1-1 Les compilateurs sont souvent écrits dans le langage qu'ils implémentent. Identifiez les avantages et les inconvénients de cette technique

1-2 En vous réfétant à l'extrait ci-dessous (Dunod – Compilateurs) donnez des exemples supplémentaires montrant pourquoi une partie avant peut nécessiter de l'information sur la machine cible et une partie arrière de l'information sur le langage source :

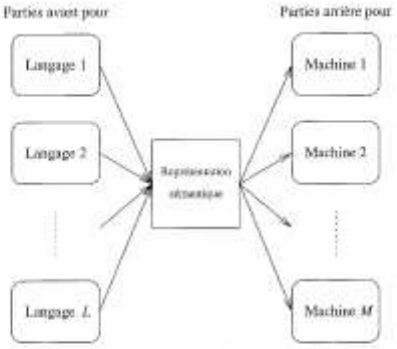


Figure 1.4 Création de compilateurs pour L langages et M machines.

1.1.1 La compilation est très fructueuse

La compilation est une branche très fructueuse de l'informatique. Parmi les raisons de ce succès, on trouve la structuration appropriée du problème, l'utilisation judicieuse de formalismes et l'utilisation d'outils chaque fois que c'est possible.

1.1.1.1 Structuration appropriée du problème

Les compilateurs analysent leurs données, en construisent une représentation sémantique et synthétisent leurs résultats à partir de cette représentation. Ce **paradigme analyse-synthèse** est très puissant et d'application très générale. Un programme qui calcule les longueurs des mots dans un texte, par exemple, peut comporter une partie avant qui analyse le texte et construit une table interne de paires (longueur, fréquence) et une partie arrière qui imprime le contenu de cette table. Pour étendre ce programme, on peut remplacer la partie avant qui analyse le texte par un module qui collecte les longueurs des fichiers dans un système de fichiers ; d'autre part, ou de plus, on peut remplacer la partie arrière par un module qui produit un histogramme plutôt que l'impression du contenu de la table ; nous utilisons ici le terme « module » pour insister sur l'interchangeabilité des parties du programme. Au total, on peut arriver à quatre programmes différents, tous centrés sur la même représentation sémantique, chacun réutilisant une bonne partie du code des autres.

De la même manière, sans la séparation stricte des phases d'analyse et de synthèse, les langages de programmation et la compilation n'en seraient pas là où ils en sont aujourd'hui. Sans elle, chaque nouveau langage nécessiterait un ensemble complètement nouveau de compilateurs pour toutes les machines intéressantes — ou alors il disparaîtrait faute d'être utilisable. Grâce à elle, une nouvelle partie avant pour ce langage est suffisante, et il n'y a plus qu'à la combiner avec les parties arrière existant pour les machines actuelles : pour L langages et M machines, on n'a besoin que de L parties avant et M parties arrière, c'est-à-dire $L + M$ modules, au lieu de $L \times M$ programmes (voir la figure 1.4).

Il faut immédiatement noter, cependant, que cette séparation stricte n'est pas complètement gratuite. Si, par exemple, une partie avant sait qu'elle fait l'analyse pour une machine disposant d'instructions spécialisées pour les branchements à plusieurs destinations, elle peut probablement analyser les énoncés de cas ou d'aiguillage de manière à tirer bénéfice de ces instructions du langage machine. De manière similaire, si une partie arrière sait qu'elle produit du code pour un langage qui ne permet pas l'emboîtement des déclarations de sous-programmes, elle peut produire pour les appels de sous-programmes du code plus simple. Beaucoup de compilateurs professionnels sont des compilateurs intégrés, conçus pour un seul langage de programmation et une seule architecture de machine, qui utilisent une représentation sémantique qui dérive du langage source et qui contient déjà des éléments de la machine cible. La structure que nous venons de mentionner a cependant joué un rôle important dans l'introduction rapide de nouveaux langages et de nouvelles machines, et elle continue à le faire.

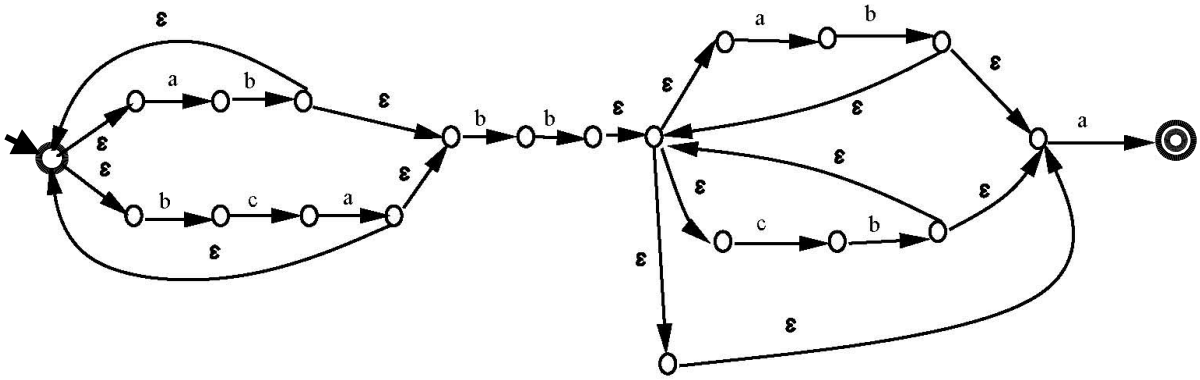
1-3 A quelle phase de la compilation peut-on détecter les erreurs suivantes ?
Identificateur mal formé en java ?
Conflit de type sin("a")
Variable non déclarée
Commentaire non fermé

Parenthèse non fermée
Begin non fermé
Mot réservé (if par exemple) utilisé comme identificateur
Non-conformité entre le nombre de paramètres formels et effectifs
Tentative de modifier une constante
Dépassement des bornes d'un tableau

1-4 Sur l'alphabet Σ = {a, b}, donner un automate fini déterministe reconnaissant les langages suivants :
– tous les mots contenant un nombre pair de "a" et un nombre pair de "b";
– tous les mots contenant un nombre pair de "a" ou un nombre pair de "b";
– tous les mots qui n'ont pas plus (¸) de quatre "a" consécutifs ;

1-5 Donnez, si vous pouvez, des expressions régulières pour les langages précédents.

1-6 Simplifiez :



1-7 Déterminez l'expression régulière correspondante à la simplification

1-8 On donne l'expression régulière :
L = x((xy)*zy*)*

Dessiner l'AFN puis l'AFD