

Functional Programming in Clean

Preface

Functional languages enable programmers to concentrate on the problem one would like to solve without being forced to worry about all kinds of uninteresting implementation details. A functional program can be regarded as an executable specification. Functional programming languages are therefore very popular in educational and research environments. Functional languages very suited for teaching students the first principles of programming. In research environments they are used for rapid prototyping of complex systems. Recent developments in the implementation techniques and new insights in the underlying concepts such as input/output handling make that modern functional languages can nowadays also be used successfully for the development of real world applications.

The purpose of this book is to teach practical programming skills using the state-of-the-art pure functional language Concurrent Clean. Clean has many aspects in common with other modern functional languages like Miranda, Haskell and ML. In addition Clean offers additional support for the development of window based applications as well as support for process communication and for the development of (distributed) applications.

The book is split into three parts.

In the first part of this book an introduction into functional programming is given. In five Chapters we treat the basic aspects of functional programming, functions, data structures, the type system and I/O handling. The target here is that you are able to write functions and programs as soon as possible. A complete description of all allowed language constructs can be found in the Clean language manual.

The main emphasis of this book lies in the second part in which eleven case studies are presented. Each case treats a tiny, but complete application in an illustrative problem domain. Each case furthermore illustrates a certain aspect of functional programming. Some case applications are reused in others to illustrate the reusability of code.

The following cases are presented:

1. a simple database, illustrating simple file I/O and the construction of dynamically configurable dialogue interfaces;
2. a relational database (based on the simple database) illustrating the basic aspects of a relational database, showing the use of functional languages as executable specification as well as the reusability of code;
3. a window based text editor, illustrating the basic way to do window based text handling;

4. a window based graphical editor, illustrating the basic aspects of window based handling of graphical elements as well as given an example of an object oriented style of functional programming;
5. parser combinators, which will give a good insight in the different kind of parsing techniques, while we also illustrate a combinatorial style of functional programming;
6. an interpreter for a functional programming language for a subset of Clean (using the parser combinator tool of Chapter 5 and the text editor of Chapter 3), illustrating the basic semantics of functional languages and the reusability of code;
7. a spreadsheet (reusing the interpreter of Chapter 6) in which we also show the difference between interpreted and compiled code;
8. a computer architecture and programming language, in which we show how a functional language can be used as executable specification of an abstract machine architecture; also an assembler and imperative programming language is defined on top of the machine architecture showing the compilation path from high level language, through assembly language, into machine language.
9. compression and decompression of data, in which we show how the change an executable specification for a data compression / decompression algorithm into an efficient functional application.
10. process control, in which the processes and process communication is illustrated;
11. a distributed organizer, in which the use of distributed processes is shown.

In the third part of this book we discuss the different kinds of programming development techniques for functional programming (Chapter 1) and finally we treat efficiency aspects (Chapter 2).

So, a lot of material is presented in this book. However, one certainly does not have to work through all case studies. Depending on the programming experience already acquired and the time available one can use this book as a textbook for or one or two semester course. The book can be used as an introductory textbook for people with little programming experience. It can also be used for people who already have programming experience in other programming paradigm (imperative, object-oriented or logical) and now want to learn how to develop applications in a pure functional language.

We hope that you enjoy the book and that it will stimulate you to use a functional language for the development of your applications.

Table of Contents

Preface	1
Table of Contents	1
Introduction to Functional Programming	1
1.1 Functional languages	1
1.2 The Clean compiler	2
1.2.1 The `Start' expression	2
1.2.2 Defining new functions	3
1.3 Standard functions	4
1.3.1 Names of functions and operators	4
1.3.2 Numeric functions	4
1.3.3 Boolean functions	5
1.3.4 Functions on lists	5
1.3.5 Functions on functions	6
1.4 Function definitions	6
1.4.1 Definition by combination	6
1.4.2 Definition by cases	7
1.4.3 Definition using patterns	7
1.4.4 Definition by induction or recursion	8
1.4.5 Layout	9
1.4.6 Comments	10
1.5 Types	10
1.5.1 Sorts of errors	10
1.5.2 Typing of expressions	11
1.5.3 Polymorphism	12
1.5.4 Functions of more parameters	13
1.5.5 Overloading	13
1.5.6 Type annotations and attributes	14
1.5.7 Well formed Types	15
1.6 Synonym definitions	16
1.6.1 Global constant functions (CAF's)	16
1.6.2 Macro's and type synonyms	17
1.7 Modules	17
1.8 Exercises	19

Numbers and Functions	21
2.1 Operators	21
2.1.1 Operators as functions and vice versa	21
2.1.2 Priorities	21
2.1.3 Association	22
2.1.4 Definition of operators	23
2.2 Partial parameterization	24
2.2.1 Currying of functions	24
2.3 Functions as parameters	25
2.3.1 Functions on lists	25
2.3.2 Iteration	27
2.3.3 Function composition	27
2.3.4 The lambda notation	28
2.4 Numerical functions	29
2.4.1 Calculations with integers	29
Calculating a list of prime numbers	29
Compute the day of the week	30
2.4.2 Calculations with reals	32
The derivative function	32
Definition of square root	33
2.5 Exercises	34
Data Structures	35
3.1 Lists	35
3.1.1 Structure of a list	35
Enumeration	36
Construction using :	37
Enumerable intervals	37
3.1.2 Functions on lists	38
Comparing and ordering lists	38
Joining lists	39
Selecting parts of lists	39
Reversing lists	41
Properties of lists	41
3.1.3 Higher order functions on lists	42
map and filter	42
takewhile and dropwhile	42
3.1.4 Sorting lists	43
Insertion sort	43
Merge sort	44
3.1.5 List comprehensions	44
Quick sort	46
3.2 Infinite lists	46
3.2.1 Enumerating all numbers	46
3.2.2 Lazy evaluation	48
3.2.3 Functions generating infinite lists	49
3.2.4 The list of all prime numbers	49
3.3 Tuples	50
3.3.1 Tuples and lists	52
3.4 Records	53
3.4.1 Rational numbers	55
3.5 Arrays	56
3.5.1 Array comprehensions	57

3.5.2	Lazy, strict and unboxed arrays	58
3.5.3	Array updates	58
3.5.4	Array patterns	59
3.6	Algebraic datatypes	59
3.6.1	Tree definitions	61
3.6.2	Search trees	62
	Structure of a search tree	63
3.6.3	Sorting using search trees	64
3.6.4	Deleting from search trees	64
3.7	Abstract datatypes	65
3.8	Run-time errors	66
3.8.1	Non-termination	66
3.8.2	Partial functions	67
3.8.3	Cyclic dependencies	68
3.8.4	Insufficient memory	69
3.9	Exercises	70
	 The Power of Types	 71
4.1	Type Classes	71
4.1.1	Overloading	72
4.1.2	A class for Rational Numbers	75
4.1.3	Derived class members	76
4.1.4	Type constructor classes	77
4.2	Existential types	78
	Creating objects by existential types	79
	A pipeline of functions	83
4.3	Uniqueness types	83
4.3.1	Graph Reduction	84
4.3.2	Destructive updating	85
4.3.3	Environment passing	86
4.3.4	Uniqueness information	86
4.3.5	Propagation of uniqueness	89
4.3.6	Uniqueness polymorphism	89
4.3.7	Attributed data types	91
4.3.8	Higher order uniqueness typing	92
4.3.9	Creating unique objects	93
4.3.10	Combining uniqueness typing and type classes	94
4.4	Exercises	94
	 Input and Output	 95
5.1	Changing the world	95
5.2	Combination of input/output functions	97
5.2.1	Monadic style	98
5.2.2	Nested scope style	99
5.2.3	Polymorphic Uniqueness	99
5.3	Some Simple Dialogs	100
5.3.1	A File Copy Dialog	100
5.3.2	A Function Test Dialog	104
5.3.3	An Input Dialog for a Menu Function	106
5.3.4	More Generic Dialog Definitions	107
5.4	A simple window	107
5.5	Timers	111
5.6	A line drawing program	112

5.7	Exercises	118
	Advanced Programming	119
6.1	Efficiency of programs	119
6.1.1	The unit to measure efficiency	120
6.1.2	Complexity	120
	Upper bounds	120
	Under bounds	121
	Tight upper bounds	122
6.1.3	Counting reduction steps	122
	memorisation	122
	Determining the complexity for recursive functions	124
	Manipulation recursive data structures	125
	Determining upper bounds and under bounds	128
6.1.4	Constant factors	129
	Measurements: generating a pseudo random list	130
	Measurements: sorting lists	131
	Other ways to speed up programs	132
6.1.5	Exploiting Strictness	134
6.1.6	Unboxed values	135
6.1.7	The cost of Currying	137
6.1.8	A word of warning	139
6.2	A guide to local definitions and scopes	139
6.2.1	Local definitions	139
6.2.2	Scope within expressions	141
6.3	Equational reasoning	142
	Direct proofs	142
	Proof by cases	143
	Proof by induction	144
	Program synthesis	146
6.4	Tracing program execution	148
6.5	Higher order functions on lists	149
6.5.1	Displaying a number as a list of characters	149
6.5.2	Folding	150
	foldr	150
	foldl	151
	Folding to the right or to the left	151
6.4	Exercises	152
	A Simple Database	155
1.1	The database program state	155
1.2	At the start and the end of the application	156
1.3	File handling	158
1.4	Displaying the database contents	160
1.5	Changing the database contents	163
1.6	Handling database queries	165
1.7	Changing the format of the database	167
1.7.1	Deleting an attribute field	168
1.7.2	Moving an attribute field	169
1.7.3	Renaming an attribute field	169
1.7.4	Adding an attribute field	170
1.8	Exercises	171

A Relational Database	173
2.1 To be written	173
2.2 Exercises	173
An Editor	175
3.1 The editor program state	175
3.2 File access	176
3.3 Displaying the text	178
3.4 Cursor handling	181
3.5 Text and window coordinates	184
3.6 Editing with keyboard actions	184
3.7 Highlighting	186
3.8 'Sane' mouse handling	187
3.9 Changing Font	189
3.10 Exercises	190
An Editor for Graphical Objects	191
4.1 User view of the graphical editor	191
4.2 Structure of the program	191
4.3 Mouse Handling	193
4.4 Grouping	197
4.5 Editing	198
4.6 Exercises	198
Parser Combinators	199
5.1 The type of parsers	200
5.2 Elementary parsers	201
5.3 Grammars	202
5.4 Parser combinators	203
5.5 Parser transformers	204
5.6 Matching parentheses	205
5.7 More parser combinators	207
5.8 Analyzing options	210
5.9 Arithmetical expressions	211
5.10 Generalized expressions	212
5.11 Monadic parsers	213
5.12 Context sensitivity	214
5.13 Common traps	215
Left recursion	215
Parsing the same structure again	216
5.14 Error handling	217
Detecting errors	217
Interrupting the parser	218
Error recovery	219
Listing errors	220
5.15 Self application	221
Environments	221
Grammars	222
Parse trees	223
Parsers instead of grammars	223
A parser generator	223
Lexical scanners	224

References	224
Exercises	224
Interpreter for a functional programming language	227
6.1 The interpreted programming language	227
Use of the interpreter	228
6.2 Parser	228
6.2.1 The function parse	230
Remarks on this parsing scheme	233
Parsing of function definitions	233
Transformation of parsed expressions	235
6.3 Evaluation	235
6.4 User Interface	237
6.5 Examples programs for the interpreter	238
Sys.fp (system file)	238
Test.fp (example file)	239
6.6 Adding type checking to the interpreter	239
6.6.1 Restriction on the interpreted language due to the type checker	239
Definition of type	240
Supporting functions	240
The functions derivetype and buildType	241
Spreadsheet	245
7.1 The calculation model of a spreadsheet	245
7.2 A spreadsheet compiler	247
7.1.2 Compiler program	248
7.3 A spreadsheet interpreter	251
Computer Architecture and Languages	254
8.1 Computer Architecture	255
8.1.1 Memory Components	256
8.2 Instructions	257
8.2.1 Storing instructions in the memory	259
8.3 Running the Machine	260
8.3.1 Booting the Machine	260
8.4 Input-Output	260
8.5 Assembly Languages	262
8.5.1 An Example Assembly Program	263
8.6 Tracing the Execution of Programs	264
8.7 High Level Languages	265
8.7.1 An Example Program in Tiny	266
8.8 Compilation	266
8.8.1 Example of generated assembly code	269
8.8.2 How to prove the correctness of the compiler	269
8.9 Interpretation	270
8.10 Correctness	271
8.11 Summary	273
8.12 Exercises	273
Compression / Decompression	276
Introduction	276
LZW compression	276

LZW compression in Clean	277
Decompression	278
Performance results	279
Decompression	281
Compression results	282
Improvements	283
Further improvements	285
Conclusion	285
Process Control	287
10.1 To be written	287
10.2 Exercises	287
A Distributed Organizer	289
11.1 To be written	289
11.2 Exercises	289
Program development	290
1.1 Software engineering	290
1.2 Waterfall model	291
1.3 Program development strategies	294
Top-down	296
Bottom-up	296
Incremental	297
Evolutionary	297
1.4 Spiral model	298
The double helix model	300
1.5 Functional programming languages	301
1.6 Software quality	302
Software maturity	303
1.7 Guidelines for software construction	305
Programming styles and paradigms	307
2.1 Programming styles	307
List comprehensions	308
Explicit recursion	309
Toolbox functions	309
Continuation passing	310
Characteristics of the programming styles	311
Execution times	311
List comprehensions	312
Explicit recursion	312
Toolbox functions	312
Continuations	313
Conclusion	313
2.2 Programming paradigms	313
Imperative	313
Logic programming	317
Object Oriented	317
Exercises	317

Efficiency of programs	319
3.1 Graph rewriting in Clean	319
3.2 Complexity	322
Analysis	323
Reduction	323
3.3 Imperative reduction machine	323
3.4 Strictness	325
Left recursion	326
unboxed basic values	328
3.5 uniqueness	329
in situ update	330
manipulation of unique objects	330
3.6 Transformations to increase efficiency	330
fold/unfold	331
deforestation	331
Clean 1.1 syntax	333
A.1 Clean program	333
A.2 Function definition	334
A.3 Graph definition and expression	335
A.5 Macro definition	336
A.6 Type definition	336
A.6 Class definition	337
A.7 Symbols	337
A.8 Identifiers	337
A.9 Denotations	338
Standard Environment version 1.1	339
B.1 Cleans' Standard Environment	339
B.1.1 StdOverloaded: predefined overloaded operations	340
B.1.2 StdClass: predefined classes	340
B.1.3 StdBool: operations on Booleans	341
B.1.4 StdInt: operations on Integers	341
B.1.5 StdReal: operations on Reals	342
B.1.6 StdChar: operations on Characters	342
B.1.7 StdList: operations on Lists	343
B.1.8 StdCharList: operations on lists of characters	344
B.1.9 StdTuple: operations on Tuples	344
B.1.10 StdArray: operations on Arrays	345
B.1.11 StdString: operations on Strings	345
B.1.12 StdFunc: operations on polymorphic functions	346
B.1.13 StdMisc: miscellaneous functions	346
B.1.14 StdFile: File based I/O	346
B.1.15 StdEnum: handling dot-dot expressions	348
Clean 0.8 I/O Library	351
C I/O library	351
C.1 General operations on the IOState (deltaEventIO)	351
C.2 Definition of the I/O system (deltaIOSystem)	352
C.3 Operations on the timer device (deltaTimer)	355
C.4 Operations on menus (deltaMenu)	356
C.5 Operations on windows (deltaWindow)	357

C.6	Operations on dialogs (deltaDialog)	358
C.7	The file selector dialogs (deltaFileSelect)	360
C.8	Predefined Controls (deltaControls)	360
C.9	Miscellaneous operations (deltaIOState)	361
C.10	Operations on pictures (deltaPicture)	361
C.11	Operations on fonts (deltaFont)	364
C.12	System-dependent constants and functions (deltaSystem)	365