



SCIENCES SUP

Cours et exercices corrigés

2^e cycle • IUT • Écoles d'ingénieurs

COMPILATEURS

*Dick Grune
Henri E. Bal
Ceriél J.H. Jacobs
Koen G. Langendoen*

DUNOD

2-005-577-1

COMPILATEURS



Cours et exercices corrigés

Dick Grune

Professeur à l'université libre d'Amsterdam

Henri E. Bal

Professeur à l'université libre d'Amsterdam

Ceriel J.H. Jacobs

Professeur à l'université libre d'Amsterdam

Koen G. Langendoen

Professeur à l'université de Delft

Traduit de l'anglais par

Carine Fédèle et Olivier Lecarme

DUNOD

Table des matières

AVANT-PROPOS	XVII
NOTE DES TRADUCTEURS	XXIII
CHAPITRE 1 • INTRODUCTION	1
1.1 Pourquoi étudier la compilation?	5
1.1.1 La compilation est très fructueuse	6
1.1.1.1 Structuration appropriée du problème	6
1.1.1.2 Utilisation judicieuse des formalismes	7
1.1.1.3 Utilisation d'outils de génération de programmes	7
1.1.2 La compilation est d'application générale	8
1.1.3 Les compilateurs contiennent des algorithmes d'utilité générale	9
1.2 Un compilateur-interprète de démonstration	9
1.2.1 L'arbre abstrait	9
1.2.2 Structure du compilateur	11
1.2.3 Le langage source	12
1.2.4 Analyse lexicale	13
1.2.5 Analyse syntaxique	15
1.2.6 Analyse sémantique	18
1.2.7 Génération de code	19
1.2.8 Interprétation	20
1.3 Structure d'un compilateur plus réaliste	21
1.3.1 Structure	22
1.3.2 Support d'exécution	24
1.3.3 Quelques raccourcis	24
1.4 Architectures de compilateurs	25

1.4.1	Largeur d'un compilateur	25
1.4.2	Qui commande?	27
1.5	Propriétés d'un bon compilateur	30
1.6	Transportabilité et changement de machine cible	31
1.7	Place et utilité des optimisations	32
1.8	Bref historique de la compilation	33
1.8.1	1945-1960 : génération de code	33
1.8.2	1960-1975 : analyse syntaxique	33
1.8.3	Depuis 1975 : génération et optimisation du code ; paradigmes	33
1.9	Grammaires	34
1.9.1	Forme d'une grammaire	35
1.9.2	Processus de dérivation	35
1.9.3	Formes étendues de grammaires	37
1.9.4	Propriétés des grammaires	38
1.9.5	Formalisme grammatical	39
1.9.5.1	Définition d'une grammaire	39
1.9.5.2	Définition du langage engendré par une grammaire	40
1.10	Algorithmes de fermeture	41
1.10.1	Une implémentation itérative de l'algorithme de fermeture	44
1.11	Présentation du code schématique utilisé	46
1.12	Conclusion	47
1.13	Résumé	47
1.14	Notes bibliographiques	48
1.15	Exercices	49
CHAPITRE 2 • DU TEXTE DU PROGRAMME À L'ARBRE ABSTRAIT		53
2.1	Du texte du programme vers les lexèmes : la structure lexicale	57
2.1.1	Lecture du texte du programme	57
2.1.1.1	Problèmes de fin ^e de ligne	58
2.1.2	Séparation entre analyse lexicale et analyse syntaxique	59
2.1.3	Expressions régulières et descriptions régulières	59
2.1.3.1	Expressions régulières et BNF ou EBNF	61
2.1.3.2	Caractères d'échappement dans les expressions régulières	61
2.1.3.3	Descriptions régulières	61
2.1.4	Analyse lexicale	62
2.1.5	Écriture manuelle d'un analyseur lexical	63
2.1.5.1	Optimisation par précalcul	66
2.1.6	Génération automatique d'un analyseur lexical	69
2.1.6.1	Situations pointées	71
2.1.6.2	Recherche concurrente	76
2.1.6.3	Précalcul des ensembles de situations	80
2.1.6.4	L'analyseur lexical final	83

2.1.6.5	Complexité de la génération d'un analyseur lexical	84
2.1.6.6	Transitions vers $E\omega$	85
2.1.6.7	Complexité d'utilisation de l'analyseur lexical	85
2.1.7	Compression des tables de transition	86
2.1.7.1	Compression par déplacement des lignes	87
2.1.7.2	Compression par coloriage de graphe	91
2.1.8	Traitement des erreurs dans les analyseurs lexicaux	92
2.1.9	Un générateur traditionnel d'analyseurs lexicaux : (F)Lex	93
2.1.10	Identification des lexèmes	96
2.1.11	Tables de symboles	98
2.1.11.1	Implémentation d'un tableau extensible indexable par chaîne	99
2.1.11.2	Une optimisation simple	102
2.1.11.3	Fonctions d'adressage	102
2.1.12	Traitement des macros et inclusion de fichiers	103
2.1.12.1	La pile de tampons d'entrée	105
2.1.12.2	Inclusion conditionnelle de texte	107
2.1.12.3	Généricité par traitement de macros	108
2.1.13	Conclusion	109
2.2	Des lexèmes vers l'arbre syntaxique : la syntaxe	110
2.2.1	Deux classes de méthodes d'analyse syntaxique	111
2.2.1.1	Principes de l'analyse descendante	113
2.2.1.2	Principes de l'analyse ascendante	113
2.2.2	Détection et récupération des erreurs	114
2.2.3	Écriture manuelle d'un analyseur syntaxique descendant	116
2.2.3.1	Analyse syntaxique par descente récursive	117
2.2.4	Génération automatique d'un analyseur syntaxique descendant	120
2.2.4.1	Analyse LL(1)	120
2.2.4.2	Les conflits LL(1) vus comme un avantage	125
2.2.4.3	Les conflits LL(1) vus comme un inconvénient	127
2.2.4.4	L'automate à pile LL(1)	132
2.2.4.5	Traitement des erreurs en analyse LL	136
2.2.4.6	Un générateur traditionnel d'analyseurs descendants — LLgen	143
2.2.5	Génération automatique d'un analyseur syntaxique ascendant	150
2.2.5.1	Analyse LR(0)	151
2.2.5.2	L'automate à pile LR	157
2.2.5.3	Conflits LR(0)	159
2.2.5.4	Analyse SLR(1)	161
2.2.5.5	Analyse LR(1)	163
2.2.5.6	Analyse LALR(1)	168
2.2.5.7	Rendre ou non une grammaire LR(1)	169
2.2.5.8	Traitement des erreurs en analyse ascendante	172
2.2.5.9	Un générateur traditionnel d'analyseurs ascendants — Yacc ou Bison	175

2.3	Conclusion	178
2.4	Résumé	179
2.5	Notes bibliographiques	182
2.6	Exercices	182
CHAPITRE 3 • DÉCORATION DE L'ARBRE ABSTRAIT : LE CONTEXTE		193
3.1	Grammaires attribuées	194
3.1.1	Graphes de dépendance	199
3.1.2	Évaluation des attributs	201
3.1.2.1	Un évaluateur d'attributs dynamique	202
3.1.3	Traitement des circuits	208
3.1.3.1	Détection de circuit dynamique	209
3.1.3.2	Détection de circuit statique	209
3.1.4	Allocation des attributs	215
3.1.5	Grammaires attribuées à visites multiples	215
3.1.5.1	Visites multiples	215
3.1.5.2	Partitions des attributs	218
3.1.5.3	Grammaires attribuées ordonnées	221
3.1.6	Types de grammaires attribuées	227
3.1.7	Grammaires L-attribuées	228
3.1.7.1	Grammaires L-attribuées et analyseurs descendants	230
3.1.7.2	Grammaires L-attribuées et analyseurs ascendants	231
3.1.8	Grammaires S-attribuées	232
3.1.9	Équivalence des grammaires L-attribuées et S-attribuées	233
3.1.10	Notations grammaticales étendues et grammaires attribuées	234
3.1.11	Conclusion	235
3.2	Méthodes manuelles	236
3.2.1	« Couture » de l'arbre abstrait	236
3.2.2	Interprétation symbolique	242
3.2.2.1	Interprétation symbolique simple	244
3.2.2.2	Interprétation symbolique complète	248
3.2.2.3	Analyse de dernière définition	250
3.2.3	Équations de flot de données	250
3.2.3.1	Mise en route des équations de flot de données	252
3.2.3.2	Résolution des équations de flot de données	254
3.2.4	Analyse interprocédurale du flot de données	257
3.2.5	Transporter les informations en amont : analyse de vie	259
3.2.5.1	Analyse de vie par interprétation symbolique	260
3.2.5.2	Analyse de vie par équations de flot de données	262
3.2.6	Comparaison de l'interprétation symbolique et des équations de flot de données	265
3.3	Conclusion	266
3.4	Résumé	266

3.5	Notes bibliographiques	270
3.6	Exercices	270
CHAPITRE 4 • TRAITEMENT DU CODE INTERMÉDIAIRE		275
4.1	Interprétation	277
4.1.1	Interprétation récursive	277
4.1.2	Interprétation itérative	281
4.2	Génération de code	286
4.2.1	Se passer de génération de code	289
4.2.2	Le point de départ	291
4.2.3	Génération de code triviale	292
4.2.3.1	Code cousu	293
4.2.3.2	Évaluation partielle	295
4.2.4	Génération de code simple	297
4.2.4.1	Génération de code simple pour une machine à pile	300
4.2.4.2	Génération de code simple pour une machine à registres	302
4.2.4.3	Compilation sur la pile ou compilation par interprétation symbolique	312
4.2.5	Génération de code pour les blocs de base	313
4.2.5.1	Passage de l'arbre abstrait à un graphe de dépendances	315
4.2.5.2	Passage du graphe de dépendances au code	320
4.2.5.3	Optimisation de code en présence de pointeurs	326
4.2.6	Système de réécriture ascendant et programmation dynamique	329
4.2.6.1	Confrontation ascendante de modèle	334
4.2.6.2	Confrontation ascendante de modèles, de façon efficace	338
4.2.6.3	Sélection des instructions par programmation dynamique	340
4.2.6.4	Confrontation de modèles et sélection d'instructions combinées	343
4.2.6.5	Adaptation de l'algorithme de système de réécriture ascendant à différentes circonstances	348
4.2.7	Allocation de registres par coloriage de graphe	349
4.2.7.1	Graphe d'interférences des registres	350
4.2.7.2	Heuristique de coloriage de graphe	352
4.2.8	Supercompilation	354
4.2.9	Évaluation des techniques de génération de code	356
4.2.10	Mise au point des optimiseurs de code	356
4.2.11	Prétraitement du code intermédiaire	358
4.2.11.1	Prétraitement des expressions	358
4.2.11.2	Prétraitement des instructions si et de débranchement	360
4.2.11.3	Prétraitement des sous-programmes	360
4.2.12	Post-traitement du code objet	363
4.2.12.1	Création des modèles de remplacement	363
4.2.12.2	Repérage et remplacement des instructions	365
4.2.12.3	Évaluation de l'optimisation à lucarne	366

4.2.13	Génération du langage machine	366
4.3	Assembleurs, relieurs et chargeurs	368
4.3.1	Problèmes de conception des assembleurs	371
4.3.1.1	Traitement des adresses internes	372
4.3.1.2	Traitement des adresses externes	374
4.3.2	Problèmes de conception des relieurs	374
4.4	Conclusion	375
4.5	Résumé	375
4.6	Notes bibliographiques	382
4.7	Exercices	383
CHAPITRE 5 • GESTION DE LA MÉMOIRE		389
5.1	Allocation de données avec désallocation explicite	391
5.1.1	Allocation de mémoire de base	392
5.1.1.1	Optimisations pour l'allocation de mémoire de base	397
5.1.2	Les listes chaînées	398
5.1.3	Les tableaux extensibles	400
5.2	Allocation de données avec désallocation implicite	401
5.2.1	Les algorithmes de récupération de mémoire de base	402
5.2.2	Préparation du travail	404
5.2.2.1	Aide du compilateur dans la récupération de mémoire	405
5.2.2.2	Quelques hypothèses simplificatrices	409
5.2.3	Comptage des références	411
5.2.4	Marquage et balayage	415
5.2.4.1	Marquage	415
5.2.4.2	Balayage et libération	416
5.2.4.3	Retournement de pointeurs (marquage sans utilisation de l'espace de la pile)	417
5.2.5	Copie entre deux espaces	419
5.2.6	Compactage	422
5.2.7	Récupération de mémoire par génération	424
5.3	Conclusion	425
5.4	Résumé	425
5.5	Notes bibliographiques	429
5.6	Exercices	430
CHAPITRE 6 • PROGRAMMES IMPÉRATIFS ET ORIENTÉS OBJET		433
6.1	Analyse sémantique	434
6.1.1	Identification	436
6.1.1.1	Portée	438
6.1.1.2	Surcharge	441
6.1.1.3	Portées importées	443
6.1.2	Contrôle de type	444
6.1.2.1	La table des types	445

6.1.2.2	Équivalence de types	449
6.1.2.3	Conversion implicite	450
6.1.2.4	Forçage de type et conversion explicite	452
6.1.2.5	Contrôle de nature	452
6.1.3	Conclusion	454
6.2	Représentation et traitement des données du langage source	454
6.2.1	Types de base	455
6.2.2	Types énumérés	455
6.2.3	Types pointeurs	455
6.2.3.1	Pointeur non valide	456
6.2.3.2	Règles de portée des pointeurs	457
6.2.4	Types articles	460
6.2.5	Types articles avec variante	461
6.2.6	Types tableaux	462
6.2.7	Types ensembles	465
6.2.8	Types sous-programmes	465
6.2.9	Types objets	465
6.2.9.1	Caractéristique 1 : héritage	467
6.2.9.2	Caractéristique 2 : réécriture de méthodes	467
6.2.9.3	Caractéristique 3 : polymorphisme	468
6.2.9.4	Caractéristique 4 : liaison dynamique	469
6.2.9.5	Caractéristique 5 : héritage multiple	471
6.2.9.6	Caractéristique 6 : héritage multiple dépendant	473
6.2.9.7	Optimisations pour l'appel de méthode	474
6.2.10	Types interfaces	475
6.3	Les sous-programmes et leur activation	476
6.3.1	Zones d'activation	476
6.3.1.1	Contenu d'une zone d'activation	477
6.3.2	Sous-programmes	479
6.3.3	Opérations sur les sous-programmes	480
6.3.4	Sous-programmes non emboîtés	483
6.3.5	Sous-programmes emboîtés	485
6.3.5.1	Appel d'un sous-programme emboîté	487
6.3.5.2	Passage d'un sous-programme emboîté comme paramètre	489
6.3.5.3	Un sous-programme emboîté comme valeur de retour	490
6.3.5.4	Saut hors d'un sous-programme emboîté	490
6.3.5.5	Curryfication d'un sous-programme	491
6.3.5.6	Conclusion	492
6.3.6	Lambda-remontée	493
6.3.7	Générateurs et coroutines	495
6.4	Génération de code pour les énoncés de contrôle du flot	495
6.4.1	Flot de contrôle local	496

6.4.1.1	Expressions booléennes dans le flot de contrôle	496
6.4.1.2	Énoncés de choix	498
6.4.1.3	Énoncés répétitifs	501
6.4.2	Appel de sous-programme	505
6.4.2.1	Identification d'un sous-programme : quoi appeler	506
6.4.2.2	Appels de sous-programmes : comment les appeler	506
6.4.2.3	Sauts non locaux	513
6.4.3	Traitement des erreurs à l'exécution	513
6.4.3.1	Détection d'une erreur à l'exécution	513
6.4.3.2	Traitement d'une erreur à l'exécution	514
6.5	Génération de code pour les modules	517
6.5.1	Génération de noms	517
6.5.2	Initialisation de modules	518
6.5.2.1	Éviter les initialisations multiples	518
6.5.2.2	Détecter les dépendances circulaires	518
6.5.3	Génération de code pour la généricité	519
6.5.3.1	Instanciation par expansion	520
6.5.3.2	Instanciation par tableaux d'informations	520
6.6	Conclusion	521
6.7	Résumé	522
6.8	Notes bibliographiques	525
6.9	Exercices	526
CHAPITRE 7 • PROGRAMMES FONCTIONNELS		533
7.1	Une brève description de Haskell	535
7.1.1	Règle de retrait	535
7.1.2	Listes	536
7.1.3	Compréhension de liste	537
7.1.4	Confrontation de modèle	538
7.1.5	Typage polymorphe	538
7.1.6	Transparence des références	539
7.1.7	Fonctions d'ordre supérieur	540
7.1.8	Évaluation paresseuse	542
7.2	Compilation des langages fonctionnels	543
7.2.1	Le noyau fonctionnel	544
7.3	Contrôle de type polymorphe	546
7.3.1	Application de fonction polymorphe	547
7.4	Suppression des artifices syntaxiques	548
7.4.1	Traduction des listes	548
7.4.2	Traduction de la confrontation de modèle	549
7.4.3	Traduction des compréhensions de listes	552
7.4.4	Traduction des fonctions emboîtées	553

7.5	Réduction de graphe	555
7.5.1	Ordre de réduction	559
7.5.2	Le moteur de réduction	561
7.6	Génération de code pour les programmes du noyau fonctionnel	565
7.6.1	Éviter la construction de certaines arêtes d'application	567
7.7	Optimisation du noyau fonctionnel	569
7.7.1	Analyse d'exactitude	569
7.7.1.1	Analyse d'exactitude pour les fonctions récursives	571
7.7.1.2	Génération de code pour les arguments exacts	573
7.7.2	Analyse d'emboîtement	575
7.7.3	Appels terminaux	576
7.7.4	Transformation accumulatrice	577
7.7.5	Limitations	580
7.8	Manipulation de graphe avancée	580
7.8.1	Nœuds à longueur variable	580
7.8.2	Marquage des pointeurs	581
7.8.3	Allocation des nœuds par agrégats	581
7.8.4	Nœuds d'application vectoriels	581
7.9	Conclusion	582
7.10	Résumé	583
7.11	Notes bibliographiques	585
7.12	Exercices	586
CHAPITRE 8 • PROGRAMMES LOGIQUES		591
8.1	Le modèle de programmation logique	593
8.1.1	Blocs de base	593
8.1.2	Le mécanisme déductif	595
8.2	Le modèle général d'implémentation, interprété	596
8.2.1	Instructions de l'interprète	598
8.2.2	Éviter les listes de buts redondantes	601
8.2.3	Éviter de copier les queues de liste de buts	601
8.3	Unification	602
8.3.1	Unification des articles, des listes et des ensembles	603
8.3.2	Implémentation de l'unification	605
8.3.3	Unification de deux variables libres	608
8.3.4	Conclusion	609
8.4	Le modèle d'implémentation général, compilé	610
8.4.1	Procédures de liste	611
8.4.2	Recherche de la clause compilée et unification	613
8.4.3	Choix optimisé de la clause dans la WAM	618
8.4.4	Implémentation du mécanisme de coupure	621
8.4.5	Implémentation des prédicats <code>assert</code> et <code>retract</code>	622

8.5	Code compilé pour l'unification	628
8.5.1	Instructions d'unification de la WAM	629
8.5.2	Déduction d'une instruction d'unification par une évaluation partielle manuelle	630
8.5.3	Unification des articles dans la WAM	632
8.5.4	Une optimisation : mode lire/écrire	638
8.5.5	Autres optimisations d'unification de la WAM	641
8.5.6	Conclusion	642
8.6	Résumé	642
8.7	Notes bibliographiques	645
8.8	Exercices	645
CHAPITRE 9 • PROGRAMMES PARALLÈLES ET DISTRIBUÉS		649
9.1	Modèles de programmation parallèle	652
9.1.1	Partage de variables et moniteurs	652
9.1.2	Modèles de passage de messages	654
9.1.3	Langages orientés objet	656
9.1.4	L'espace de tuples de Linda	657
9.1.5	Langages à parallélisme de données	659
9.2	Processus et processus légers	660
9.3	Partage des variables	662
9.3.1	Verrous	662
9.3.2	Moniteurs	663
9.4	Passage de messages	664
9.4.1	Localisation du récepteur	665
9.4.2	Triage	666
9.4.3	Vérification de type des messages	667
9.4.4	Sélection des messages	667
9.5	Langages orientés objet parallèles	668
9.5.1	Localisation des objets	669
9.5.2	Migration des objets	670
9.5.3	Duplication des objets	671
9.6	Espace de tuples	672
9.6.1	Éviter le surcoût de l'adressage associatif	673
9.6.2	Implémentations distribuées de l'espace des tuples	676
9.7	Parallélisation automatique	678
9.7.1	Exploitation automatique du parallélisme	679
9.7.2	Dépendances des données	681
9.7.3	Transformations des boucles	683
9.7.4	Parallélisation automatique pour machines à mémoire distribuée	684
9.8	Conclusion	687
9.9	Résumé	687
9.10	Notes bibliographiques	689

9.11 Exercices	689
ANNEXE A • UN COMPILATEUR/INTERPRÈTE ORIENTÉ OBJET SIMPLE	693
A.1 Classes déterminées par la syntaxe et méthodes déterminées par la sémantique	693
A.2 Le compilateur orienté objet simple	695
A.3 Analyse syntaxique orientée objet	696
A.4 Evaluation	700
A.5 Exercice	702
ANNEXE B • SOLUTIONS DES EXERCICES	703
B.1 Solutions d'exercices du chapitre 1	703
B.2 Solutions d'exercices du chapitre 2	704
B.3 Solutions d'exercices du chapitre 3	707
B.4 Solutions d'exercices du chapitre 4	709
B.5 Solutions d'exercices du chapitre 5	712
B.6 Solutions d'exercices du chapitre 6	712
B.7 Solutions d'exercices du chapitre 7	714
B.8 Solutions d'exercices du chapitre 8	715
B.9 Solutions d'exercices du chapitre 9	715
B.10 Solutions de l'exercice de l'annexe A	716
ANNEXE C • GLOSSAIRE	717
C.1 Anglais-français	717
C.2 Français-anglais	723
BIBLIOGRAPHIE	729
TABLE DES FIGURES	743
INDEX	757