Functional-first Programming in an Information-Rich World

Dr Kenji Takeda

Microsoft Research Connections
Functional-first programming → Information-rich programming
Recurring Problems in Software

- Getting things done
- Efficiency
- Correctness
- Complexity
What’s the Need?

Developers delivering correct, efficient software, on-time

This is the set of problems that F# helps solve
Observation #1

At the core of every functional-first language is this:

simple, correct, robust code for complex problems
Observation #2

A highly interoperable language allows rapid, non-intrusive deployment and integration of components.

Functional code is a part of a larger solution. Your code can be rapidly integrated and deployed.
Observation #2 cont.

Interoperable languages remove entire phases from the software development process

No R → C#
No Mathematica → C++
Observation #3

Strongly-typed functional languages maintain efficiency

comparable to C# and Java, and sometimes C++
Observation #4

Strongly-typed functional languages help analytical programmers tackle more complex problems
How Functional-first Helps

Simple, correct, robust code

Interoperability eliminates entire phases

Strong typing gives efficiency

Analytical developers empowered to solve complex problems
What is F# and why should I care?
F# is...

...a practical, functional-first programming language that allows you to write simple code to solve complex problems.
F# and Open Source

F# 2.0 compiler+library open source drop

Apache 2.0 license

www.tryfsharp.org

http://blogs.msdn.com/dsyme
Simple code,
Strongly typed
type Command = Command of (Rover -> unit)

let BrakeCommand =
    Command(fun rover -> rover.Accelerate(-1.0))

let TurnLeftCommand =
    Command(fun rover -> rover.Rotate(-5.0<degs>))
Simplicity: Functional Data

```csharp
let swap (x, y) = (y, x)

let rotations (x, y, z) = [(x, y, z); (z, x, y); (y, z, x)]

let reduce f (x, y, z) = f x + f y + f z
```

```fsharp
let swap (x, y) = (y, x)

let rotations (x, y, z) = [(x, y, z); (z, x, y); (y, z, x)]

let reduce f (x, y, z) = f x + f y + f z
```
The Big Trends

THE WEB   MULTICORE   DATA
Parallel I/O

Async.Parallel [ httpAsync "www.google.com"
                httpAsync "www.bing.com"
                httpAsync "www.yahoo.com" ]

|> Async.RunSynchronously
Async.Parallel [ for i in 0 .. 200 -> computeTask i ]

|> Async.RunSynchronously
Units of Measure
1985

Mirror on underside of shuttle

Big mountain in Hawaii

SDI experiment: The plan
1985

SDI experiment:
The reality
Attention All Units, Especially Miles and Feet!

Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. The astronauts reported seeing the bright-blue low-power laser beam emanating from the top of Mona Kea, but the experiment failed because the shuttle’s reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot 10,023 feet above sea level on Mona Kea; that number was supplied to the crew in units of feet, and was correctly fed into the onboard guidance system -- which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot 10,023 nautical miles above sea level. The San Francisco Chronicle article noted that “the laser experiment was designed to see if a low-energy laser could be used to track a high-speed target about 200 miles above the earth. By its failure yesterday, NASA unwittingly proved what the Air Force already knew -- that the laser would work only on a ‘cooperative target’ -- and is not likely to be useful as a tracking device for enemy missiles.” [This statement appeared in the S.F. Chronicle on 20 June, excerpted from the L.A. Times; the NY Times article on that date provided some controversy on the interpretation of the significance of the problem.] The experiment was then repeated successfully on 21 June (using nautical miles). The important point is not whether this experiment proves or disproves the viability of Star Wars, but rather that here is just one more example of an unanticipated problem in a human-computer interface that had not been detected prior to its first attempted actual use.
Metric mishap caused loss of NASA orbiter

September 30, 1999
Web posted at: 4:21 p.m. EDT (2021 GMT)

In this story:

Metric system used by NASA for many years
Error points to nation's conversion lag

By Robin Lloyd
CNN Interactive Senior Writer

(CNN) -- NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.
let EarthMass = 5.9736e24<kg>

// Average between pole and equator radii
let EarthRadius = 6371.0e3<m>

// Gravitational acceleration on surface of Earth
let g = PhysicalConstants.G * EarthMass / (EarthRadius * EarthRadius)
Formalizing an Extensional Semantics for Units of Measure

Andrew J. Kennedy
Microsoft Research Cambridge
akenn@microsoft.com

Bugs caused by units-of-measure errors can have catastrophic consequences, the most famous of which was the loss in 1999 of NASA’s Mars Climate Orbiter probe [8], caused by a confusion between newtons (the SI unit of force) and lbf (the Imperial unit).

Many researchers have proposed preventing such errors at development time by type-checking [5, 6] or by static analysis [4, 2]. The former approach is exemplified by the Fortress programming language [1] and a recently-prototyped extension to F# [9]. Here, numeric types are parameterized by units, so float<"m"/"s"^2> represents an acceleration, and functions can be polymorphic in units, so float<"m"/"s"> -> float<"m"/"s"^2> is the type of fun \(x \rightarrow x^2\).

What is a "semantics" of units? An intensional approach would be to tag run-time values with their units, and then show that for type-correct programs the rules governing units are not broken at run-time. In contrast, we claim that the essence of unit correctness is the invariance of program behaviour under scaling: compare the well-known invariance of physical laws under scaling. Many consequences flow from this extensional interpretation: for example, rather than model unit expressions as abstract syntax trees, they are modelled as maps from unit variables to integers, so extensional equality is the right equality for units. Likewise, substitutions are modelled as homomorphisms over unit expressions.

Experience

As is typical with mechanization, a number of techniques were attempted before settling on the clearest approach. The general direction has been of increasing abstraction, preferring an abstract algebraic approach to a syntactic one. For example, rather than model unit expressions as abstract syntax trees, they are modelled as maps from unit variables to integers, so extensional equality is the right equality for units. Likewise, substitutions are modelled as homomorphisms over unit expressions.

Interested in units of measure?

Kennedy, WMM 2008

search for “kennedy units”
Examples and Case Studies
Example - power company

I have written an application to balance the national power generation schedule ... for an energy company.

...the calculation engine was written in F#.

The use of F# to address the complexity at the heart of this application clearly demonstrates a sweet spot for the language ... algorithmic analysis of large data sets.

Simon Cousins (Eon Powergen)
Example power company  
Interoperation ... Seamless. The C# programmer need never know.

Units of measure ... a huge time saver...it eradicates a whole class of errors...

Exploratory programming ... Working with F# Interactive allowed me to explore the solution space more effectively.

Unit testing ... a joy to test. There are no complex time-dependent interactions to screw things up....

Parallelism ... The functional purity makes it ripe for exploiting the inherent parallelism in processing vectors of data.

Code reduction ... Vectors and matrices...higher order functions eat these for breakfast with minimal fuss, minimal code. Beautiful.

Lack of bugs... Functional programming can feel strange... Functional programming can feel strange. Once the type checker is satisfied that's often it, it works.
Example - Biotech

...F# rocks - building algorithms for DNA processing and it's like a drug. 12-15 at Amyris use F#... A complete genome resequencing pipeline with interface, algs, reporting in ~5K lines and it has been incredibly reliable, fast and easy to maintain. A suffix tree in 150 lines that can index 200,000 bases a second ;)

F# v. Python: F# has been phenomenally useful. I'd be writing a lot of this in Python otherwise and F# is more robust, 20x - 100x faster to run and faster to develop.

Units of measure: I started labelling the coordinates as one or zero based and immediately found a bug where I'd casually mixed the two systems. Yay F#!
Example - F# in Advertisement Ranking & Rating @ Microsoft

Around 95% of the code in these projects has been developed in F#.

- **F# allowed for rapid development of prototypes**, and thus also rapid verification or falsification of the underlying mathematical models.
- **Complex algorithms**, for example to compute Nash equilibria in game theory, **can be expressed succinctly**.
- **Units of measure** reduced the chance of errors dramatically: Prices, probabilities, derivatives, etc. can already be kept apart at compile time.
How Functional-first Helps

Simple, correct, robust code

Interoperability improves time-to-market

Strong-typing gives efficiency

Analytical developers empowered to solve more complex problems
Lesson: Combining with the right tools is key
**Examples**

<table>
<thead>
<tr>
<th><strong>A mathematical model</strong></th>
<th>• F# + existing C++ components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A trading engine</strong></td>
<td>• Oracle + F# (Server) + F#/C# (Silverlight)</td>
</tr>
<tr>
<td><strong>A calculation engine with GPU execution</strong></td>
<td>• F# + FCore Math</td>
</tr>
<tr>
<td><strong>A scalable web service</strong></td>
<td>• Data Services + F# + ASP.NET</td>
</tr>
<tr>
<td><strong>A scalable big-data service</strong></td>
<td>• F# + Hadoop + ServiceStack</td>
</tr>
<tr>
<td><strong>A Web 2.0 startup</strong></td>
<td>• F# (Server) + SQL Server + DataFeeds + ASP.NET + F# (WebSharper) + HTML5</td>
</tr>
</tbody>
</table>
### Recent Developments in F# @ Microsoft

<table>
<thead>
<tr>
<th>F# 3.0</th>
<th>• queries, powerful data integration, better tooling, portable libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>F# + Azure</td>
<td>• for scalable service programming</td>
</tr>
<tr>
<td>F# + Azure Hadoop</td>
<td>• for scalable big-data programming</td>
</tr>
<tr>
<td>F# + Azure Cloud Numerics</td>
<td>• for scalable math programming</td>
</tr>
</tbody>
</table>
Information-rich programming
“We live in an Information Society”
The developer’s perspective

- Languages do not integrate information
  - Non-intuitive
  - Not simple
  - Disorganised
  - Static
  - High friction
A Big Problem
New thinking required for languages

• Bringing information into the language
• Solution: **Type Providers**
• Why F#?
  – Control of the language
  – Type inference
  – Strong tooling
  – Interoperability
  – LINQ
  – Open architecture
A Type Provider is....

“A compile-time component that provides a computed space of types and methods on-demand ...”

“A compiler plug-in...”

“An adaptor between data/services and the .NET type system...”
Note: F# still contains no data

Open architecture

You can write your own type provider
Intellisense for Data

Open Climate Data Meeting on January 31st

Join us at the World Bank for a discussion about AppsForClimate, and using open data to address the challenges of climate change.
// F# World Bank Type Provider demo (data.worldbank.org)

// Reference the provider for the World Bank....
#r "..\WorldBank.TypeProvider\bin\debug\WorldBank.TypeProvider.dll"
#load @"FSharpChart.fsx"
open System.Drawing
open Samples.Charting

// Access the data....

/// Seq.sortBy fst |> FSharpChart.Line

// Plot debt of different countries in a single chart using nicer chart style

let countries =
  WorldBank.Countries.India; WorldBank.Countries.UnitedKingdom;
  WorldBank.Countries.Iceland; WorldBank.Countries.Switzerland;

val dashGrid : Grid
val it : ChartTypes.CombinedChart = (Chart)
Complex data
open System

#r "Microsoft.Research.Science.Data"
open namespace Microsoft.arch.Science.Data.Imperative

// referencing SDS type provider
#r @"C:\Users\kenjitak\Documents\Projects\F#\Demos\Data.SDS\Data.SDS\bin\Debug\Data.SDS.dll"

// starting DataSet Viewer connected to the 'view' DataSet
type viewType=Data.SDS.DataSet"c:/Demo/template.csv?openMode=readOnly"

let view = viewType(DataSet.OpenSharedCopy("c:/Demo/template.csv"))
view.untyped().SpawnViewer("c:/Demo/Demo.dsvx")

// high res grid of benchmark monthly mean temperature for 1960-1990 from Climate Research Unit
type CRU=Data.SDS.DataSet"C:/Demo/grid_10min_tmp.nc?openMode=readOnly"
let cru=CRU()

// select area of British isles and current month
Programming the web

Open API Timeline

105 APIs 352 601 1,116 1,628 2,647 4,678
Type Providers: Applications

- ...web data
- ...data markets
- ...network management
- ...a spreadsheet
- ...web services
- ...CRM data
- ...social data
- ...SQL data
- ...XML data
In Summary – Functional-First Languages

- Functional-first languages deliver real value
- Rapid, correct development is central
- Parallelism a bonus
- F.P. as a recruitment strategy: languages are important, people even more so
In Summary – F#

Improved time-to-market for analytical components

Ready for supported use in VS2010 + VS11

A bright future ahead for web/data/cloud

Code correctness, efficiency and interoperation in the modern enterprise
Learn more at
F# Tutorial Session: Wednesday@1700

http://fsharp.net
www.tryfsharp.org