des automates à états finis. Cette phase peut échouer et doit être capable de signaler des erreurs dites erreurs lexicales.

- Exemple-

2. Analyse syntaxique

Le but de cette analyse est de reconnaître la suite des entités reconnues en lexicale. Elle se base sur les grammaires de type2, des grammaires algébriques et des automates à piles. Le résultat de cette phase est un arbre syntaxique. Cette phase est la mieux formalisée de toute la compilation.

Il existe deux méthodes d'analyse d'un programme syntaxiquement :
- Les méthodes descendantes
- Les méthodes ascendantes

Méthodes Descendantes : Elles consistent à partir de l'axiome de la grammaire et par une série de dérivations , à aboutir au programme à analyser syntaxiquement. Il existe deux types de méthodes d'analyse descendantes, les méthodes non déterministes et les déterministes. Dans notre cours, nous nous intéressons qu'aux méthodes déterministes.

Méthodes déterministes
Elles consistent à emprunter un chemin et aller jusqu'au bout (pas de retour arrière). Plusieurs méthodes existent pour faire une analyse syntaxique descendante déterministe :
- Méthode LL(1)
- Descendante récursive
- Automates

Remarque : Avant de faire une analyse descendante déterministe, s'assurer que la grammaire n'est pas récursive gauche et les propriétés LL(1) vérifiées.

Méthodes Ascendantes
Elles consistent à partir du programme à analyser, faire une série de réductions jusqu'à aboutir à l'axiome. Il existe plusieurs méthodes d'analyse ascendantes :
- LR(1)
- SLR(1)
- LALR(1)

Les compilateurs existants se basent sur l'analyseur LALR(1).
Cette étape peut aussi signaler des erreurs, appelés erreurs syntaxiques.

3. Génération de la forme intermédiaire

Une phase optionnelle dans les compilateurs est la génération de la forme intermédiaire qui peut faciliter l'écriture des programmes. Parmi les formes intermédiaires, nous citons :
- La forme postfixée
- Les quadruplets
- L'arbre abstrait
- …..

Exemple

4. Traduction dirigée par la syntaxe ou analyse sémantique

L'analyse sémantique est une étape étroitement liée à la syntaxique. Cette étape se base sur une grammaire appelée grammaire sémantique, et l'analyse peut être descendante ou ascendante.

Cas descendant
Dans ce cas, si une routine sémantique doit être insérée dans la grammaire, on crée un non-terminal dérivant en Ɛ.

Exemple : si on : $A \rightarrow \alpha\beta$, $A \in N$ et $\alpha, \beta \in (T \cup N)$ et si une routine sémantique doit être insérée entre α , β , alors :
$A \rightarrow \alpha B \beta$, $B \in N$
$B \rightarrow Ɛ$

Exemple if <condition> then <inst1> else <inst2>, et si on doit insérer une routine après condition, inst1 et inst2, on crée des non-terminaux dérivant en Ɛ.

Cas Ascendant
Dans ce cas, si une routine sémantique doit être insérée, on procède au découpage de la grammaire.

Exemple : Si on a : $A \rightarrow \alpha\beta$ et une routine sémantique doit être insérée entre α et β , alors : $A \rightarrow B\beta$ et $B \rightarrow \alpha$

Comme dans les phases précédentes, cette phase peut échouer et doit être capable de signaler des erreurs dites erreurs sémantiques.

5. Optimisation de code

D'une manière générale, le rôle de l'optimisation est de découvrir à la compilation, des informations sur le comportement à l'exécution et d'utiliser cette information pour améliorer le code généré par le compilateur. Historiquement, les premiers compilateurs comprenaient très peu d'optimisation. Avec l'arrivée des nouvelles architectures, l'optimisation devient de plus en plus nécessaire et occupe maintenant une place

importante dans le processus de compilation.

Exemple : while a>0 do begin x:=0, a:=a+1end.

6. Génération de code

Dans cette étape, il faudrait connaître les caractéristiques de la machine cible (registres) codes opérations, ….). Le but de cette étape, est de traduire le programme lexicalement, syntaxiquement et sémantiquement correct en code objet. C'est la dernière étape de la compilation.

Table des symboles

C'est une table contenant les différents objets manipulés par le programme, ainsi que leurs attributs. Elle est vide au début de la compilation et est mise à jour dans les différents étapes de la compilation.

Traitement des erreurs

Des erreurs peuvent survenir dans n'importe quelle étape de la compilation, ainsi un détecteur est un processus de correction peuvent être rajoutées dans chaque étape.