A Preliminary Survey of Functional Programming

Microsoft Technical Report MSR-TR-2010-147

Caitlin Sadowski
Microsoft Research
supertri@cs.ucsc.edu

Daan Leijen
Microsoft Research
daan@microsoft.com

Abstract

Functional programming has had a profound impact on the development of mainstream languages such as C# or Java. We wanted to get a better sense of developer's perceptions of functional programming, and also better understand which functional programming concepts are useful to developers. This paper reports the results of a preliminary survey on this topic.

1. Introduction

We sent out a small probe survey to a selection of 100 randomly selected developers working at Microsoft in September of 2010. We asked five general questions about functional programming:

1. Tell me about your experience with functional programming.
2. If you were to describe functional programming to a programmer who is unfamiliar with the term, how would you explain what the term means?
3. What is functional programming good for?
4. What are the problems with functional programming?
5. Which concepts or techniques do you associate with functional programming that exist in the languages which you are most familiar with?

In total, 19 developers responded to the probe. In this paper, we present the responses from our five question survey:

- In Section 2 we present the responses from developers completely unfamiliar with functional programming.
- In Section 3 we present the full responses from developers who were familiar with the term.

2. Non Responses

Of the 19 developers who responded to our survey, 5 of them were not familiar with the term "functional programming".

1. I actually had to look up what functional programming was so I guess I really don't have much experience. To answer the rest of your questions I would have to use the definitions I found on the web so I don't think that would be much help to your project.
2. What is functional programming?
3. I've never really used functional programming, not sure what a concise definition is and not sure why I would use it. It's not really on my radar.
4. I'm sorry I'm not familiar with functional programming.
5. I have never used functional programming other than a part of a single class in college. So I'm not really knowledgeable enough to answer the questions.

3. Survey Responses

Of the 19 developers who responded to our survey, 14 of them went on to answer the specific questions. In the following five subsections, we present responses for each of the five survey questions.

3.1 Tell me about your experience with functional programming.

1. I've written programs in ML and Haskell. These were all either for academic or recreational purpose. I've never written any functional programs professionally.
2. SQL: Works great. Excel: Great for analyzing financial stuff (or really anything with a lot of numbers) Scheme: Used in college 10 years ago, can't remember what for.
3. I have experience building a functional language.
4. - learned Lisp, ML, Haskell etc. in undergraduate programming language classes immediately fell in love with the strongly type functional languages (ML, Haskell, etc.)
   - used F# and OCaml extensively for graduate programming language class
   - used F# occasionally for a few other things / classes
   - read a lot of PLDI/POPL/ICFP papers that talk in terms of functional languages
   - use functional style in C# quite regularly (eg. building OCaml-style LINQ operators)
5. I have done some work at school in using lambda calculus. used schema for some projects.
6. I learned SML-NJ at college, which was the first I had ever seen or heard of functional languages. I haven't done anything with functional languages since until the last few weeks, where I've investigated F# because of its fsyacc and fslex tools for parsing.
7. No practical experience with functional languages except playing a bit with LISP a while ago. Sometimes used functional programming ideas (building from a set of no-side-effect modules).
8. Work through samples in comp sci magazines.
9. Studied in college.
10. Had a couple classes on it in school, a while ago. We used Scheme. Haven't done much with it sense.

11. My only experience with functional programming was in college, in my compilers class, where we used SML to write a compiler.
12. Took a class on LISP in college.
13. I mostly use C# and I don't use any pure functional languages, so my day-to-day experience with functional programming is limited to the features introduced in C# 3.0. I also have limited experience with Python, and a parallel programming language called ZPL, both of which use some functional concepts.
14. I've not used it at work.

3.2 If you were to describe functional programming to a programmer who is unfamiliar with the term, how would you explain what the term means?

1. Functional programing is where you program by definition and let the language worry about the machinery. Primarily this is done via recursion. You declare the base case and what the results of the base case are, and then you declare the relationship between the general case and a smaller version of the general case. For example:

$$factorial(1) = 1$$

$$factorial(n) = multiply(n, factorial(n-1))$$

2. Functions don't have side effects. Playing with immutable objects.
3. Do you remember high school algebra? Well, it is like that. No statements, no ordering, only functions. Something like bunch of Excel formals.
4. - Focus on declarative programming (what you want, not how to get it)
   - Algorithms often more closely match their mathematical description
   - Often more concise and easier to reason formally about the behavior of
   - Can be challenging to reason about, especially for someone unused to the functional style
5. Functional programming is a programming methodology where functions do not affect global state and all state that a function operates on is passed as parameters and return values
6. Functional languages allow you to use functions as values. The ability to pass functions to other functions gives the ability to program in a more generic way with more code reuse, in a way that can be very elegant.
7. Using only building blocks with no side effects.
8. Functional Programming is analogous to declarative programming with strict type & bounds checking.
9. Assuming that the person only expected a 1-2 minute explanation and assuming that I had to say what comes to mind without previous investigation, I would say the following: I would say it is another programming paradigm. I would explain that it is based on 2 core ideas, lazy evaluation and lack of global state. I would explain that lazy evaluation can be exemplified in a situation where we have a stream of data, the next item only needs to be generated/computed when asked. The same way, when you are consuming the result of a function then, if it returns a list, the next item only needs to be computed when asked. And I would explain that the lack of global state can be achieved by always passing the necessary state via the parameters of the function call (and parameters are passed by copy, not by reference).
10. I'd say you describe what you want the program to do, rather than how to do it.
11. In functional programming, everything returns a value; there are no void methods like in procedural programming. Honestly, I'm not sure beyond that; I'd have to look it up online.
12. Model of programming, which encapsulates the data and the function operating on it in one entity. Allows to express recursive algorithms and reason theoretically about their correctness, which is difficult to do (not sure if at all possible?) with iterative languages like C++. (I know this is a lame explanation, but I have never stopped and thought about a definition).
13. Functional programming is different from imperative programming, which is the model that most programmers are familiar with. When writing an imperative program, the programmer specifies a list of steps that the computer must carry out in order to produce a result. When writing a functional program, the steps are often not explicit. For example, a mathematical equation could be used to specify a desired result, without specifying how the computer should evaluate the equation. Another difference between imperative and functional programming is that functional programming avoids side effects (e.g., changing a variable's value after an initial assignment).
14. Functions cannot have side effects (i.e. global state). Functions can use the caller's stack.

3.3 What is functional programming good for?

1. Functional programming is good for succint and elegant writing of algorithms, especially recursive ones. It is usually easier to write a correct algorithm in a functional language than an imperative one.
2. Research? Building provably (at least the software) reliable systems.
3. On their own, nothing. When you combine it with other technologies then plenty. For instance the pure nature of a functional language can be a great help and at the same time could be called bad news.
4. - More careful control of side-effects, e.g. reasoning precisely about concurrency; transactional memory is much easier to introduce in Haskell than C#
   - Implementing mathematical algorithms at a higher level of abstraction
   - Many functional languages have strong type systems, which help make it easier to write correct programs (but really these are somewhat orthogonal)
   - Managing complexity of large applications through stronger abtractions e.g. once you test "bool list", you can be assured that "int list" etc. also works (since 'a list can't possibly know the difference)
5. Wherever a functionality could be purely expressed as f(X), where X = set of params. Most notably, mathematical systems.
6. It lends itself well to creating parsers. I'm not sure when else it might be preferred.
7. Very beneficial for concurrent programming. May be good for creating very compact programs.
8. Analysis tools running in constrained/controlled environments.
9. Parallel programming mostly because of the simplicity achieved by not having to synchronize access to shared (global) resources. It is also good when translating from a mathematical specification into a programming language because the

syntax and semantics of the functional programming constructors resemble what we have in mathematical specifications and therefore it is easier to port from one to another; in this sense, I believe some very sensitive systems have favored the implementation via functional languages because those systems have had the requirement of first being proved correct via a mathematical specification (I think that was the case for some underground train systems in France, for example).

10. I remember it was really good on recursive problems. It was also useful to pass code around as data and to bind values for some parameters and effectively create a new function.
11. Mathematical/scientific problems? Compilers, apparently.
12. Expressing recursive algorithms in their code form, reason about their correctness, automatically prove invariants, etc. Not sure what kind of applications it is good for though.
13. Functional programming can be used for the same purposes as imperative programming (i.e. anything). Because it avoids or prohibits side effects, it can help programmers reduce complexity by limiting where changes in state can occur.
14. Continuation-passing style programming? Not sure.

3.4 What are the problems associated with functional programming?

1. Functional programming is programming without side effects, but most useful programs do need some sort of side effects such as reading user input or writing output. While facilities to handle these exist, they tend to be cumbersome. Handling activities that are inherently sequential, such as any sort of interactive program is also difficult.
2. Seems harder to model the mutable world with immutable objects. Perhaps this is only a perception, but it is what it is. CPUs and Kernels don't seem to be optimized to play well with functional programming, especially immutable types. This is likely due to the costs of erecting protection domains. I could be wrong, but it seems like a system would have to be built from the ground up with functional programming in mind to solve the problem.
3. State, mutability, recursion, caching, ordering of execution, "Time? What time", side effects, etc.
4. - Functional style tends not to be appropriate for perf critical scenarios e.g. in my F# computational biology assignments, I often had to rewrite the hot inner loops in imperative style to avoid GC pressure and enable better optimizations, etc.
   - Can be harder to reason about performance and control flow e.g. takes some skill to get good at ensuring tailcalls, to understand callstacks in a debugger, etc.
   - Strict functional programming, while very powerful, is hard to cope with in practice
   - Perception or reality of being hard to learn
5. State passing between functions is very heavy and the avoidance of global state really complicates writing functions. State passed to functions can be very huge. In addition, even continuations need to be passed as parameters and this makes reasoning about exceptions kind of hard.
6. It's hard to wrap your brain around at first. Also, it can be difficult to understand what the compiler is doing.
7. Totally different paradigm from mainstream; very hard to grasp if it was not your first language.
8. - Leaky abstractions. Since computers and real world don't actually work as functional programming, functional programming must either show procedural constructs or go through horrible hoops to hide them.
   - No "popular science" evangelist. All docs start with "functional programming has its roots in Lambda Calculus and treats computation as mathematical functions which avoid mutable data." (Yeah, ok, that helps)
9. Not very flexible, mostly due to the lack of global state. Not very cognitive, compared to object oriented languages, because real life has agents (objects) execute tasks by carrying state and communicating with each other, and functional programming doesn't incorporate these commonly accepted way of doing things in life (therefore, not very cognitive).
10. I remember it required a different mindset from procedural languages. And that once you grok it, it was quite cool and productive.
11. Seems like it would be unintuitive for a lot of software scenarios. Having never used it outside of college, I don't know by experience what the problems are.
12. Apart from it being non-intuitive to the general thinking about programming and perhaps slow code, I am not sure what else.
13. At this point the main problem with functional programming is that it is less popular and therefore less familiar to programmer than imperative programming. Earlier problems such as poor performance have been solved by modern functional language compilers.
14. Not sure.

3.5 Which concepts or techniques do you associate with functional programming that exist in the languages which you are most familar with?

1. Are you looking for a grab bag of functional stuff? Recursion, tail recursion, cons, functors and first-order functionals. Note: I considered functional programming to be distinct from logical programming (like in prolog) though they share many similarities.
2. I don't really. I'm a C++ developer.
3. - functions == values
   - Currying
   - Monads
   - Side effects
   - Immutable values
   - etc.
4. - Anonymous functions / lambdas
   - Immutable data / side-effect free procedures
   - Recursion
   - Pipeline composition: LINQ in .NET
5. lambda queries such as in LINQ. Exception handling that is based on continuations
6. Passing functions to other functions, currying, functors.
7. Just avoiding side effects (e.g. avoid globals / statics).
8. Recursion (which is inappropriate in a broadly shipping product). High-Order functions (which I associate with the richer types of C# & Java).

9. Delegates in .NET, IEnumerable, lambda expresions in .NET, composition of method calls, functional references in C++, struct types in .NET to pass parameters, yield constucts in .NET to simplify the implementation of IEnumerable classes, etc.
10. I've been using C# mostly the last few years. I know it has gotten some functional programming features in it, but I haven't taken advantage of them and aren't real familiar with them.
11. Perhaps the trend of having all methods return some object so you can string together a bunch of properties/methods, e.g. foo.Concat(bar).Reverse().Add(blah).Transform().etc().
12. Recursion, encapsulation, functors in .NET (similar to the lambda functions).
13. The language I use every day is C#, which added functional capabilities in v3.0. One frequent application of one of these features, lambda expressions, is in using mock objects in unit tests. For example, to instruct a mock object's function to return a particular value, the $\Rightarrow$ lambda operator can be used as follows, where `HasQueryString` is the name of a Boolean function and `cu` is a variable that refers to a mock object containing `HasQueryString`:

    cu.Setup(x $\Rightarrow$ x.HasQueryString).Returns(true);

    Lambdas are also often used in LINQ expressions.
14. Lambda expressions in C#.

4. Conclusions

This report is a preliminary look at developer perceptions of functional programming. We were struck by the range of responses. We also noticed that recent .NET additions (such as LINQ, lambda expressions) have really increased the visibility of functional programming concepts.