The Early History of F#

Kenneth Fossen
May 2021

This essay presents the paper "The Early History of F#" [25] by Done Syme [6] presented at the HOPL IV [12]. The paper discusses the historical summary for how the F# language was created, and how the early days functional programming (FP), and Don Syme's personal experience with FP influenced F#'s design and implementation.

I choose this paper, for a few reasons. I've been experiencing FP through other University in Bergen's (UiB) courses (INF122/22 [9] [22]), and have been enjoying the new way of thinking and the Hindley-Miler type-inference magic [11]. I've also been curious to transfer this knowledge from my courses where I've been using FP, into my part-time job , where we mainly work in the imperative language C# and .NET environment. And this paper would give me a good introduction to the language, learn it by using it, and get to understand how FP and imperative programming can inter operate in the same code base in the .NET SDK. [8] If I was to get to use FP in my current workplace that does .NET, this interop between C#, F# and .NET seems like the likely way of letting this happen, and hopefully we could harvest the best of both worlds for the software development team.

1 Brief Overview of the HOPL Paper

The HOPL paper "The early history of F#" [25] is journey of strongly typed functional programming languages in the early 80's to the creation of F# in late 2001 and inclusion of .NET Generics in 2005.

In this period, it covers briefly, what the basis of functional programming is, and how this inspired the paper "Why no one uses functional programming" by Philip Wadler [26]. *This paper was central to my understanding of the programming language landscape*[25, p 9] for Don when he entered Microsoft Research in 1998. The rise of object-oriented programming (OO) that lead to Wadler and Odersky with a mission to *integrate specific technical features associated with strongly typed functional languages into "mainstream" OO languages.*[25, p9] that then formed the creation of Pizza programming language. This later led to Generics being incorporated in Java, and later influenced Dons work on .NET Generics [24]. This was an important inclusion in .NET as it shaped parts of F#.

In 2003, 0.5 version of F# was released, but it didn't get much attention before it's F# 1.0 in 2005. From 2001 to 1.0 the was added a many features and the language matured. Many of the early inclusions in F# 1.0 is today the most beloved features of the language. The Pipeline operator is one of the popular features that was added early in the language. Later Units-of-measure was added and Type Providers to mention some.

In the period 2005 $\rightarrow$ 2020, the paper covers the evolution of F#, and how the industry changed and how Microsoft had to adapt to this. F# got a new dawn with becoming open source, and cloud computing was on the rise where it played a central part.

Don also takes us through some language mistakes [25, p 51] that in retrospective should not have been added to the language. Here he mentions the e.g the Back piping operator and he discusses his early F# decision about not including, a oft-requested feature in F# at present time, type classes that originates from Haskell.

2 Brief Overview of F#

F# is a strongly typed functional programming language that has been heavily inspired by OCaml [21]. It started out as a project getting OCaml to run on .NET, but there was resistance in the Microsoft Research community. *So, why do we really need a .NET port of OCaml? OCaml is working fine on Windows, and on many other OS.*. [25, p 16].

But the author decided to write F# from scratch and port the core of OCaml to target .NET.

It is strongly typed through their own implementation of Hindley Miler that works together with .NET Generics. It comes as a part of .NET SDK since 2010, and has a rich REPL through `dotnet fsi` that supports F# Scripting. Other tools are mentioned in section *Tool Support for F#*

Popular features of F# is Units-of-Measure [5], Active Patterns [2], Type Providers and Quotations [3].

Units-of-Measure is type-safety for metrics. If two measures are incompatible, you e.g. can't sum them. Also is removes any doubt about what metric this number is representing in your code. Popular example for this is NASA's Mars Climate Orbiter that failed due to a confusion between newtons and pound force. [17]

Active Patterns allows for partial, complete and multi-case patterns. Makes it easy to create new patterns and use them. Don's example is parsing Int and Bool in his paper. `let (|Int|_) str = ...` [25, p 29].

Quotation is used for Meta-programming [19] in F# with Abstract-Syntax Tree's (AST) [1] and can be used to generate code or work with language creation in F#.

Type providers is a feature that gives strongly typed data sources. E.g working with JSON or Databases.

NuGet [7] is the packet manager for .NET projects for C# and F#, and Since F# is delivered through .NET SDK, you can add any new NuGet packet and start using these libraries in your code. Doesn't matter if it is written in C# or F#. This goes also the other way.

3 Pipe operator: An Overview

It became a favorite instantly as I made me able to chain together several commands, letting it pass on the parameters from one to the next, and at the same time increased the readability. The operator is is just a triangle symbol `|>`, that has the definition, where **f** is a function and **x** is a parameter.

```
let (▷) x f = f x
```

From my experience with Haskell, I've come accustomed to reading from right to left in these cases, but this swap, made it intuitive and easy to read from left to right and following the logic. It's also extendable to `||>` (two) and `|||>` (tree parameters) to pass on to the next. Her is a an example from the paper comparing F# style chaining with `|>` with Haskell style using F#.

```
[1..10]
   ▷ List.map(fun x → x*x)
   ▷ List.filter(fun x → x % 2 = 0)

-- instead of
List.filter (fun x → x % 2 = 0)
   (List.map (fun x → x * x) [1..10])
```

The pipe operator also comes with a backward pipe operators, `<|`, `<||` and `<|||`. The backward operator is not recommended to use, since it doesn't add readability for the user. This example we use the backward pipe operator. In this case we cannot chain more backward operators, due to the nature of left-to-right associating for the operator that is inherited from OCaml.

```
let antagonist a str = str + " " + a + " your "

let joinMe str1 str2 = str1 + str2
let protagonist = "Luke,"

let plot_twist =
   protagonist
   ▷ antagonist "I'm"
   ▷ joinMe ◁ "father"
(* See a TIE fighter, so cool *)

printfn "%s" plot_twist
(* Luke, I'm your father *)
```

4 Related Work on F#

Form the first HOPL in 1978, to the HOPL IV 2020, there has been a few contributions that link to this paper. At HOPL I, the *History of LISP* [18] is of interest as F# builds in these ideas that was incorporate into LISP. At the HOPL II, there was also a new contribution with regards to LISP, called *The evolution of LISP* [23]. After this we can mention *A history of Haskell: being lazy with class* [13] that has influenced F# design choices and is a very popular functional programming language in academia. HOPL IV, we can mention *The History of Standard ML* [16] that was a direct competitor to F# running on .NET with SML.NET port but never became anything of, and lastly *A History of Clojure* [10] that is a functional dialect of LISP interacting with Java that is sharing some of the same ideas of F#.

Related work that paved the road for F# to integrate with .NET is the authors work on .NET Generics. [24]. Besides this, fsharp.org is a good starting place for more information about the language itself, community and many other resources. One part of this website is Academic Papers page that contains a list of academic publications that could be of interest for the reader to study up on. Furthermore you can follow their work on their GitHub account github.com/fsharp.

5 Tool Support for F#

The great thing about F# is that is runs everywhere the .NET SDK [8] is installed. This means you can use `dotnet` too that comes with the .NET SDK to create projects. But this is not the only way to use F#. It is easy to follow F# guides on Microsoft's web pages [4] with F# Scripting (*.fsx*) and use either VSCode [20] or your favorite editor. VSCode supports the community created Ionide extention [14] that gives good language support for F#. The REPL that follows .NET SDK, I found not useful and I would stay away from it, due to its cumbersome addition of double `;;` for commands and other quirks from the OCaml notation. I mainly used VSCode with Ionide, but I will also highly recommend JetBrains Rider [15] with .NET development.

6 Personal Experience

In the paper, I really enjoyed how Don Syme took us through the history from the early 70's of FP to the inspiration for how and why F# was created. It was exciting to see what inspired and what other languages influenced Don before he created this language. My limited use of the language has shown me a positive side of the F# language, though I struggle some with things, when moving from Haskell world. I look forward to try to incorporate this language into my work, and here I believe Units-of-Measure is a big thing we could benefit from.

One of the things I struggle with in F# was getting going, F# Scripting is one thing, modules and projects are another. Why can't we just use them straight up as we can in Haskell? Since C# put a lot of restrictions on how they did things in F#, I'm curious to ask, how the author would have seen F#'s if it wasn't so tied to C# and their development? Also the author mentions that `Span` helped iron out some minor problems since F# 2.0. Are there other aspects of C# and/or .NET that is hindering F# move in the direction that would make F# more adaptable and usable?

References

[1] *Abstract syntax tree*. In: *Wikipedia*. Page Version ID: 1016693387. Apr. 8, 2021. URL: https://en.wikipedia.org/w/index.php?title=Abstract_syntax_tree&oldid=1016693387 (visited on 04/25/2021).

[2] Phillip Carter. *Active Patterns - F#*. URL: https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/active-patterns (visited on 04/25/2021).

[3] Phillip Carter. *Code Quotations - F#*. URL: https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/code-quotations (visited on 04/25/2021).

[4] Phillip Carter. *Get started with F#*. URL: https://docs.microsoft.com/en-us/dotnet/fsharp/get-started/ (visited on 04/23/2021).

[5] Phillip Carter. *Units of Measure - F#*. URL: https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/units-of-measure (visited on 04/25/2021).

[6] *Don Syme*. In: *Wikipedia*. Page Version ID: 1003588951. Jan. 29, 2021. URL: https://en.wikipedia.org/w/index.php?title=Don_Syme&oldid=1003588951 (visited on 04/21/2021).

[7] Jon Douglas. *What is NuGet and what does it do?* URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget (visited on 04/29/2021).

[8] Tom Dykstra. *.NET SDK overview*. URL: https://docs.microsoft.com/en-us/dotnet/core/sdk (visited on 04/25/2021).

[9] *Funksjonell programmering / Functional Programming*. University of Bergen. URL: https://www.uib.no/en/course/INF122 (visited on 04/21/2021).

[10] Rich Hickey. "A history of Clojure". In: *Proceedings of ACM on programming languages* 4 (HOPL 2020). Publisher: ACM, pp. 1–46. ISSN: 2475-1421. DOI: 10.1145/3386321.

[11] *Hindley–Milner type system*. In: *Wikipedia*. Page Version ID: 1016004296. Apr. 4, 2021. URL: https://en.wikipedia.org/w/index.php?title=Hindley%E2%80%93Milner_type_system&oldid=1016004296 (visited on 04/21/2021).

[12] *HOPL IV*. URL: https://hopl4.sigplan.org/ (visited on 04/21/2021).

[13] Paul Hudak et al. "A history of Haskell: being lazy with class". In: HOPL III. ACM, 2007, pp. 12–1–12–55. ISBN: 978-1-59593-766-7. DOI: 10.1145/1238844.1238856.

[14] *Ionide*. URL: https://ionide.io/ (visited on 04/21/2021).

[15] Jetbrains. *Rider: The Cross-Platform .NET IDE from JetBrains*. JetBrains. URL: https://www.jetbrains.com/rider/ (visited on 04/25/2021).

[16] David MacQueen, Robert Harper, and John Reppy. "The history of Standard ML". In: *Proceedings of ACM on programming languages* 4 (HOPL 2020). Publisher: ACM, pp. 1–100. ISSN: 2475-1421. DOI: 10.1145/3386336.

[17] *Mars Climate Orbiter*. In: *Wikipedia*. Page Version ID: 1016599571. Apr. 8, 2021. URL: https://en.wikipedia.org/w/index.php?title=Mars_Climate_Orbiter&oldid=1016599571 (visited on 04/25/2021).

[18] John McCarthy. "History of LISP". In: *SIGPLAN notices* 13.8 (1978). Publisher: ACM, pp. 217–223. ISSN: 0362-1340. DOI: 10.1145/960118.808387.

[19] *Metaprogramming*. In: *Wikipedia*. Page Version ID: 1017521092. Apr. 13, 2021. URL: https://en.wikipedia.org/w/index.php?title=Metaprogramming&oldid=1017521092 (visited on 04/25/2021).

[20] Microsoft. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (visited on 04/25/2021).

[21] *OCaml – OCaml*. URL: https://ocaml.org/ (visited on 04/25/2021).

[22] *Programmeringsspråk / Programming Languages*. University of Bergen. URL: https://www.uib.no/en/course/INF222 (visited on 04/21/2021).

[23] Guy L. Steele and Richard P. Gabriel. "The evolution of Lisp". In: *SIGPLAN notices* 28.3 (1993), pp. 231–270. ISSN: 0362-1340. DOI: 10.1145/155360.155373.

[24] Don Syme. "ILX: Extending the .NET Common IL for Functional Language Interoperability". In: *Electronic Notes in Theoretical Computer Science*. BABEL'01, First International Workshop on Multi-Language Infrastructure and Interoperability (Satellite Event of PLI 2001) 59.1 (Nov. 1, 2001), pp. 53–72. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80453-0. URL: https://www.sciencedirect.com/science/article/pii/S1571066105804530 (visited on 04/21/2021).

[25] Don Syme. "The early history of F#". In: *Proceedings of ACM on programming languages* 4 (HOPL 2020). Publisher: ACM, pp. 1–58. ISSN: 2475-1421. DOI: 10.1145/3386325.

[26] Philip Wadler. "Why no one uses functional languages". In: *SIGPLAN notices* 33.8 (1998). Place: New York, NY Publisher: ACM, pp. 23–27. ISSN: 0362-1340. DOI: 10.1145/286385.286387.