Making Magic with F#
Type Providers

Dr Kenji Takeda (@ktakeda1)
Microsoft Research

#fsharp
Proposition 1
We are living through an information revolution
The Information Revolution

Open API Timeline

- 2000: 105 APIs
- 2002: 352 APIs
- 2003: 601 APIs
- 2005: 1,116 APIs
- 2006: 1,628 APIs
- 2007: 2,647 APIs
- 2008: 4,678 APIs
- 2010: 6,432 APIs
- 2011: 10,537 APIs

- 2012
- 2013
Proposition 2
Our programming languages are information-sparse
Proposition 3
This is a big problem

especially for statically typed languages
(Java, C#, F#, VB, ...)

We need to bring information into the language...

At internet-scale, strongly tooled, strongly typed
But before we get into that...
Part 1

Functional-first Programming and F#
F# is free, open source, cross platform, independent

fsharp.org

Microsoft contribute to F#, and so do many others
The Visual F# Tools from Microsoft are supported, enterprise-ready and come with Visual Studio.
Understanding the Situation

- Data Engineers: Data Information Services
- Analytical Programmers/Data Scientists: Code Analysis Algorithms Parallel
- Design: Presentation Publication UI
The Recurring Business Situation

“I lead a team developing...”

- Analytical Components
- Data-rich Services
- Analytical Components
- Data-rich Services
- Analytical Components
- ...
The Recurring Business Problems

- Time to Market
- Efficiency
- Correctness
- Complexity
  - for analytical components
Is Time to Market a Problem?

Late Models ➔ Missed market opportunities
  - Financial model

Late Services ➔ Users have gone elsewhere
  - Gaming service

Late Components ➔ Millions evaporate
  - Ad ranking engine
Is Correctness a Problem?

Buggy Models $\rightarrow$ Major risks to institutions

Quant model

Buggy Services $\rightarrow$ Users walk away

Gaming service

Buggy Analytical Components $\rightarrow$ Millions leak away

Ad ranking engine
Is Complexity a Problem?

Intractable Models → Can’t enter markets

Intractable Services → Can’t deliver services

Intractable Analytical Components → Can’t ship
The Recurring Business Problems

- Time to Market
- Efficiency
- Correctness
- Complexity

- for analytical components and services
What’s the Need?

Analytical programmers delivering correct, efficient components in the enterprise, on-time

This is one set of problems that functional-first programming helps solve.
Why?
Observation #1

At the core of every functional-first language is:

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

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

... functional-first code is a part of a larger solution. With F# your code can be rapidly integrated and deployed.
Observation #2 cont.

Interoperable languages remove entire phases from the analytical software development process.

...no R $\Rightarrow$ C#  
...no Mathematica $\Rightarrow$ C++  
...no Excel $\Rightarrow$ Java
Observation #3

Strongly-typed functional-first languages maintain efficiency...as good as C# and Java, and sometimes C++
Observation #4

Strongly-typed functional languages help analytical programmers tackle more complex problems...more time in the domain, less time on nulls and object hierarchies.
Recap – How Functional-first Helps

- Simple, correct, robust code
- Interoperability eliminates entire phases
- Strong-typing gives efficiency
- Analytical developers empowered to solve complex problems
Example #1 (power company)

We 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 #1 (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. ... once the type checker is satisfied that's often it, it works.

Correctness

Time to Market

Correctness

Time to Market

Correctness

Time to Market

Efficiency
Example #1 (Simon Cousins, Energy Sector)

350,000 lines of C# OO by offshore team

The C# project took five years and peaked at ~8 devs. It never fully implemented all of the contracts.

30,000 lines of robust F#, with parallel + more features

The F# project took less than a year and peaked at three devs (only one had prior experience with F#). All of the contracts were fully implemented.

An application to evaluate the revenue due from Balancing Services contracts in the UK energy industry

http://simontcousins.azurewebsites.net/does-the-language-you-use-make-a-difference-revisited/
Example #1 (Simon Cousins, Energy Sector)

Zero
bugs in deployed system

“F# is the safe choice for this project, any other choice is too risky”

An application to evaluate the revenue due from Balancing Services contracts in the UK energy industry

http://simontcousins.azurewebsites.net/does-the-language-you-use-make-a-difference-revisited/
<table>
<thead>
<tr>
<th>Implementation</th>
<th>C#</th>
<th>F#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braces</td>
<td>56,929</td>
<td>643</td>
</tr>
<tr>
<td>Blanks</td>
<td>29,080</td>
<td>3,630</td>
</tr>
<tr>
<td>Null Checks</td>
<td>3,011</td>
<td>15</td>
</tr>
<tr>
<td>Comments</td>
<td>53,270</td>
<td>487</td>
</tr>
<tr>
<td>Useful Code</td>
<td>163,276</td>
<td>16,667</td>
</tr>
<tr>
<td>App Code</td>
<td>305,566</td>
<td>21,442</td>
</tr>
<tr>
<td>Test Code</td>
<td>42,864</td>
<td>9,359</td>
</tr>
<tr>
<td>Total Code</td>
<td>348,430</td>
<td>30,801</td>
</tr>
</tbody>
</table>
**Example #3: F# in Insurance**

I work for a large actuarial company... ...Despite adopting Agile/Scrum ...the usual delays, complications and sometimes ...failures.

We used F#, and quickly created a system which would perform the necessary calculations highly efficiently, in parallel, and with a perfect match to the spreadsheet results.

All of the advantages which are commonly touted for F# do play out in practice. **Immutability, Easy Parallelisation, Expressiveness, Testability, Conciseness, Flexibility, Productivity**

[ Company name omitted ]
Example #4: F# in 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 would be writing a lot of this in Python otherwise and F# is more robust, 20x - 100x faster to run and faster to develop.

Darren Platt, Amyris BioTechnologies
Example #5: F# at Kaggle

At Kaggle we initially chose F# for our core data analysis algorithms because of its expressiveness.

We’ve found ourselves moving more and more of our application...into F#. The F# code is shorter, easier to read, easier to refactor, and, because of the strong typing, contains far fewer bugs.

As our data analysis tools have developed, we’ve seen domain-specific constructs emerge very naturally. As our codebase gets larger, we become more productive.

fsharp.org/testimonials
Example #6: 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.
Example #7: F# for Social Gaming

F# is becoming an increasingly important part of our server infrastructure that supports our mobile and web-based social games with millions of active users. F# first came to prominence in our technology stack in the implementation of the rules engine for our social slots games which by now serve over 700,000 unique players and 150,000,000 requests per day at peaks of several thousand requests per second.

The F# solution offers us an order of magnitude increase in productivity and allows one developer to perform the work that are performed by a team of dedicated developers on an existing Java-based solution, and is critical in supporting our agile approach and bi-weekly release cycles.

Yan Cui, Lead Server Engineer
http://fsharp.org/testimonials
Example #8: F# for Machine Learning at Microsoft

I wrote the first prototype of the click prediction system deployed in Microsoft AdCenter in F# in a few days.

For a machine learning scientist, speed of experimentation is the critical factor to optimize.

Unlike C# and C++, F# was designed for this mode of interaction. Switching to F# was liberating and exhilarating.

The world is moving toward functional programming with good justifications: the code is cleaner and easier to debug in a distributed environment.

Dr. Patrice Simard, Microsoft Distinguished Engineer, fsharp.org/testimonials
Our bids for tendered contracts in quantitative finance are regularly half the price of competitors because of the increased productivity we get from F#.

We are regularly able to deliver correct, robust, performant solutions on-time, which is what our customers value most.

Daniel Egloff, QuantAlea Consulting, Zurich

http://fsharp.org/testimonials
Summary – The Data Agrees

- Simple, correct, robust code
- Interoperability improves time-to-market
- Strong-typing gives efficiency
- Analytical developers empowered to solve complex problems
F# is changing...

“F# is for Windows” → F# runs on many platforms
Overview

F# is changing...

“Microsoft makes F#”

“F# has many contributors”
Overview

F# is changing...

One perspective (Microsoft’s)
http://msdn.microsoft.com

Many perspectives
http://fsharp.org
F# for Android

fsharp.org/use/android
F# for Linux, Mac

fsharp.org/use/linux
fsharp.org/use/mac
F# for iOS

fsharp.org/use/ios
Give your world some F# spice!
Amazon Web Services .NET SDKs

**Supported Services**

**Compute & Networking**
- AWS Direct Connect
- Amazon EC2
- Elastic Load Balancing
- Auto Scaling
- Amazon EMR
- Amazon Route 53
- Amazon VPC

**Storage & Content Delivery**
- Amazon S3
- Amazon Glacier
- Amazon CloudFront
- AWS Storage Gateway
  - AWS Import/Export

**App Services**
- Amazon Elastic Transcoder
- Amazon SQS
- Amazon SNS
- Amazon SES
- Amazon SWF
- Amazon CloudSearch

**Database**
- Amazon DynamoDB
- Amazon RDS
- Amazon Redshift
- Amazon ElastiCache
- Amazon SimpleDB

**Deployment & Management**
- AWS Elastic Beanstalk
- AWS CloudFormation
- Amazon CloudWatch
- AWS Data Pipeline
- AWS Identity and Access Management
- AWS OpsWorks
Azure .NET SDKs

http://www.windowsazure.com
https://github.com/WindowsAzure/

Compute
- Create a web site
- Create a multi-tier app
- Host on a virtual machine
- Customize a domain name
- Publish with TFS

Data Services
- Store data in SQL Database
- Store data in Blobs
- Store data in Tables
- Store data using MongoDB
- Manage SQL Database

App Services
- Send email with SendGrid
- Monitor with New Relic
- Increase perf with caching
- Message between apps
- Authenticate users

SHOW ALL
SHOW ALL
SHOW ALL
Back to the main topic...
The Main Topic
You can easily find out more about...

- F# Basics
- F# for Data Science
- F# for GPUs
- F# for Cloud Data
- F# for Testing
- F# for DSLs
- F# + R
- F# + Excel

F# Deep Data Integration
Data is like water...
Data is like water...

- Everyone needs it. Everyone knows where to get it.
- Nobody is sure where it really came from, or goes to.
- ...really knows its true cost, or true value.
- ...likes to pay for it, or to share it.
- ...knows how much is wasted.
- You might get washed away by it.
- You only find out it was bad after you have drunk it.
Actually these days it’s more like a flood...
The Problem

Our programming tools are data-sparse

getting data into a programming language is tiresome, error prone and boring
We need to bring data into the language...

At internet scale, strongly tooled, strongly typed
Demo
Problem: Integrate all of freebase.com

“as if it were a library”

>40M entities, >1Billion facts, >24,000 types, >65,000 properties
A Type Provider is....

“Just like a library”

“A design-time component that computes a space of types and methods on-demand...”

“An adaptor between data/services and the .NET type system...”

“On-demand, scalable compile-time provision of type/module definitions...”
Theme #1

On-Demand Types = Internet Scalable Magic
Theme #2

Many Data Sources, One Mechanism
All your types are belong to us....
type NorthwndDb =
SqlDataConnection<
ConnectionString = @"AttachDBFileName = 'C:\project\northwnd.mdb"; "

let db = NorthwndDb.GetDataContext()

let customerNames =
query { for c in db do
  where (c.CustomerName = "AlphabeticalListofProducts")
  select c.CustomerName }
```sql
let connectionString = @"Data Source=(LocalDb)\v11.0;Initial Catalog=AdventureWorks"

let query = "
    SELECT TOP(@TopN) FirstName, LastName, SalesYTD
    FROM Sales.vSalesPerson
    WHERE CountryRegionName = @regionName AND SalesYTD > @salesMoreThan
    ORDER BY SalesYTD
"

type SalesPersonQuery = SqlCommandProvider<query, connectionString>
let cmd = SalesPersonQuery()
```
type BankClosure =
  InferRows=10, InferTypes=true, IgnoreErrors=true)

let bankClosureResults = new BankClosure()

// Preview the header row.
let header = bankClosureResults.HeaderRow

for x in bankClosureResults.Data do
  x.
```csharp
1: type Simple = JsonProvider<""
   { "name":"John", "age":94 } 
   ">
2: let simple = Simple.Parse(""
   { "name":"Tomas", "age":4 } 
   ")
3: simple.Age
4: simple.Name
```
type Author = XmlProvider<""""<author name="Paul Feyerabend" born="1924" />""""><
let sample = Author.Parse(""""<author name="Karl Popper" born="1902" />"""")

printfn """"%s (%d)"""" sample.Name sample.Born
Hadoop/Hive

type HadoopData = HiveTypeProvider<"tryfsharp", Port=10000, DefaultTimeo

let data = HadoopData.GetDataContext()

let testQuery1 =
  query { for x in data. do
    select x }

module AbaloneCatchAnaly sis

ExecuteQuery
GetTable
GetTableMetadata
GetTableNames
Host
Port
UserName
abalone
World Bank

```csharp
using Samples.WorldBank;

let data = Samples.WorldBank.GetDataContext();

let results = data.Countries.Select(c => new { Country = c.Name, Value })

<table>
<thead>
<tr>
<th>Country</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>2014</td>
</tr>
<tr>
<td>Albania</td>
<td>2014</td>
</tr>
<tr>
<td>Algeria</td>
<td>2014</td>
</tr>
<tr>
<td>American Samoa</td>
<td>2014</td>
</tr>
<tr>
<td>Andorra</td>
<td>2014</td>
</tr>
<tr>
<td>Angola</td>
<td>2014</td>
</tr>
<tr>
<td>Antigua and Barbuda</td>
<td>2014</td>
</tr>
<tr>
<td>Arab World</td>
<td>2014</td>
</tr>
</tbody>
</table>
Freebase

```csharp
#r @"..\TypeProviders\Debug\net40\Samples.DataStore.Freebase.dll"

open Samples.DataStore.Freebase

// Access the service types using our API key
type Freebase = FreebaseDataProvider<Key=API_KEY>
let ctxt = Freebase.GetDataContext()

ctxt.\'\'Arts and Entertainment\'\'
```

The publishing domain is home to most aspects of the written word -- books, magazines, scholarly papers, academic papers, etc. Most of the data we have imported from Wikipedia, although we are looking into other possible data sources. We encourage feedback for authors, writings, or publications if we're missing information, please see the documentation for instructions.
type NetFlixCatalog = ODataService<"http://odata.netflix.com/Catalog/">

let netflix = NetFlixCatalog.GetDataContext()

netflix.
  Credentials
  DataContext
  Genres
type TerraService = WsdlService<"http://msrmaps.com/TerraService2.asmx?WSDL">

let terraClient = TerraService.GetTerraServiceSoap ()
    let myPlace = new TerraService.ServiceTypes.msrmaps.com.Place(City = "Redm"
    let myLocation = terraClient.ConvertPlaceToLonLatPt(myPlace)
printfn "Redmond Latitude: %f Longitude: %f" (myLocation.Lat) (myLocation
// Pull in stock prices for some tickers then compute returns
let data = [
    for ticker in [ "MSFT"; "AAPL"; "VXX"; "SPX"; "GLD" ] ->
        ticker, getStockPrices ticker 255 |> R.log |> R.diff ]

// Construct an R data.frame then plot pairs of returns
let df = R.data_frame(namedParams data)
R.pairs(df)
Tachyus is a Silicon Valley startup that aims to be “a Data Start-Up for the Oil Industry”. They aim to create an array of sensors and mobile applications to help oil and gas producers better record and analyze their wells. According to the New York Times coverage:

The start-up represents an anomaly of sorts in Silicon Valley. Many new businesses focus on high-technology products for the Internet or green technology, but Mr. Sloss and his co-founders, Paul Orland and Francisco LePort, have instead homed in on the decidedly older and dirtier business of drilling for hydrocarbons.

Last week Tachyus announced that it has raised $6M in funding from a group led by Founders Fund. At the time of the announcements, one of the Tachyus engineers announced that they went from “from zero to product launch in 12 weeks” and “we couldn’t have done it without F#”. Founder Paul Orland commented “we are using 100% F#”
Providing Units of Measure via F#’s Units of Measure

If the metadata contains units...

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissipated</td>
<td>/meteorology/tropical_cyclone/dissipated</td>
</tr>
<tr>
<td>Highest winds</td>
<td>/meteorology/tropical_cyclone/highest_winds</td>
</tr>
<tr>
<td>Lowest Pressure</td>
<td>/meteorology/tropical_cyclone/lowest_pressure</td>
</tr>
<tr>
<td>Damages</td>
<td>/meteorology/tropical_cyclone/damages</td>
</tr>
</tbody>
</table>

let cyclones = data.``Science and Technology``.Meteorology.``Tropical

let topWind = cyclones.``Hurricane``.top

val topWind : float<metre/second>

Full name: Demo.topWind

...then these can be projected into the programming language.
FSharp.Data

[link](https://fsharp.github.io/FSharp.Data)
on NuGet, use it in Visual Studio today
Scalable (meta)data integration into programming is a key challenge of our era

“F# type providers” are a simple, powerful point in the design space

The techniques have many, many applications

People use this “for real” in production F# systems
In Summary

Open, cross-platform, strongly typed, efficient, rock-solid stable

The safe choice for enterprise data programming

Unbeatable data integration

Visual F# - tooling you can trust from Microsoft

http://fsharp.org
Questions?

Give your life an F# edge!

http://fsharp.org