What are the primary differences between Haskell and F#? [closed]

Asked 16 years, 3 months ago Modified 10 years, 7 months ago

Viewed 33k times

135

votes

Closed. This question needs to be more focused. It is not currently accepting answers.

Closed 10 years ago.

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

I've searched on the Internet for comparisons between F# and Haskell but haven't found anything really definitive. What are the primary differences and why would I want to choose one over the other?

haskell f# functional-programming language-comparisons

Share

edited May 21, 2014 at 21:22

nicael

19k ● 13 ● 61 ● 91

asked Sep 4, 2008 at 23:24

ljs
37.8k ● 36 ● 109 ● 124

1 The nice thing about F# is that due to its mixed paradigm it provides a better starting point for the OO programmer. He can gradually come up to speed with functional thinking and still fall back on using familiar OO libraries as he would have in C#. Haskell forces the whole functional enchilada on you all at once. – M. Lanza Aug 13, 2014 at 18:07

It's important to note that F# is not a functional programming language. It adopted quite a bit from FP however, so it's possible to find perhaps many similarities, but nevertheless primarily I'd say they're entirely different languages. – MasterMastic Jan 11, 2015 at 4:04

Comments disabled on deleted / locked posts / reviews

5 Answers

Sorted by: Highest score (default)

127 votes

Haskell is a "pure" functional language, where as F# has aspects of both imperative/OO and functional languages. Haskell also has lazy evaluation, which is fairly rare amongst functional languages.

What do these things mean? A pure functional language, means there are no side effects (or changes in shared state, when a function is called) which means that you are guaranteed that if you call f(x), nothing else happens besides returning a value from the function, such as console output, database output, changes to global or static variables.. and although Haskell can have non pure functions (through monads), it must be 'explicitly' implied through declaration.

Pure functional languages and 'No side effect' programming has gained popularity recently as it lends itself well to multi core concurrency, as it is much harder to get wrong with no shared state, rather than myriad locks & semaphores.

Lazy evaluation is where a function is NOT evaluated until it is absolutely necessary required. meaning that many operation can be avoided when not necessary. Think of this in a basic C# if clause such as this:

```
if(IsSomethingTrue() && AnotherThingTrue())
{
    do something;
}
```

If `IsSomethingTrue()` is false then `AnotherThingTrue()` method is never evaluated.

While Haskell is an amazing language, the major benefit of F# (for the time being), is that it sits on top of the CLR. This lends it self to polyglot programming. One day, you may write your web UI in ASP.net MVC, your business logic in C#, your core algorithms in F# and your unit tests in Ironruby.... All amongst the the .Net framework.

Listen to the Software Engineering radio with Simon Peyton Jones for more info on Haskell: Episode 108:

[Simon Peyton Jones on Functional Programming and Haskell]

Share

edited Apr 4, 2014 at 14:32

wst
11.8k ● 1 ● 26 ● 41

answered Sep 4, 2008 at 23:54

Xian
76.6k ● 12 ● 45 ● 49

8 Another potential major benefit of F# (depending on the situation) is that it's not lazy, which means that the learning curve for reasoning about space-time behaviour will be much simpler for virtually everyone. – cjs May 19, 2009 at 9:02

5 Direct link to se-radio Episode 108 (Simon Peyton Jones): [se-radio.net/podcast/2008-08/…] – Herrmann Jun 5, 2010 at 15:57

9 Another feature worth pointing out is that a language that enforces pure programming, the compiler has a LOT more freedom for many optimizations. Languages like F# that encourage purity but still allow unchecked impure operations to be dropped inside any block of code lose out on some potential optimizations, as the compiler is forced to assume every call contains necessary side effects. – Ben Nov 17, 2011 at 1:39

12 @JonHarrop: That's a statement about algorithms, which is unrelated to the applicability of compiler optimizations. Enforced purity languages generally allow you to write explicitly impure code if you really need to (if nothing else, using an FFI to call C). The compiler can apply code transformations more freely when it isn't bound to preserve the order of (unknown) side effects. In "encouraged purity" languages you write mostly similar code so the same

optimizations would be applicable, only they might invalidate side effects (which aren't there, but the compiler can't assume that). – Ben Jun 11, 2012 at 22:32

3 @JonHarrop When did I claim they were all algorithms? In any case, we're comparing Haskell and F#, not Haskell and C. F# is an "encouraged purity" language, so you often write pure code. All I'm claiming is that the *exact same code* can often be better optimised by the compiler in an "enforced purity" setting, because the F# compiler has to assume that calls have side effects. You're talking about rewriting the code to use a different algorithm (one which depends on side effects); I'm taking about the "optimisability" of pure code in F# vs Haskell. – Ben Jun 12, 2012 at 13:14

53 votes

Big differences:

- Platform
- Object orientation
- Laziness

The similarities are more important than the differences. Basically, you should use F# if you are on .NET already, Haskell otherwise. Also, OO and laziness mean that F# is closer to what you (probably) already know, so it is probably easier to learn.

Platform : Haskell has its own runtime, F# uses .NET. I don't know what the performance difference is, although I suspect the average code is about the same before optimisation. F# has the advantage if you need the .NET libraries.

Object orientation : F# has OO, and is very careful to make sure that .NET classes are easy to use even if your code isn't OO. Haskell has type classes which let you do something like OO, in a weird sort of way. They are like Ruby mixins crossed with Common Lisp generic functions. They're a little like Java/C# interfaces.

Laziness : Haskell is lazy, F# is not. Laziness enables some nice tricks and makes some things that look slow actually execute fast. But I find it a lot harder to guess how fast my code will run. Both languages let you use the other model, you just have to be explicit about it in your code.

Minor differences:

- Syntax : Haskell has slightly nicer syntax in my opinion. It's a little more terse and regular, and I like declaring types on a separate line. YMMV.
- Tools : F# has excellent Visual Studio integration, if you like that sort of thing. Haskell also has an older Visual Studio plugin, but I don't think it ever got out of beta. Haskell has a simple emacs mode, and you can probably use OCaml's tuareg-mode to edit F#.
- Side effects : Both languages make it pretty obvious when *you* are mutating variables. But Haskell's compiler also forces you to mark side effects whenever you use them. The practical difference is that you have to be a lot more aware of when you use libraries with side effects as well.

Share

answered Sep 15, 2008 at 15:33

Nathan Shively-Sanders
18.4k ● 4 ● 49 ● 56

1 see for performance. Not sure how legit it is – nawfal Feb 18, 2013 at 22:44

---

35 votes

F# is part of the ML family of languages and is very close to OCaml. You may want to read this discussion on the differences between Haskell and OCaml.

Share

answered Sep 5, 2008 at 0:22

Mark Cidade
99.8k ● 33 ● 229 ● 237

---

33 votes

A major difference, which is probably a result ofthe purity but I less see mentioned, is the pervasive use of monads. As is frequently pointed out, monads can be built in most any language, but life changes greatly when they are used pervasively throughout the libraries, and you use them yourself.

Monads provide something seen in a much more limited way in other languages: abstraction of flow control. They're incredibly useful and elegant ways of doing all sorts of things, and a year of Haskell has entirely changed the way I program, in the same way that moving from imperative to OO programming many years ago changed it, or, much later, using higher-order functions did.

Unfortunately, there's no way in a space like this to provide enough understanding to let you see what the difference is. In fact, no amount of writing will do it; you simply have to spend enough time learning and writing code to gain a real understanding.

As well, F# sometimes may become slightly less functional or more awkward (from the functional programming point of view) when you interface with the .NET platform/libraries, as the libraries were obviously designed from an OO point of view.

So you might consider your decision this way: are you looking to try out one of these languages in order to get a quick, relatively small increment of improvement, or are you willing to put in more time and get less immediate benefit for something bigger in the long term. (Or, at least, if you don't get something bigger, the easy ability to switch to the other quickly?) If the former, F# is your choice, if the latter, Haskell.

A couple of other unrelated points:

Haskell has slightly nicer syntax, which is no suprise, since the designers of Haskell knew ML quite well. However, F#'s 'light' syntax goes a long way toward improving ML syntax, so there's not a huge gap there.

In terms of platforms, F# is of course .NET; how well that will work on Mono I don't know. GHC compiles to machine code with its own runtime, working well under both Windows and Unix, which compares to .NET in the same way, that, say, C++ does. This can be an advantage in some circumstances, especially in terms of speed and lower-level machine access. (I had no problem writing a DDE server in Haskell/GHC, for example; I don't think you could do that in any .NET language, and regardless, MS certainly doesn't want you doing that.)

Share

edited Dec 28, 2012 at 16:03
NlightNFotis
9,785 ● 6 ● 45 ● 67

answered May 16, 2009 at 3:38
cjs
27.1k ● 10 ● 91 ● 110

2
votes

Well, for one I'd say a main advantage is that F# compiles against the .NET platform which makes it easy to deploy on windows. I've seen examples which explained using F# combined with ASP.NET to build web applications ;-)

On the other hand, Haskell has been around for waaaaay longer, so I think the group of people who are real experts on that language is a lot bigger.

For F# I've only seen one real implementation so far, which is the Singularity proof of concept OS. I've seen more real world implementations of Haskell.

Share

answered Sep 4, 2008 at 23:28
Erik van Brakel
23.8k ● 3 ● 53 ● 66

4 F# has already had several major success stories (Halo 3, AdCenter, F# for Visualization) that dwarf the nearest thing Haskell has to a success story (Darcs). – J D Nov 6, 2008 at 2:20

Worth pointing out that there are downside for this, of course. (CLR limitations) – MasterMastic Apr 14, 2014 at 11:44