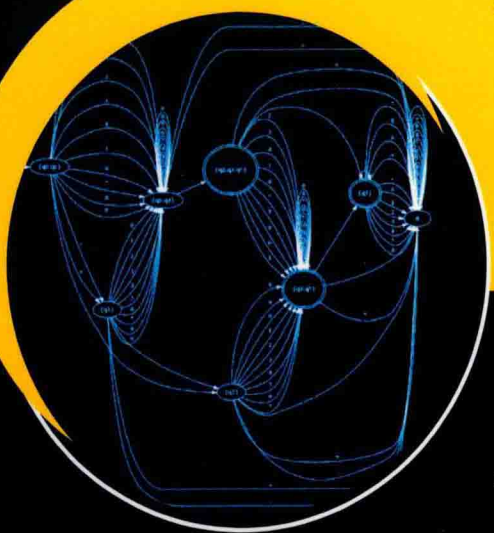
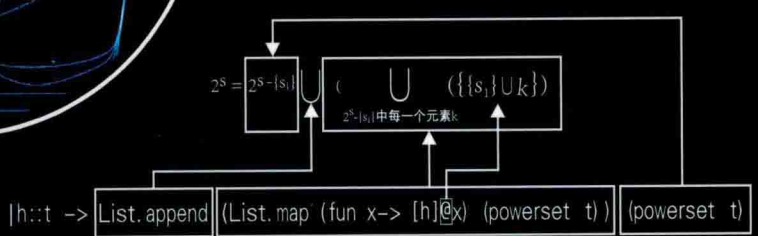
F#语言
及其在自动机理论中的应用

潘庆和　孙华东　著





電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

F#语言及其在自动机理论中的应用

潘庆和　孙华东　著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书以函数式编程语言 F#的基础知识为主要内容，详细介绍利用 F#解决 DFA，NFA、∈-NFA 的描述和互相转化问题，DFA、正则表达式和正则表达式、∈-NFA 之间的转化问题，下推自动机的瞬时描述问题，对确定性图灵机也做了相关的描述和模拟。

本书适合作为高等院校相关专业的研究生教材，也可供开发人员参考。

图书在版编目（CIP）数据

F#语言及其在自动机理论中的应用 / 潘庆和，孙华东著. —北京：电子工业出版社，2014.8

ISBN 978-7-121-24386-8

Ⅰ. ①F… Ⅱ. ①潘… ②孙… Ⅲ. ①程序语言-程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（(2014)）第 218840 号

责任编辑：富 军
印 刷：北京京华虎彩印刷有限公司
装 订：北京京华虎彩印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：720×1000 1/16 印张：12.5 字数：217.6 千字
版 次：2014 年 8 月第 1 版
印 次：2014 年 8 月第 1 次印刷
定 价：38.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

本书主要介绍两部分的内容：第一部分介绍函数式编程语言 F#的基础知识，包括代码编写方式、基本语法和常用数据结构；第二部分主要利用 F#对自动机理论中的一些理论问题进行描述和求解。其中，第一部分并没有对 F#的全部语言特征进行介绍，只介绍在第二部分中会用到的基础知识；第二部分理论问题的描述和解决主要是依照《Introduction to Automata Theory, Languages, and Computation, 3rd Edition》中自动机部分的脉络展开的，它是自动机理论领域的权威著作之一，也是目前国内外广泛使用的教材。结合本书进行实际编程操作可加深对自动机理论部分的理解。

在应用 F#解决理论问题时，不只使用 F#函数式编程设计风格，也使用了命令式编程风格，这些风格都是 F#所支持的，而且这种混合的编程方式也可能有助于从命令式方式向函数式方式编程的过渡。基础部分介绍的数据结构，使用也比较灵活，目的是展示解决方法的多样性。F#提供了很多模块函数，方便结构间的转换。

目前，国内函数式程序设计的教材比较少，并且自动机教学偏重于理论，本书通过利用函数式编程方法解决自动机理论中的相关问题，兼顾了这两种情况。

本书得到了作者单位哈尔滨商业大学博士科研启动资金的大力支持。

如果读者阅读中发现问题，请及时与我们联系，希望大家多多批评指正。

目　录

第一部分　基　础　篇

第 1 章　F#简介和环境搭建 …… 3

1.1　F#简介 …… 3

1.2　环境搭建 …… 5

1.3　F#工程结构 …… 5

第 2 章　F#语言基础 …… 21

2.1　值与类型 …… 21

2.2　函数 …… 35

2.3　列表 …… 39

2.4　数组 …… 45

2.5　序列 …… 46

2.6　控制结构 …… 48

第二部分　应　用　篇

第 3 章　DFA（Deterministic finite state automata） …… 55

3.1　什么是 DFA …… 55

3.2　DFA 识别的语言 …… 59

第 4 章 NFA（Nondeterministic finite state automata） …… 65
4.1 什么是 NFA …… 65
4.2 NFA 识别的语言 …… 68
4.3 NFA 到 DFA 的转化 …… 72
第 5 章 ∈-NFA …… 93
5.1 什么是∈-NFA …… 93
5.2 ∈闭包 …… 98
5.3 ∈-NFA 识别的语言 …… 101
5.4 ∈-NFA->DFA 的转化 …… 103
第 6 章 正则表达式和自动机 …… 118
6.1 正则表达式 …… 118
6.2 DFA 转化为正则表达式 …… 120
6.3 正则表达式转化为自动机 …… 128
第 7 章 下推自动机 …… 160
7.1 下推自动机的定义 …… 160
7.2 PDA 的瞬时描述 …… 165
第 8 章 图灵机 …… 179
8.1 图灵机简介 …… 179
8.2 图灵机的瞬时描述 …… 181
参考文献 …… 192

第一部分

基 础 篇

第 1 章　F#简介和环境搭建

第 2 章　F#语言基础

第 1 章　F#简介和环境搭建

1.1　F#简介

F#（读音“F Sharp”）是微软推出的函数式、强类型、多范式编程语言。强类型是指程序中出现的量必须具有明确的类型，可以指定，也可以由环境推断；多范式是指 F#可以支持函数式编程、命令式编程及面向对象编程等多种编程方式。F#是.NET 众多编程语言，如 C#、VB.NET、VC++中的一员。

近年来，函数式编程思想和概念越来越受到业界的重视，正从传统的计算机科学研究领域转到实际生产环境。F#作为微软推出的函数式编程语言，具有着特殊优势。从比较权威的 TIOBE 编程语言排行榜可以看出，F#当前已经上升至编程语言排行榜的前 15 名之内，如图 1-1 所示。

因此，学习和了解这门语言有着非常重要的意义。学习一门语言比较有效的方式应该包括两点：第一，如果已学过其他的编程语言，那么可以通过比较学习的方式掌握新语言的语法特点，如果是新接触，就要通过学习实例代码的方式掌握语言的语法规则和解决问题的基本形式；第二，最好有一个待解的问题贯穿学习过程的始终，因为学习的时间可能比较有限，带着问题思考，以解决一个问题的心态吸取相关的知识。有了这两点就可以奠定一定的基础，而不一定需要在一开始就面面俱到的掌握语言的所有特点。本书的编写考虑到了这

Jul 2014	Jul 2013	Change	Programming Language	Ratings	Change
1	1		C	17.145%	-0.48%
2	2		Java	15.688%	-0.22%
3	3		Objective-C	10.294%	+0.05%
4	4		C++	5.520%	-3.23%
5	7	^	(Visual) Basic	4.341%	+0.01%
6	6		C#	4.051%	-2.16%
7	5	v	PHP	2.916%	-4.27%
8	8		Python	2.656%	-1.38%
9	10	^	JavaScript	1.806%	-0.04%
10	12	^	Transact-SQL	1.759%	+0.19%
11	9	v	Perl	1.627%	-0.52%
12	13	^	Visual Basic .NET	1.495%	+0.24%
13	37	^^	F#	1.093%	+0.86%
14	11	v	Ruby	1.072%	-0.51%
15	45	^^	ActionScript	1.067%	+0.86%
16	-	^^	Swift	1.054%	+1.05%
17	17		Delphi/Object Pascal	1.031%	+0.34%
18	15	v	Lisp	0.829%	-0.04%
19	18	v	MATLAB	0.781%	+0.10%
20	20		Assembly	0.777%	+0.20%

图 1-1　2014 年 7 月 ITOBE 全球编程语言排行

两个特点，因此主要分为两个部分：F#基础篇和 F#解决问题的应用篇。这两部分有机地结合在一起。基础篇所介绍的语法都是在应用篇中遇到的，而不是将 F#语言的所有语法全面地描述。应用篇选取的问题是自动机理论领域的相关问题，在解决这些问题的时候，基于 F#多范式编程的特点，不只使用了函数式编程方式，也使用了命令式编程方式，同时对于数据结构的选择也比较灵活，这样做的好处是可以在比较熟悉的命令式编程方式环境下过渡到函数式编程的风格。

1.2　环境搭建

Visual Studio 2013 版中包括 F# 3.0 版本，可以安装此环境后运行本书中的例子。另外也可以选择其他的 IDE（集成开发环境），如 SharpDevelop 和 MonoDevelop 等。本书中的环境为 Visual Studio 2013 Ultimate。安装后，启动程序 Visual Studio，将出现下面的启动界面，如图 1-2 所示。

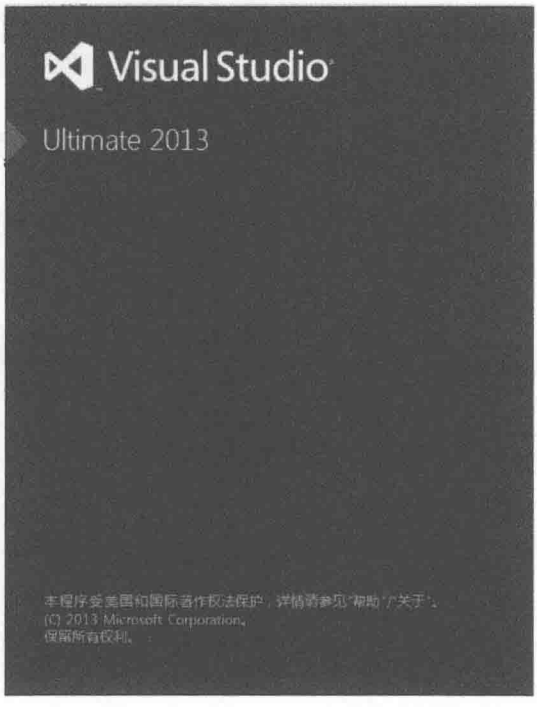


图 1-2　Visual Studio 2013 的启动界面

1.3　F#工程结构

1. 新建项目

可以像.NET 框架下其他语言的项目建立方式一样，建立一个 F#项

目，这可以通过“文件”菜单的“新建”→“项目”来完成，如图 1-3 所示。

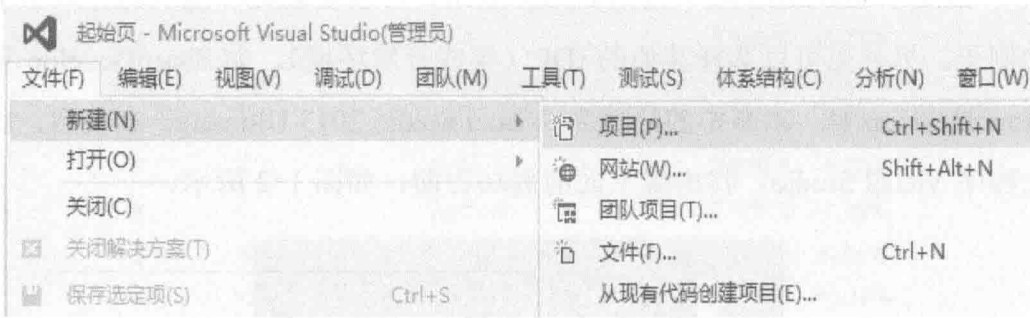


图 1-3　新建项目

弹出下面的窗口，显示 F#新建项目可以选择的项目类型，如图 1-4 所示。

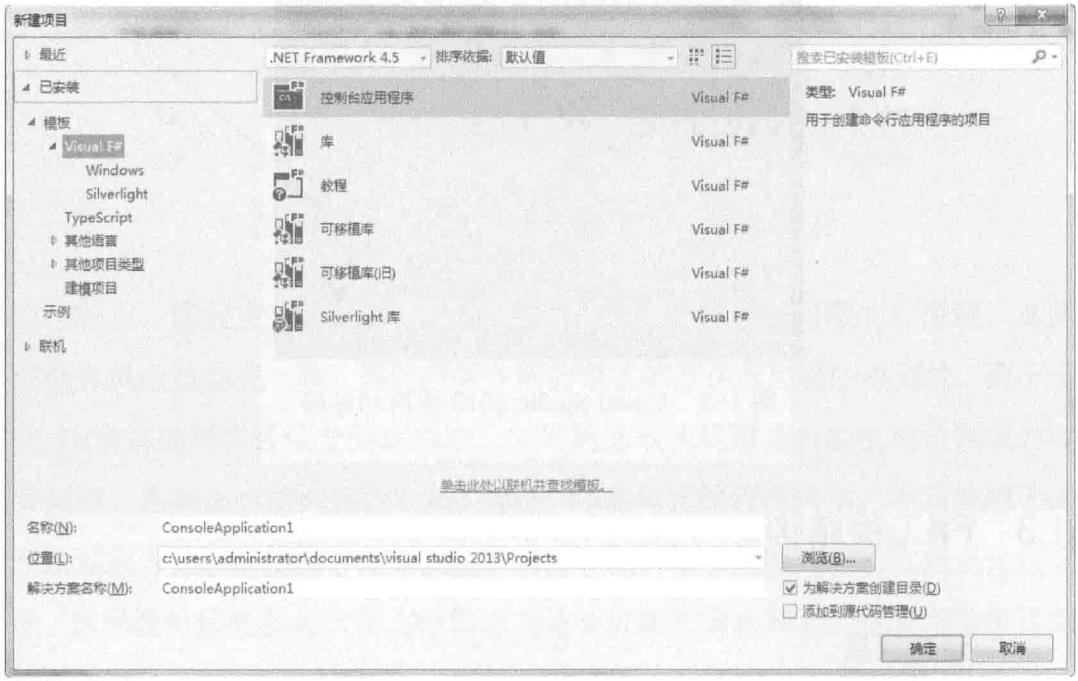


图 1-4　F#项目可供选择的类型

其中：

① 控制台应用程序：创建一个命令行应用程序；

② 库：创建一个新库，该库可被其他应用程序或库引用；

③ 教程：可以快速查看 F#提供的示例；

④ 可移植库：可以创建一个可移植类库，可以被.NET4.5 和 Windows Store 应用共同使用；

⑤ 可移植库（旧）：创建一个可移植库，可以被.NET4.0 和 Silverlight 应用共同使用；

⑥ Silverlight 库：创建一个可以在 Silverlight 中使用的库。

在这些模板中，并没有出现 Windows Form 应用、WPF 应用及 ASP.NET 应用，这是因为，对于这几类设计工具并没有升级至能够产生或理解 F#代码。尽管如此，仍可以使用这些技术构造 F#应用，但需要更多的手工设置工作。

2. 工程结构

比如，在上面的对话框中选择“控制台应用程序”模板，设置好名称、位置和解决方案名称后，单击“确定”后，将会建立这个工程，并直接导航到如图 1-5 所示的工程界面。

尽管乍一看 F#的工程界面布局和其他类型的.NET 工程界面布局相似，但 F#内部的工程组织结构和传统的.NET 语言有很大的不同。主要不同在于：

（1）F#代码是自上而下执行的

这意味着不但在一个单独的代码文件内各种声明语句是有序的，整个工程内所有文件的排列顺序也是很重要的。如果我们建立一个新文件 B 并在

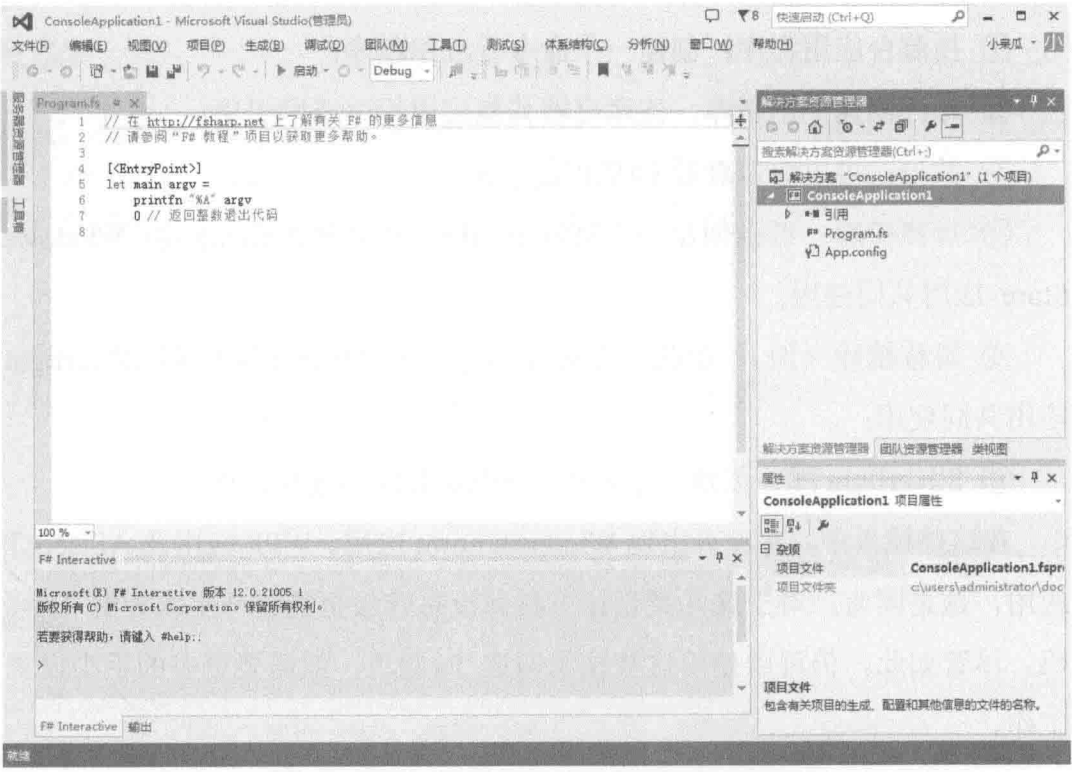


图 1-5　F#控制台应用程序的工程界面

文件中进行一些定义，且这些定义中的某些会被以前所定义的文件 A 中的代码所使用，那么如果新建文件 B 在工程中的位置位于以前的文件 A 之后，那么编译器将产生编译错误，提示这些新的定义找不到。解决的方式是将新建文件 B 移动到文件 A 上方，IDE 中已经考虑到了这一点，提供了相应的功能菜单，如图 1-6 所示。其中，“上移”、“下移”、“在上方添加”及“在下方添加”将辅助完成此类操作。

（2）在 F#工程中不允许出现文件夹

F#工程的 IDE 不允许向工程添加文件。

图 1-6　文件顺序移动选项

3. 第一个 F#程序

（1）编译方式

下面以上面建立的控制台应用程序工程为基础，给出一个 F#程序的例子。

在图中所示代码 Program.fs 的第 6 行下加入下面两行，即

```
let bool_list = List.map (fun x -> x % 2 = 0) [1 .. 5]
printfn "%A" bool_list
```

形成的程序为

```
// 在 http://fsharp.net 上了解有关 F# 的更多信息                    ①
// 请参阅“F# 教程”项目以获取更多帮助。                              ②
```

```
[<EntryPoint>]                                                    ③
let main argv =                                                   ④
    printfn"%A"argv                                               ⑤
    let bool_list = List.map (fun x -> x % 2 = 0) [1 .. 5]        ⑥
    printfn"%A"bool_list                                          ⑦
    0 // 返回整数退出代码                                          ⑧
```

为了说明方便，在每行程序的后面标注了序号。

①和②两行为注释，F#的注释有两种形式。其中，“//”为行注释，在需要注释的文字或代码之前加上“//”，后面的部分被认为是被注释的内容，从⑧可以看出，行注释不一定非得标注在每一行的开始，位置比较灵活，比如这里放在了程序代码的后面。

另一种注释的形式可认为是块注释，记号为“(**)”,可将要注释的内容放在“(*”和“*)”之间，如对于①、②两行，可以用块注释的方法注释为

```
(*
    在 http://fsharp.net 上了解有关 F# 的更多信息                ①
    请参阅“F# 教程”项目以获取更多帮助。                          ②
*)
```

③中的[<EntryPoint>]，正如其名称所示，表示“进入点”，在[<EntryPoint>]后定义的函数为控制台应用程序的入口点，应用程序由此开始执行，④即是[<EntryPoint>]属性所修饰的函数。

④定义了一个函数 main。其中，let 为定义函数所必需的关键字，在定义其他一些值时也需要使用 let 这个关键字，如⑥所定义的 bool_list。

argv 为 main 函数的参数，其类型为 string []，即字符串数组，任何被定义为进入点的函数必须以一个字符串型数组作为参数，该参数将存储程序被启动时传入的参数，同时进入点函数，也需要返回一个整数值，如⑧中的 0。

这里需要注意的是，由于已经有[<EntryPoint>]属性修饰了，因此④中定义的函数将会被认为是入口点函数，只要形式满足进入点函数的定义，即有字符串数组作为参数，以整数值作为返回值就可以了，与函数具体的名称是没有什么关系的。作为参数的 argv，其名称也可改为其他合法的名称，且返回的整数值也可指定为别的值。

④行最后一个“=”后的部分是 main 函数的函数体。在 F#中，函数体并不需要用“begin”、“end”或“{”，“ }”这样的界限标志符号括起来，而是使用缩进来指示代码的层次结构，其他的语言，如 python 也采用这种缩进的方式表示代码的逻辑层次。观察⑤、⑥、⑦和⑧这四行，其缩进层次相同，逻辑层次也相同。

⑤中 printfn 的作用是输出，向控制台输出相应的信息，功能与 C 语言中的 printf 类似，但省去了“(”和“)”。这里，printfn 接收两个参数，“%A”和 argv。其中，“%A”规定了输出格式，还有其他的格式可以选择；argv 是传递给程序的参数形成的字符串数组。那么，⑤的功能是向控制台输出编译好的控制台应用程序在启动运行时所接受的参数信息。

⑥中定义了一个量 bool_list。它的值是 List.map (fun x -> x % 2 = 0) [1 .. 5] 的执行结果。这里可将⑥和④进行一下对比，④定义了一个函数，因为 main 在一个空格后接了 argv 这个标志，无论这个标志是什么，都表示它是 main 的一个参数，其他语言中函数的参数一般放在括号中，F#中的函数定义和调用形式，正如上面的 printfn 形式一样，可以省略括号，同时也支持将参数加括号括起的形式，不过这两种方式有一定的区别。在函数部分，我们将介绍不同形式之间的差异。如果像⑥中在用 let 定义时只给出一个量，后面直接连接一个“=”，那么一般定义的是一个量，用来保存“=”右边部分的执行结果。这个量也可以保存一个函数，比如