Table des matières

Avant-propos **XV**

1 Introduction à quelques styles de programmation **1**
1.1 La programmation fonctionnelle 1
1.1.1 Simplifier les conditions de la programmation 2
1.1.2 Posséder des formalismes sous-jacents 3
1.1.3 Considérer les fonctions comme des valeurs 4
1.1.4 Les contraintes de la programmation fonctionnelle 5
1.2 La programmation générique 7
1.2.1 Concilier la flexibilité et la précision de typage 8
1.2.2 Concilier la variabilité et la précision des liens entre modules . 10
1.3 La programmation objet 12
1.3.1 Modéliser de manière efficace 13
1.3.2 Rendre les types d'objets compatibles 13
1.3.3 Considérer les objets comme des valeurs 14
1.4 De la combinaison des styles 14

2 Introduction rapide à OCaml **15**
2.1 L'environnement interactif dédié 15
2.1.1 Utiliser l'environnement interactif 15
2.1.2 La part du typage 18
2.1.3 Exploiter la bibliothèque standard 19
2.2 Lire les types 20
2.2.1 Vérification de l'existence de définitions 21
2.2.2 Lire les types des fonctions unaires 21
2.2.3 Lire les types des fonctions n-aires 22
2.2.4 Lire les types paramétrés 25
2.3 Définir de nouvelles valeurs 27

3 Principes de la programmation fonctionnelle **31**
3.1 La base de la programmation fonctionnelle 31
3.1.1 Terminologie 31
3.1.2 Les variables et les liaisons 33
3.1.3 Définir de nouvelles liaisons globales 33
3.1.4 Définir des environnement locaux 36

3.1.5 Vies et masques des liaisons . . . 37
3.1.6 Le ramasse-miettes . . . 39
3.1.7 Définir de nouvelles fonctions . . . 40
3.1.8 Définir de nouvelles fonctions récursives . . . 41
3.1.9 Les environnements locaux dans les fonctions . . . 42
3.1.10 Les valeurs fonctionnelles et les fermetures . . . 45
3.1.11 Les constructions `let` : un système de cloisonnements . . . 47
3.2 Problèmes d'évaluations . . . 48
3.2.1 L'indépendance du résultat d'une évaluation . . . 48
3.2.2 L'évaluation par valeur . . . 49
3.2.3 L'évaluation par nom et l'évaluation paresseuse . . . 51
3.2.4 Le choix d'une stratégie d'évaluation . . . 54
3.2.5 Des cas particuliers aux stratégies d'évaluation . . . 55

4 Fonctions **57**
4.1 Formes et facettes des fonctions . . . 57
4.1.1 Les paramètres fonctionnels et la transmission de code . . . 57
4.1.2 Les fonctions dans des structures et organisation du code . . . 59
4.1.3 Les fonctions anonymes . . . 60
4.1.4 Utiliser les fonctions anonymes . . . 61
4.1.5 Définir des fonctions n-aires et curryfiées . . . 62
4.1.6 Utiliser les fonctions curryfiées . . . 63
4.1.7 La récursivité terminale . . . 66
4.1.8 L'infixation . . . 69
4.2 Généricité des fonctions . . . 71
4.2.1 Polymorphismes . . . 71
4.2.2 Les fonctions génériques . . . 72
4.2.3 Généricité comparée des fonctions . . . 75
4.2.4 La généricité des prédicats prédéfinis de comparaison . . . 77
4.2.5 Les limitations de la généricité des fonctions . . . 78
4.3 Typage des fonctions . . . 80
4.3.1 Le typage explicite . . . 80
4.3.2 Les abréviations de type . . . 81
4.3.3 Utiliser le typage explicite . . . 82
4.3.4 L'inférence de type et ses messages . . . 83
4.3.5 L'inférence de type et la recherche d'erreurs . . . 85

5 Histoires de types **89**
5.1 Zoologie des types . . . 89
5.1.1 La hiérarchie classique . . . 89
5.1.2 La hiérarchie d'accès . . . 91
5.1.3 Les types à caractère fonctionnel ou impératif . . . 93
5.2 Les types d'enregistrements . . . 94
5.2.1 Définir des types d'enregistrements . . . 94
5.2.2 Instancier les types d'enregistrements . . . 94
5.2.3 Accéder aux champs des enregistrements . . . 95

5.2.4 Les possibilités des types d'enregistrements 96
5.2.5 Les types de données et les enregistrements récursifs 97
5.2.6 Les limitations et les avantages des types d'enregistrements . 99
5.2.7 Aparté : la redéfinition de types 100
5.3 Les types inductifs 101
5.3.1 Définir des types inductifs 101
5.3.2 Instancier les types inductifs 102
5.3.3 Les possibilités des types inductifs 103
5.3.4 L'inductivité des types inductifs 105
5.4 La reconnaissance de motif 106
5.4.1 La notion de motif 106
5.4.2 La reconnaissance de motif 106
5.4.3 Le filtrage 109
5.4.4 Les raisons d'utiliser le filtrage 112
5.4.5 Les limitations de la reconnaissance de motif et du filtrage . . 114
5.5 Premières utilisations des types inductifs 117
5.5.1 L'union de types distincts 117
5.5.2 Clarification des programmes 118
5.5.3 L'implémentation des fonctions partielles 118
5.5.4 L'émulation de la surcharge 119
5.5.5 Programmation défensive : le style à typage dense 121
5.5.6 Les types inductifs simples et les types d'enregistrements . . . 123
5.5.7 Le typage d'expressions structurées 124
5.5.8 La représentation de grammaires 125
5.6 Le problème de l'extensibilité des types 129
5.6.1 Les éléments et les critères de l'extensibilité des types 129
5.6.2 L'extensibilité des types d'enregistrements 130
5.6.3 L'extensibilité des types inductifs 131
5.6.4 Les types inductifs et l'étiquetage explicite 133
5.6.5 Des types inductifs flexibles : les variants polymorphes 135

6 Structures de données fonctionnelles 141
6.1 Les listes 142
6.1.1 La représentation d'une structure de données séquentielle . . . 142
6.1.2 Les listes : un type structuré prédéfini 144
6.1.3 Quelques fonctions sur les listes 145
6.2 Techniques sur les structures fonctionnelles. I. 148
6.2.1 Les itérateurs 148
6.2.2 Structures vides et fonctions partielles 149
6.2.3 Les tables d'associations et les contraintes de structure 151
6.3 Les arbres 152
6.3.1 Les arbres binaires 152
6.3.2 Les arbres : un exemple de structures de données composées . 157
6.3.3 Les compositions multiples 160
6.4 L'effet de persistance 160
6.4.1 Une propriété de la programmation fonctionnelle 160

6.4.2 Les avantages de la persistance . . . 163
6.4.3 Les inconvénients de la persistance . . . 165
6.5 Techniques sur les structures fonctionnelles. II. . . . 166
6.5.1 Remédier aux inconvénients de la persistance . . . 166
6.5.2 Maîtriser les parcours sur les structures et leurs ordonnancements 170
6.5.3 La récursivité terminale sur les listes . . . 173
6.5.4 La récursivité terminale sur les arbres . . . 174
6.6 Difficultés des structures fonctionnelles . . . 176
6.6.1 Limitations des types inductifs . . . 176
6.6.2 Ambiguïtés des types inductifs . . . 178
6.6.3 L'homogénéité et l'hétérogénéité des structures de données . . . 183
6.6.4 Les graphes ou le problème des structures non fonctionnelles . 186

7 Traits impératifs 193
7.1 Les exceptions . . . 194
7.1.1 Fonctions partielles et niveaux de traitement des erreurs . . . 194
7.1.2 Un type de base prédéfini : les exceptions . . . 197
7.1.3 Lever des exceptions . . . 197
7.1.4 Capturer et traiter des exceptions . . . 199
7.1.5 Les exceptions comme moyen d'optimisation . . . 200
7.1.6 Les exceptions en tant que valeurs . . . 202
7.2 Les procédures et l'unité . . . 203
7.2.1 Les procédures . . . 203
7.2.2 Les séquences . . . 205
7.2.3 Un `if-then` simple . . . 206
7.3 Utiliser les procédures . . . 206
7.3.1 La technique primitive de recherche d'erreurs . . . 206
7.3.2 Des évaluations extérieures à l'environnement interactif . . . 207
7.3.3 L'affichage graphique de base . . . 209
7.3.4 Les assertions . . . 210
7.3.5 Le traitement de fichiers ou OCaml en langage de script . . . 211
7.4 Les tableaux : un type structuré prédéfini . . . 214
7.4.1 L'accès direct et l'affectation . . . 214
7.4.2 La création de nouveaux tableaux . . . 215
7.4.3 Les itérateurs internes sur les tableaux . . . 216
7.4.4 Les boucles . . . 216
7.4.5 Les chaînes de caractères : un type à caractère impératif . . . 218
7.5 Les enregistrements à champs modifiables . . . 219
7.6 Les références . . . 219
7.6.1 Des éléments modifiables . . . 219
7.6.2 Les références : un type de base prédéfini . . . 220
7.6.3 Une difficulté de typage : les références génériques . . . 222
7.6.4 Opérations à caractère impératif et références génériques . . . 223
7.6.5 Utilisation des fermetures et références génériques . . . 224
7.6.6 Structures à caractère impératif et références génériques . . . 224
7.6.7 La non-généralisation et les variables faibles de type . . . 226

7.6.8 La non-généralisation et l'application de fonctions 229
7.6.9 Les structures impératives et les variables faibles de type . . . 231
7.7 Le choix de l'impératif 232
7.7.1 Quelques difficultés de l'impératif 232
7.7.2 La propagation de l'impératif 235
7.7.3 L'interfaçage avec le langage C 237

8 Techniques de programmation fonctionnelle 239
8.1 La programmation incrémentale 240
8.1.1 Un problème : l'analyse d'une suite de symboles 241
8.1.2 Un premier prototype 242
8.1.3 Des tests plus élaborés 245
8.1.4 Une représentation graphique 248
8.1.5 Les avantages de la programmation incrémentale 249
8.2 La généralisation 251
8.2.1 L'idée de la généralisation 251
8.2.2 La généralisation des transformations : les pliages 252
8.2.3 La généralisation des traitements : un résumé 257
8.2.4 La généralisation des algorithmes 258
8.2.5 La généralisation explicite de type 263
8.2.6 La généralisation explicite de type par polytypisme 265
8.2.7 Généralisations et réutilisations 267
8.3 La programmation dirigée par les données 272
8.3.1 Une généralisation généralisée 272
8.3.2 Une technique de programmation globale 275
8.3.3 Le style impératif et la programmation dirigée par les données 276
8.3.4 L'extensibilité des types et la programmation dirigée par les données 278
8.4 Représentation de données par des fonctions 279
8.4.1 Des ensembles comme des fonctions 280
8.4.2 Des sous-espaces comme des fonctions 283
8.4.3 Des formes géométriques comme des fonctions 285
8.4.4 Des réels comme des fonctions 288
8.5 Le contrôle de l'évaluation 289
8.5.1 Le principe du contrôle de l'évaluation 290
8.5.2 La congélation : un nouvel usage du type `unit` 290
8.5.3 La congélation et la paresse 292
8.5.4 Le comportement de la congélation fonctionnelle 295
8.5.5 Des structures de données à gestion paresseuse 297
8.5.6 Congeler le flux de l'évaluation et récursivité terminale 298
8.6 Les listes paresseuses ou flots 301
8.6.1 La congélation d'une structure séquentielle 301
8.6.2 Accéder aux éléments des flots 303
8.6.3 Variations sur la définition des flots 304
8.6.4 La récursivité et l'infini congelés 305
8.6.5 Les itérateurs et les compositions de flots 307

8.6.6 User à bon escient de l'infini congelé . . . 309
8.6.7 Les flots et la frontière de la paresse . . . 311
8.6.8 La production de valeurs sous forme de flots . . . 313
8.6.9 Les itérateurs externes sous forme de flots . . . 314
8.6.10 Des ensembles infinis sous forme de flots . . . 316
8.6.11 Les flots : une technique globale de programmation . . . 318
8.7 Les arbres paresseux . . . 319
8.7.1 La congélation d'une structure arborescente . . . 319
8.7.2 Utilisations des arbres paresseux . . . 323
8.7.3 Le choix de la paresse explicite . . . 327
8.8 Programmation par passage de continuation . . . 328
8.8.1 La notion de continuation . . . 328
8.8.2 Le passage de continuation et les fonctions partielles . . . 329
8.8.3 Le contrôle de l'évaluation par continuations . . . 331
8.8.4 Le cas des arbres rouge-noir . . . 333
8.8.5 Le cas des arbres 2-3 . . . 335

9 Programmation modulaire 341
9.1 Les modules : un encapsulateur général . . . 342
9.1.1 Définir des modules . . . 342
9.1.2 Accéder aux éléments des modules . . . 343
9.1.3 Les types de données sous forme modulaire . . . 344
9.1.4 Remarques sur la règle de définition des modules . . . 346
9.1.5 Les modules locaux . . . 348
9.2 Les signatures : des types pour les modules . . . 349
9.2.1 Les signatures inférées . . . 349
9.2.2 Définir des signatures . . . 350
9.2.3 Les types abstraits . . . 351
9.2.4 Des éléments implémentés dans les signatures . . . 352
9.2.5 Les signatures pour le typage explicite des modules . . . 353
9.2.6 Le rapport d'instanciation entre modules et signatures . . . 356
9.3 Techniques d'écriture de signatures . . . 358
9.3.1 Les signatures comme moyen de spécification et d'interface . . . 358
9.3.2 Les limitations des spécifications par signatures . . . 358
9.3.3 Atténuer les limitations des spécifications par signatures . . . 359
9.3.4 La généralisation des signatures . . . 360
9.3.5 Les signatures et le style fonctionnel ou impératif . . . 361
9.4 Le privé et le public des modules . . . 362
9.4.1 Le public en général . . . 362
9.4.2 Le privé en général . . . 363
9.4.3 La compatibilité de signatures . . . 364
9.4.4 Des exceptions publiques . . . 365
9.4.5 Des types publics . . . 366
9.4.6 Des types privés et des types de données abstraits . . . 367
9.5 Les types de données abstraits . . . 369
9.5.1 Les raisons du masquage et de la substitutivité . . . 369

9.5.2 Faire évoluer les types de données . . . 370
9.5.3 Optimiser les types de données par « mémoïsations » . . . 372
9.5.4 Le masquage et la généricité des prédicats prédéfinis . . . 375
9.5.5 Le masquage et les optimisations par l'impératif . . . 376
9.5.6 La nécessité du masquage individualisé des types abstraits . . . 379
9.5.7 Le masquage individualisé et les contraintes de type . . . 382
9.5.8 La généralisation conceptuelle des signatures . . . 384
9.5.9 Les tactiques de nommage des types de données abstraits . . . 386
9.5.10 Les paramètres universels et les types abstraits . . . 388
9.6 Les relations modulaires par inclusions . . . 392
9.6.1 L'inclusion de modules et de signatures . . . 392
9.6.2 L'inclusion pour l'héritage . . . 393
9.6.3 L'inclusion pour l'adaptation . . . 395
9.7 Les relations modulaires par imbrications . . . 396
9.7.1 Les imbrications de modules . . . 396
9.7.2 Les ouvertures de modules . . . 397
9.7.3 Les imbrications dans les signatures : les sous-modules abstraits 399
9.7.4 Les liens entre signatures par sous-modules abstraits . . . 399
9.7.5 Les sous-modules abstraits et les inclusions contraintes . . . 401
9.7.6 Le masquage des types des sous-modules . . . 402
9.7.7 Aplanir les imbrications de modules . . . 404
9.7.8 Les agrégations modulaires de types de données . . . 404
9.7.9 Les associations modulaires de types de données . . . 406
9.7.10 Un exemple plus complet : les « hommes-pommes » . . . 409
9.8 Le masquage des types en question . . . 411
9.8.1 Le choix du masquage . . . 411
9.8.2 Les « modules-enregistrements » . . . 413
9.8.3 Le masquage *a posteriori* . . . 415
9.8.4 Le masquage et l'association modulaire . . . 416
9.8.5 Les limitations des contraintes de types . . . 417
9.8.6 L'extension de la représentation des valeurs . . . 418
9.9 Les modules en fichiers et compilation séparée . . . 420
9.9.1 Le chargement de modules dans la boucle d'interaction . . . 420
9.9.2 La compilation séparée de modules . . . 422
9.9.3 Les modules précompilés et la boucle d'interaction . . . 424
9.9.4 Les modules précompilés et les effets de bord . . . 425
9.9.5 La technique des sur-modules : les paquetages . . . 426

10 Programmation modulaire générique : les foncteurs 431
10.1 Les foncteurs . . . 432
10.1.1 L'idée des « fonctions de modules » . . . 432
10.1.2 Définir des foncteurs . . . 433
10.1.3 Utiliser les paramètres des foncteurs . . . 434
10.1.4 Appliquer les foncteurs . . . 435
10.1.5 Le typage explicite des résultats de foncteurs . . . 436
10.1.6 Les types partagés et la transmission de types . . . 437

10.1.7 Les paramètres non modulaires 438
10.1.8 Les foncteurs : une programmation modulaire générique typée 439
10.2 Techniques d'utilisation des foncteurs 440
10.2.1 Les héritages modulaires génériques 440
10.2.2 Les adaptations modulaires génériques 442
10.2.3 Les décorations modulaires génériques 443
10.2.4 Les composites modulaires génériques 445
10.2.5 Les types de données génériques 447
10.3 Les foncteurs et la maîtrise de la généricité des signatures 448
10.3.1 Les signatures et les paramètres universels de type 448
10.3.2 Les signatures et les types abstraits 449
10.3.3 Passer d'une généricité des types à une autre ? 451
10.4 Les foncteurs et leur compilation séparée 453
10.5 Les foncteurs n-aires 456
10.5.1 Définir des foncteurs n-aires 456
10.5.2 Appliquer les foncteurs n-aires 457
10.5.3 Un exemple plus complet : les graphes génériques 461
10.5.4 Les contraintes sur les paramètres des foncteurs 463
10.5.5 Les types partagés entre les paramètres des foncteurs 467
10.5.6 Une difficulté : les types partagés et le masquage 469
10.5.7 La technique de la « pépinière de types » 470
10.6 Les signatures de foncteurs 472
10.6.1 Des types pour les foncteurs 472
10.6.2 La spécification et les signatures de foncteurs 475
10.7 Les foncteurs de foncteurs 476
10.7.1 Une première utilisation de foncteurs de foncteurs 476
10.7.2 Les contraintes sur les paramètres fonctoriels 477
10.7.3 La généricité contrôlée par foncteurs de foncteurs 477
10.7.4 Remarque sur la généralisation par foncteurs 480
10.7.5 Des modules et des foncteurs presque de première classe . . . 481
10.8 Le choix du modulaire générique 482
10.8.1 Les foncteurs : une autre programmation fonctionnelle 483
10.8.2 Les foncteurs : un outil de génie logiciel 483
10.8.3 La programmation modulaire générique en question 487
10.9 Représenter une architecture générique 489
10.9.1 Le développement logiciel et la programmation générique . . . 489
10.9.2 Les graphes de spécification et les automates d'architecture . 491
10.9.3 Représenter la compatibilité des signatures 494
10.10 Exemples d'automates d'architecture 497
10.10.1 Des files de priorité génériques 497
10.10.2 Une gestion de hiérarchies de signatures 500
10.10.3 Des constructeurs de lignes de production génériques 501
10.10.4 Remarques sur les automates d'architecture 505

11 Programmation objet **507**
11.1 Pourquoi une nouvelle espèce de valeurs ? 507
11.1.1 Les objets sous forme d'enregistrements 508
11.1.2 Les objets sous forme de modules 509
11.1.3 Les objets sous forme de valeurs fonctionnelles 511
11.2 Les objets et les classes 513
11.2.1 Définir des objets 513
11.2.2 Accéder aux éléments d'un objet 515
11.2.3 Les références mutuelles au sein des objets 516
11.2.4 Les types des objets 516
11.2.5 La compatibilité entre les types d'objets 518
11.2.6 Les types ouverts 520
11.2.7 Les classes : des constructeurs d'objets 521
11.2.8 Types ouverts et généricité contrainte 525
11.3 Quelques possibilités des classes 525
11.3.1 Un mécanisme d'héritage 525
11.3.2 Les classes génériques 526
11.3.3 Les classes à la généricité contrainte 529
11.3.4 Les redéfinitions et les liaisons retardées 529
11.3.5 Les types des classes 531
11.3.6 La spécification individualisée : les méthodes abstraites 533
11.4 Quelques techniques de programmation objet 534
11.4.1 L'interaction avec la programmation fonctionnelle 534
11.4.2 Implémentations multiples aux types compatibles 535
11.4.3 La bonne gestion des méthodes n-aires 540
11.4.4 Les types de données sous forme de classes 543
11.4.5 Programmation objet et programmation modulaire 545
11.4.6 Programmation objet et programmation modulaire générique . 546
11.5 Le choix de l'objet 550
11.5.1 Les avantages principaux de la programmation objet 550
11.5.2 La programmation objet et la récursivité mutuelle 550
11.5.3 L'extensibilité en programmation objet 552
11.5.4 Les inconvénients principaux de la programmation objet . . . 553