Getting started with F#

Jon Sporring

July 2, 2016

1 Source code

F# is a functional first programming language that also supports imperativ and object oriented programming. It also has strong support for parallel programming and information rich programs. It was originally developed for Microsoft's .Net platform, but is available as open source for many operating systems through Mono. In this text we consider F# 4.0 and its Mono implementation, which is different from .Net mainly in terms of the number of libraries accessible. The complete language specification is described in `http://fsharp.org/specs/language-spec/4.0/FSharpSpec-4.0-latest.pdf`.

F# has 2 modes of execution, *interactive* and *compiled*. Interactive mode is well suited for small experiements or back-of-an-envolope calculations, but not for programming in general. In Mono, the interactive system is started by calling `fsharpi` from the console, while compilation is performed with `fshparc` and execution of the compiled code is performed using the `mono` command. The various forms of fsharp programs are identified by suffixes:

`.fs` An *implementation file*

`.fsi` A *signature file*

`.fsx` A *script file*

`.fscript` Same as `.fsx`

`.exe` An *executable file*

The implementation, signature, and script files are all typically compiled to produce an executable file, but syntactical correct code can also be entered into the interactive system, in which case these are called *script-fragments*. The implementation and signature files are special kinds of script files used for building *modules*.

2 Executing programs

Programs may either be executed by the interpreter or by compiling and executing the compiled code.

In `Mono` the interpreter is called `fsharpi` and can be used in 2 ways: interactively, where a user enters 1 or more script-fragments separated by the `;;` token, or to execute a script file treated as a single script-fragment. To illustrate the difference, consider the following program, which declares a value `a` to be the *float* value 3.0 and finally print it to the console:

```
let a = 3.0
printfn "%g" a
```

An interactive session is obtained by starting the console, typing the `fsharpi` command, typing the lines of the program, and ending the script-fragment with the `;;` token:

```
$ fsharpi

F# Interactive for F# 4.0 (Open Source Edition)
Freely distributed under the Apache 2.0 Open Source License

For help type #help;;

> let a = 3.0
- printfn "%g" a;;
3

val a : float = 3.0
val it : unit = ()

> #quit;;
```

The interpreter is stopped by pressing `ctrl-d` or typing `#quit;;`. Conversely, executing the file with the interpreter as follows,

```
$ fsharpi gettingStartedStump.fsx
3
```

Finally, compiling and executing the code is performed as,

```
$ fsharpc gettingStartedStump.fsx
F# Compiler for F# 4.0 (Open Source Edition)
Freely distributed under the Apache 2.0 Open Source License
$ mono gettingStartedStump.exe
3
```

Both the interpreter and the compiler translates the source code into a format, which can be executed by the computer. While the compiler performs this translation once and stores the result in the executable file, the interpreter translates the code everytime the code is executed. Thus, to run the program again with the interpreter, then it must be retranslated as '`$ fsharpi gettingStartedStump.fsx`', but since the program has been compiled, then the compile-execute only needs the be re-executed '`$ mono gettingStartedStump.exe`'. The interactive session results in extra output on the *type inference* performed, which is very useful for debugging and development of code-fragments, but both executing programs with the interpreted directly from a file and compiling and executing the program is much prefered for programming complete programs, since the starting state is well defined, and since this better supports *unit-testing* a tool for debugging programs consiting of many code-fragments.