Compilation

Présentation générale

claire.lefevre@univ-angers.fr

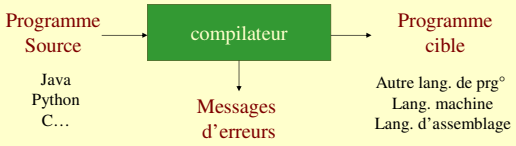
Compilateurs

- Un compilateur : un traducteur d'un langage source vers une machine cible
- Utilisation de base : compilation de langages de programmation
- Mais les idées et techniques utilisées sont utiles pour la conception de beaucoup d'autres applications

compilateur

- Programme qui lit un programme écrit dans un premier langage (langage source) et le traduit en un programme équivalent écrit dans un autre langage (langage cible)

+ signale à l'utilisateur la présence d'erreurs



Modèle de compilation par analyse et synthèse

- Analyse : sépare le programme source en ses différents constituants et crée une représentation structurée du prg
- Synthèse : construit le programme cible à partir de cette représentation

Analyse du programme source 3 phases

- Analyse lexicale : lecture linéaire du programme source pour regrouper les caractères en unités lexicales (« catégories de mots »)
  - Les unités lexicales sont généralement décrites par des expressions régulières

Exemple

Entrée (texte source) : x := y + 5

Unités lexicales : Ident, nombre, operateur, affectation, …

Sortie : Ident (x)
Affect
Ident (y)
Oper (+)
Nombre (5)

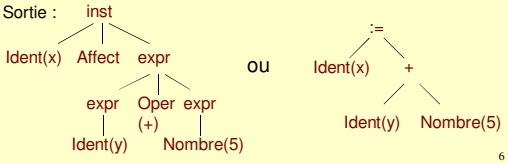
- Analyse syntaxique (ou grammaticale) : on regroupe les unités lexicales (« mots ») en structures grammaticales (« phrases ») qui seront représentées par un arbre syntaxique (ou un arbre abstrait)
  - Les structures grammaticales sont généralement décrites par une grammaire non contextuelle

Exemple

Entrée : Ident(x) Affect Ident(y) Oper(+) Nombre(5)

Règles syntaxiques : inst → Ident Affect expr
expr → expr Oper expr | Nombre | Ident

Sortie :

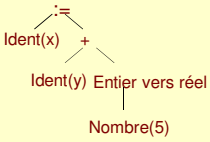


- Analyse sémantique : contrôle si le programme contient des erreurs sémantiques (en part. de typage) et collecte les informations de type nécessaires à la phase de production de code qui suit

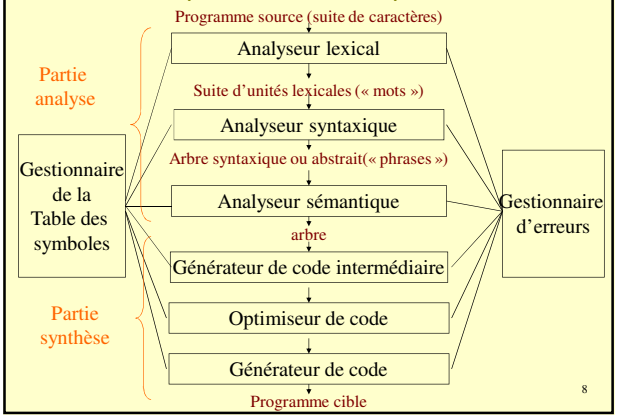
  En particulier, vérifie que les opérandes de chaque opération sont conformes aux spécifications du langage source, et insère éventuellement des conversions

Exemple

Sortie :



Les phases d'un compilateur



La table des symboles

- Elle sert à enregistrer :
  - les identificateurs utilisés
  - des informations sur ceux-ci (emplacement mémoire, type, portée, ou encore, pour un sous-programme, nombre et type des arguments, mode de passage des arguments…)
- Une table des symboles : une structure de données contenant un enregistrement pour chaque identificateur, avec des champs pour les informations le concernant (ses *attributs*)
- À chaque phase de la compilation, la table des symboles est complétée et/ou utilisée

La gestion des erreurs

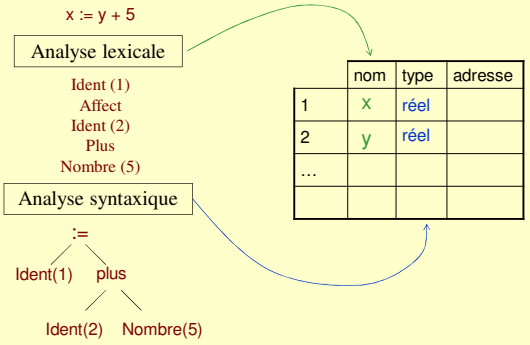
- Autre tâche importante d'un compilateur : la détection des erreurs (et, éventuellement, la reprise sur erreurs)
- Types d'erreurs détectables lors des phases d'analyse
  - Niveau lexical : les caractères lus ne forment aucun mot possible
  - Niveau syntaxique : phrases non conformes à la syntaxe du langage
  - Niveau sémantique : problèmes de typage d'expressions

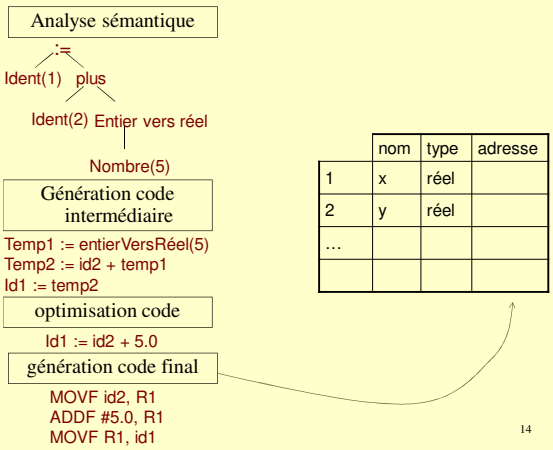
La partie synthèse (construction du programme cible)

- Production de code intermédiaire : construction d'un programme pour une machine abstraite
  - De nombreuses formes sont possibles
  - 2 contraintes : facile à produire, facile à traduire en langage cible
  - Ex : code 3-adresses
- Optimisation de code : amélioration du code intermédiaire pour que le code machine résultant s'exécute plus rapidement
  - Difficulté : améliorer significativement le code sans trop ralentir la compilation

- Production de code : production de code machine ou de langage d'assemblage pour la machine cible
  - Rôles :
    - Choix d'emplacements mémoire pour les variables
    - Traduction du code intermédiaire en une suite d'instructions machine
  - Exemple d'instructions de langage d'assemblage :
    - MOVF M1,M2 transfert d'1 nombre réel
    - MULTF M1,M2 multiplie 2 réels, résultat dans M2
    - ADDF M1,M2 additionne 2 réels, résu. dans M2
    - R1 registre (zone de calcul)
    - # préfixe des constantes

Récapitulatif





Regroupement de phases

En pratique, on regroupe souvent plusieurs phases

- Parties frontales et finales
  - Frontale : tout ce qui dépend du programme source et est indépendant du langage cible
    - Partie analyse
    - Génération de code intermédiaire
    - Une partie de l'optimisation de code
    - Traitement des erreurs, gestion de la table des symboles…
  - Finale : ce qui dépend de la machine cible et du code intermédiaire (pas du langage source)
    - Une partie de l'optimisation de code
    - Génération de code
  - Intérêt de ce découpage : la partie frontale est indépendante de la machine cible, on peut donc la réutiliser pour différentes machines, il suffit de réécrire la partie finale

- Les passes

Une passe = lire un fichier en entrée
et écrire un fichier en sortie

- Coûteux en temps

⇒ On implante généralement plusieurs phases de la compilation en une seule passe

Ex : analyseurs lexical et syntaxique

En pratique, il existe de grandes variations dans la façon dont les phases sont regroupées en passes,

=> Ici, on s'intéressera davantage aux phases qu'aux passes

Analyse lexicale

1ère phase de la compilation : reconnaissance des « mots »

- Ses tâches
  - Lire les caractères du programme source
  - Produire une suite d'unités lexicales (que l'analyseur syntaxique va utiliser pour former des « phrases »)
  - + éliminer tous les caractères non significatifs dans le programme
  - + relier les messages d'erreurs de la compilation au programme source (numéroter les lignes)
- Principe de fonctionnement
  - Reconnaître des motifs dans des chaînes
  - Déclencher des actions lorsqu'ils sont reconnus

  Divers outils logiciels fonctionnent selon ce principe.

  Ex : LEX, outil standard UNIX, qui permet de générer des analyseurs lexicaux

Définitions

- Unité lexicale : catégorie lexicale (« type de mot »)
- Modèle : règle qui définit l'unité lexicale (qui décrit l'ensemble des valeurs qu'elle peut prendre)
- Lexème : valeur de l'unité lexicale (suite de caractères du texte source)

Attributs des unités lexicales

L'analyseur lexical retourne une suite d'UL à l'analyseur syntaxique
Mais une UL pouvant représenter plusieurs lexèmes possibles, il faut fournir des informations additionnelles :

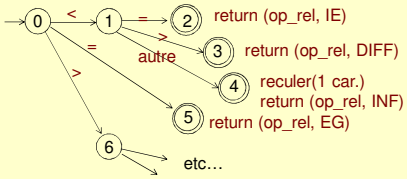
- Valeur pour un nombre ou autre constante
- Entrée dans la table des symboles pour un identificateur
- …

Spécification et reconnaissance des UL

- Spécification : par des expressions régulières
- Reconnaissance : par des automates (AFD)
  - L'AFD reconnaît les motifs spécifiés par l'expression régulière
  - Et effectue des actions dès qu'il a reconnu une UL

Exemple

Op_rel : < | <= | <> | = | > | >=



Construction de l'analyseur

Pb : construire un AFD à partir d'une ER

1) Transformation ER → AFN-ε → AFD
2) Lex : outil standard UNIX
   Entrée : ER + actions
   Sortie : AFD + programme C (ou C++) qui l'utilise pour reconnaître les unités lexicales du fichier d'entrée
3) D'autres outils et méthodes existent…

Schéma général de l'analyseur lexical