CUFP 2008 Report

This report summarises the talks given at the Commercial Users of Functional Programming workshop, held in Victoria on 26 September 2008. The report includes the talk abstracts, and summary notes taken by the rapporteur during the meeting. Most of the talks are available in video form on the CUFP web site: http://cufp.galois.com. The goal is to give the reader a sense of what went on, rather than to reproduce the full content of talk; for the full details, watch the video.

Rapporteur: Simon Thompson, University of Kent, UK

Invited talk: *Why Microsoft is Investing in Functional Programming.*
Don Syme, Microsoft

Abstract: *Over the last 10 years Microsoft have made investments in programming technologies strongly influenced by functional programming techniques and languages, from generics in C# 2.0, LINQ in .NET 3.5 and futures in the .NET Parallel Extensions Library. Most recently Microsoft has announced that it is bringing F#, a functional/ OO language for .NET, to product quality, as well as continuing its research investments in both Haskell and foundational language techniques.*

This talk will take a look at these developments and why Microsoft is making these investments. We will focus on F# in particular: where does F# fit in the spectrum of Microsoft development tools? What kind of tasks is F# fashioned for? How has F# grown up over the years to be influenced by OCaml, C#, Haskell, Python and other languages? We'll look at these and other questions as we explore one of the many ways in which research and practice have come together over the last 10 years at Microsoft

Key factors in making F# successful within Microsoft: simplicity, economics, fun. This has had the result that F# is being taken to the standard of “product quality”, so that it becomes part of the suite of languages within Visual Studio.

A slogan summarising F#: F# = OCaml core + C#/.NET object model and platform.

Simplicity: give examples in both F# and C#: there’s more noise than signal in the C#; and this comparison often used in MS to persuade users from C# to F#. The usual reasons for FPers: pattern matching, implicit types; “let”, type inference etc. These advantages work in data- and control-rich and symbolic domains: focus on the core strengths (e.g. not presentational aspects, which come from other .NET languages.

F# can be characterised as a puppy: friendly, approachable because “Built in .NET” and can reuse libraries, tools etc.. Fits clearly into a developers’ mindset: gradualism (cf C to C++). It’s the end-to-end experience that makes the difference.

Influential case study of: Ad Ranking: auctioning ads on search sites. Related work: chess skill through time "True Skill Through Time" (on MSR blog):

http://blogs.technet.com/apg/archive/2008/04/05/trueskill-through-time.aspx

Economics (for Microsoft): there are real economies of scale in putting out new languages. Have investment in .NET, libraries, VS, Silverlight, tools, … This is a basis for low cost, high value additions, including functional programming (and lots of others). Support from external bodies (e.g. financial companies) highly influential for the company.

Fun: great set of software engineering tools. e.g. explore components and libraries. Sit down with .NET APIs for any large package, and F# allows people to do exploratory programming against these APIs or DLLs. People want it and like it. People - in certain important domains - more productive with it.

Controlling Hybrid Vehicles with Haskell.

Tom Hawkins, Eaton Corporation

Abstract: *To address environmental concerns and rising fuel prices, Eaton is developing a family of hydraulic hybrid systems to increase fuel economy for heavy duty trucks. Hydraulic hybrids work much the same as their electric hybrid counterparts. Instead of an electric motor and battery, hydraulic hybrids use pumps, valves, and accumulators to capture and return a vehicle's kinetic energy. Most of these components are under direct software control, and, due to the nature of the application, often require a high level of safety. By using functional languages, we are able to intuitively describe safety critical behavior of the system, thus lowering the chance of introducing bugs in the design phase. Our experimental environment uses a Haskell DSL called Atom, which compiles a program of atomic state transition rules into a form that can be flashed onto a vehicle's electronic control unit (ECU). With proper rule scheduling, Atom can transform a multi-task, multi-rate program into a single thread of execution, thereby eliminating the need for run-time scheduling, context switching, and inter-process communication. We coin this "RTOS Synthesis" as it does most of the work of a real-time operating system, but with compile-time guarantees as opposed to run-time exceptions.*

This is real Haskell Garbage Collection: "mark and sweep? no, clump and dump!" as it's using Haskell in control in hydraulic vehicles. Haskell used in Eaton for lots of things; the talk focused on the Atom DSL, which is inspired by Bluespec and STM. Based on notion of atomic state transition rules. Safety: these are dangerous machines; the machines run on the road. Atom automates: multi-rate thread scheduling and synchronisation.

The biggest challenge in doing this: getting FP through the door. It's not in the IT department, but in a department that has to "get things done".

Functions to Junctions: Ultra Low Power Chip Design With Some Help From Haskell.

Gregory Wright, Antiope

Abstract: *Antiope Associates designs custom wireless communication systems. This talk describes how we used Haskell to design a protocol for an ultra-low power radio chip. Haskell played two roles: it was the main language for the simulation system used to design and debug the protocol, and it was used in tools we wrote to verify that the protocol was correctly implemented in silicon.*

The challenges of working with customers and vendors unfamiliar with functional programming will be mentioned, as well as the lessons we can draw from using these techniques in a small company.

"Haskell: the preferred language of highly evolved predators"

Concentrating on the business side rather than the technical in this talk. Antiope = 6 people: ultra low power radios. Label on supermarket shelves to show prices: receives radio from central station. Has to be cheap (v. low profit margin in supermarkets), high throughput (every tag in the store in one shift: can switch between sale price or everyday low price - EDLP - policy), reliable, min 5 year life with coin cell battery.

FP for system architecture, protocol design, hardware design; work with others for many aspects. Crucial problem: protocol: how to trade throughput and latency to get low power. Can't recompile!

Haskell: principal language for system simulator plus numerous utilities tying together design verification and testing. The former was planned, the "glue" role was an unexpected benefit.

Model in Haskell so that can move easily into VHDL: functions -> hardware blocks. Manual translation used here, because of the specialised constraints (e.g. analogue behaviour as well as digital on one chip).

How did FP help? Protocol design really good: HOFs and other abstraction mechanisms. Elegant syntax helps: as someone who doesn't do it all the time, readability crucial.

Glue? Wrote large numbers of parsing and unparsing programs to feed VHDL, testing etc. Nice analogies with circuit design e.g. pipelines "no impedance mismatch".

Good FPers are usually good at breaking problems down into tractable pieces i.e. good at design. Should / can every programmer be a designer? Analogy of the savannah: grazers vs predators.

Minimizing the "Immune Response" to Functional Programming.

David Balaban, Amgen

Abstract: *Functional programming was introduced at Amgen for three main reasons:*

- *To rapidly build software to implement mathematical models and other complex, mathematically oriented applications*
- *Provide a more mathematically rigorous validation of software*
- *To break developers out of their software development rut by giving them a new way to think about software.*

Our experience is that using functional programming reduces the critical conceptual distance between thought/algorithm design and code. In several projects, we have been able to develop code quickly and to verify — to an applied mathematician's satisfaction — the correctness of Haskell code straightforwardly; we have yet to achieve this with more traditional mainstream languages. In several cases, the Haskell code has evolved quickly, matching the pace at which our understanding of the underlying problems has evolved.

We have found that many of the informatics staff are enthusiastic about and inspired by this approach, even though many have not yet begun to apply Haskell in their routine work. On the other hand, those developers who have a more traditional approach to software development may view our prodding toward a more mathematical approach to thinking about algorithms and programming with scepticism. For these staff, our prodding toward a more mathematical approach to thinking about algorithms and programming is viewed as a distraction at best and often produces a genuine intellectual immune response.

We will illustrate the places where functional programming has been useful to us with examples from pharmacodynamics and supply chain management. These will show, for example, how lazy evaluation can greatly simplify the coding of a complex simulation. We will also describe plans to expand the use of functional programming with additional training classes and recruitment strategies that may make it easier to find people with deep functional programming experience and applied mathematics skills.

Products for cancer care: drugs to get you through chemotherapy.

Introducing FP has had varying success: very successful in scientific applications; barely started on use in code validation. Success can be bi-modal: appeals to be best programmers: but others can be resistant.

Multi-disciplinary cooperation: system ↔ differential equations ↔ difference equations ↔ computer program (the Hertz-Rosen methodology). Involves scientists/ physicians, applied mathematicians and computer scientists. Any computing people involved have to be enthusiastic about the underling science and math. A very striking point was made about how there could be feedback from the Haskell model (e.g unfold) can help the math models (e.g. dynamical systems).

What about software validation? Haskell narrows the distance between the models and the algorithms (and the code). Use 50% of the data to tune, and 50% to check the result.

How was the company persuaded? It's in the central simulation group, and he's a VP,so can approve this. Unless you can make a productivity argument you'll fail.

Quantitative Finance in F#.

Howard Mansell, Credit Suisse

Abstract: *Building valuation models for derivative trades requires rapid development of mathematical models, made possible by composition of lower-level model components. We have found that F#, with the associated toolset, provides a unique combination of features that make it very well suited to this kind of development. In this talk, I will explain how we are using F# and show why it is a good match. I will also talk about the problems we have had, and outline future enhancements that would benefit this kind of work.*

30+ programmers - mainly quants - working with F# on a daily basis. Naming contest among their staff about the activity: slogan for what they are doing:

3. F# F'Sure
2; F# Falls Flat
1: Microsoft Finally Delivers Something Functional

Mathematical models of derivatives contracts: especially exotic contracts. Pre-F#: low level computationally intensive routines in C++, exposed as COM objects (pretty stable): on top: Excel, Haskell, C++, C#. In house DLSs, compilers and code generators. After F#: higher up part: F# for composing lower-level functions. Reducing excel and C++. C# used only when can't use F#.

Why F#? 2005: licensing a problem then. Now that it's a product, can look again. Like that it's a functional language: pipelining, pattern matching etc. Robustness. Likes the OO support can call and implement COM objects.Red squiggly lines are a big plus: immediate type checking allows earlier bug fixing. Expressiveness of type system better than most mainstream languages, focus on the correct by construction approach.

Why have financial companies moved into FP when others have not? In all cases it's not part of the IT department, it's part of a trading group. In that context, it's all about value: just need to demonstrate that it's worth doing.

What were the risks identified in adopting F#? The F# timeline could change, and be dependent on that. Would not work properly and could not fix themselves. In fact v. few bugs in the compiler. What happens with numerically intensive code in F#: can avoid this.

Is Haskell Ready for Everyday Computing?

Jeff Polakow, Deutsche Bank

Abstract: *I will talk about my experience using (mostly) Haskell to design and implement the software infrastructure for a small trading group at Deutsche Bank.*

Most of the applications I write deal with such quotidian tasks as acquiring data from external sources, linking up related information from different sources, searching for specific patterns and making data available through a webserver. In addition to outlining my overall system architecture and highlighting some novel aspects of my implementation, I will discuss the various pros and cons, technical and otherwise, of using Haskell in a corporate environment.

Information management is the main task putting together information from various providers, each of which provides partial information.

Why Haskell? Because he can. Chance to put theory into practice … is it good enough? Easiest way for him to be productive: switch from OCaml because of syntax.

System overview: information in and out of databases, with web server as repository for the traders. haskell as the glue that puts it all together: HAppS used as a web server, HDBC for database access (with ODBC component); also a lot of home written stuff, mostly Haskell but some F#.

Among the novelties: Statically typed tables with mini SQL DSL: manipulate tables in memory; generate SQL queries to create a table in memory. Using HLists for the representation. "You get used to 5 page error messages!" The advantages are getting static guarantees, good code reuse, purity allows poorly documented code to be upgraded easily. Some disadvantages: 6.8.3 upgrade was painful. Some libraries don't like XP. Errors / inadequacies of some libraries. Much library documentation is poor.

Everyday computing: HDBC good. Web stuff poorly documented, but some good stuff. ghci as a shell, HSH etc great for scripting. FFI good, but hard to interact with .NET and Java. Is it ready for everyday programming? Yes, if seasoned Haskell programmer, comfortable with lazy / strict tradeoffs, can read library source code, capable of understanding and fixing linker errors … .

Xen and the art of OCaml.

Anil Madhavapeddy, Citrix

Abstract: *XenServer is a virtualization product that offers near bare-metal virtualization performance for virtualized server and desktop operating systems. It includes a comprehensive implementation of the XenAPI, which encapsulates the configuration and deployment of VMs, storage and networking topologies across pools of physical hosts running XenServer. The management tool-stack is written almost entirely in Objective Caml, and is one of the largest systems-level projects written in that language to date.*

In this talk, we will firstly describe the architecture of XenServer and the XenAPI and discuss the challenges faced with implementing an Objective Caml based solution. These challenges range from the low-level concerns of interfacing with Xen and the Linux kernel, to the high-level algorithmic problems such as distributed failure planning. In addition, we will discuss the challenges imposed by using OCaml in a

commercial environment, such as supporting product upgrades, enhancing supportability and scaling the development team.

Technical agenda: interaction of virtualisation and programming language technology.

Xen project, 6 grad students at Cambridge. Xen 3.0 released 2005, open source Python toolstack. XenSource founded in Cambridge and Palo Alto, 2005; 2006 commercial XenServer distribution begins. 2006 hit and run on the Systems Research Group brought in 4 OCaml people, 2 Xen hackers. XenSource bought by Citrix $500m; Dell/ HP ship embedded XenServer, supported by MS in completely transparent way.

How does OCaml come in? First, wrapping up C "hypercalls" and using as a language for composition. XenAPI: all in OCaml: describes all that's in an API. Manages resource pools etc. In summary, across the levels from low to high. OCaml experience:

Pleasure	*Pain*
Modules/ poly variants	Objects
Meta-programming	CamlP4
OMake	OCaml Build
Custom Standard Lib	Community Libs

Successes: development speed; only 1 compiler bug in two years. In two years from nothing to enterprise tool stack.

Next? Porting OCaml directly to Xen, … ; higher level: declarative data centres (e.g. amazon). Integration with F#. Caught up with VMWare in 2 years (vs 10 years).

From OCaml to Javascript at Skydeck.
Jake Donham, Skydeck

Abstract: *Skydeck is a startup software company building a service to help consumers manage their cell phones online. From the beginning we have used OCaml as our primary language for software development.*

A key piece of our system is a Firefox extension that imports cell phone bills and phone usage data from phone carrier web sites. Firefox extensions are written in Javascript; to speed our development we wrote OCamljs, a Javascript back-end to the OCaml compiler. Using OCamljs we get the benefits of OCaml's expressiveness and compile-time typechecking, the use of OCaml-specific tools like OCamlbuild and Camlp4, and easier integration with our OCaml server infrastructure. In particular, OCamljs lets us make and deploy changes very quickly with confidence.

This talk will describe OCamljs and how we use it at Skydeck, and reflect briefly on the wisdom of using an atypical language at a startup.

Tool for managing a person's cellphone data : reads mobile phone call log, and present it back to the user in interesting ways. Get info from provider's website, if you give Skydeck your credentials. Need to do via web API, or using Firefox or XULrunner app.

Most of the system is written in OCaml: bill parsing, web services etc. But the web is Javascript. Written JS back end for OCaml.Why? Type system: correctness. CPS is enforced by type system. Transparent RPC to server. Lots of OCaml tool support

How does OCamljs work? Ocamlc compiler to “lambda” intermediate language; untyped language. Ocamljs translates this to JS. Almost everything comes for free, except objects, as want OCaml objects to be JS objects.

OCaml at a startup? You have a clean slate and so can choose more widely than a larger company. You need tools for productivity; static type checking keeps you on course. Need to hire good people for a startup to succeed: OCaml attracts them.

The Mobile Messaging Gateway, from Idea to Prototype to Launch in 12 Months.

Francesco Cesarini, Erlang Training and Consulting

CTO Erlang Training and Consulting. Largest contract ETC has received, and which allowed the company to double in size in 12 months. Small note on website “in house systems development” but this has been major development. Names of company and programmers changed to protect the innocent.

Project: Mobile Messaging Gateway: optimize for unreliable transportation medium; optimize the data channel; optimize power usage. Other aspects: authentication, provisioning and billing. Tracing, audit logs and statistics. Protocol translations - particular IM and Email backends, such as Google Talk, Yahoo, AOL and others, vary in different ways. Control of users, devices IM/email clients.

Initially did not bid for the project: instead offered to do 2-3 person month prototyping exercise. Concerns about Erlang: throughput? does fault tolerance really work. Gave the company a chance to find out feature implementation times, system throughput, and TCP/IP behaviour of Erlang under pressure. Did 6 weeks work design -> delivery. Getting 500-600 message / second on a Pentium II machine with half a gig of memory: much better than expected. Awarded contract.

Question of waterfall (company) vs. agile (ETC). First delivery in May; 2 weeks late; problem of integrating the different parts of the system. First customer acceptance: changes in priorities, scope etc in June, July and August. Decision to go live on December 24th … major success.

Stress tests: 10,000 transactions / second; 100,000 simultaneously connected users; 5,000 tranactions/sec over 24 hours. This was the major hurdle in time and effort. Took 2-3 months to sort out the stress tests … details in the slides.

The challenge was not technical
- organizational: relations and customer focus;
- building a team as double in size each year.

Did you make money? Yes

Who do you hire? People need to know Erlang. Lots of thesis projects and summer internships. Need now to pull in people who have not gone through internships.

Buy a Feature: An Adventure in Immutability and Actors.

David Pollak, Lift Web Framework

Abstract: *I will discuss the functional programing paradigms that we used to build Buy a Feature, a multi-user, web-based, real-time, serious game. These paradigms include Actors to manage concurrency, event streams as the sole mechanism for gameplay, and various immutable data structures that are composed based on the event streams. I will also briefly touch on the Scala programming language and lift web framework.*

I will then discuss the experience of adding new team members to the project, the kind of defects in the application (hint: none are concurrency related), the experience of adding new features, and a general discussion of how well functional paradigms translate into a real-world web application.

Lift web framework fits on top of Scala. Buy a Feature is online game: negotiations between players: selling features to each other. Spreadsheet-like browser interface. Scala compiles to Java byte code. Good support for Comet and Ajax. Built-in Erlang style actors.

Actors provide excellent concurrency management: event-oriented, asynchronous. Actors in UI and cross session game managers. All events serialised to the database, and so "Tivo-like" replay through the gameplay. Events communicate all state changes. Immutability makes rerunning very nice.

Initial sell: if you want to hire me, you'll have to use lift. Client: "I could do this faster in Rails": client getting interested as he's a geek as a client. Once first code went live (5 weeks) all the questions went away. 100s of simultaneous participants on a single box. Other companies beginning to take an interest in Lift and Scala: they allow for "Exploration Driven Development".

Team integration: initial disbelief over code size: expecting 40k lines, was 1,300 lines! They first attempted to dive below the abstractions: no http-request etc at the programmer level, and they wanted to be able to do this. Eventually they adopted map, fold and filter. Lack of tool support and examples in the wild are speed bumps. Use emacs as primary editor. Need FP expert as mentor for the rest of the team.

Developing Erlang at Yahoo.

Nick Gerakines and Mark Zweifel, Yahoo

Abstract: *Yahoo is no stranger to functional programming languages. It has had significant products in languages like Lisp and Scheme. Somehow Erlang, and other* *function languages, are often overlooked when most developers are researching various problems and systems. This is very unfortunate given the power and flexibility that these languages provide. At Yahoo there are places where functional languages and make a phenomenal difference in the way problems and solutions are approached. This presentation will cover how Erlang, a powerful and flexible functional language, gave us exactly what we needed at a critical time and how it was approached as a production language at Yahoo.*

Yahoo has lots of official languages C/C++, Java, PHP, Perl (deprecated). Unofficial: Ruby, Python, Objective-C (native iPhone developments), Erlang.

Delicious: http://delicious.com/ saving URLs on the basis of tagging, can survey these. Can look at personal and social trends over time. Delicious 2.0 launched July 31st. Complete rewrite: one year. Uses Erlang. Back end: large C++ stack. Ties to several subsystems e,g aka spam, search, etc.

3 use cases for Erlang.
#1 Data migrations from old to new system.
#2 Rolling migrations between both systems running in parallel.
#3 Algorithmics: spotting spam, popularity and so on.
In all cases Erlang provided reliability and greater controllability than other languages e.g. perl. For example, could control system throughput much more easily in Erlang, and could replace cron-based jobs with self-monitoring systems. Also easily supports system O&M.

Complications? only started with Erlang as development started. Hesitation from management and engineers: exploring as develop. Issues
- Erlang is very different, and engineers are very stubborn.
- Not enough mass at Yahoo to make it unquestionable
- Tension was already high … adding a new language didn't make this easier.
The project convinced management: front-line managers are happy, convinced by performance figures.

Ad Serving with Erlang.

Bob Ippolito, Mochimedia

Abstract: *We've been using Erlang at Mochi Media since the 2006 launch of MochiAds, our advertising platform for Flash games. It's the first time any of us have done any production software with a functional programming language, and thus far it's turned out to be a great experience. The success of the MochiAds platform led us to writing many other products (internal and external) on the same technology stack. We believe Erlang to be one of the key reasons why we're able to develop more quickly and scale more easily than our competitors.*

CTO and co-founder. MochiAds is monetization platform for Flash Game ecosystem. API that any Flash game developer can put in their game ... like AdSense from google. MochiBot (Python) fast but not fast enough. Wanted easy multi-node distribution; Python couldn't help. Built prototype in Erlang and very successful.Why Erlang?

performance (which is all about concurrency), concurrency, distribution (for reliability) fault tolerance (avoiding error handling, as it's in the OTP layer).C.

Use Erlang in a whole bunch of systems: MochiAds and MochiBot. All ad targeting in real time using Erlang. Real-time analytics. "Anything that talks on a network it Put as much state data as possible into the URL: pre-compute compressed versions

Lessons learned: network partitioning sucks, open source not always robust. In fact, the only open source that they use their own code. Favourite Erlang features: module reloading in live systems great. Pattern matching is really cool. Binaries are great for arcane file formats. Lightweight processes are good for concurrency and modelling. Syntax is concise but not cryptic.

Discussion … "This isn't being recorded, right?"

Chaired by Andrew Adams-Moran, Galois

DEFUN and CUFP feedback

Simon Peyton Jones encouraged all participants to fill in the feedback forms for the three days of CUFP and DEFUN, as well as signing up for the google group:

Is a more formal organisation possible?

There was discussion about whether it was time for some more formal organisation: a birds of a feather group, sponsorship for organisation of various kinds? There's a standing CUFP page, and google group http://groups.google.com/group/cufp already, and this could provide a focus for

- listings of users of functional programming;
- listings of consultants offering functional programming skills;
- job opportunities in functional programming (complementing existing lists on e,g, the Haskell website);
- incorporating Functional Programming in the Real World? updates for this to Phil Wadler please;
- also discussed below in discussion of standardising libraries.

Standardising libraries?

There are different pressures on commercial users than researchers: Open Source software moves too fast (examples in the talks today). Would this be the right forum to organise more stable libraries?

Initiatives like hackage and the Haskell Platform (regular releases of a set of libraries for Haskell every six months) may well put on a brake on the speed of change, and make it easier for commercial users, as well as tool builders and writers of books on functional programming.

Agreement with the “blessing body” approach. e.g. clients can be happy with Apache blessed code, when they might be worried about other producers. Size of the company can be a problem. This could be a company or a community organisation …

ETC has its own distribution of Erlang on its “to do” list e.g. database drivers. Will also provide commercial support.

Linux-style operation (red hat etc)? Paying for the software could prove difficult in larger companies when doing things in “under the radar” approach; would potentially kill innovation. Though there’s a difference between trying out and distributing products which actually use FP in anger.

What are the barriers stopping people from contributing to open source libraries?

It would be useful to have some way of users indicating where changes and additions are needed … on the other hand, perhaps people just have to roll up their sleeves and do it. Is there beginning to be an opportunity for “red hat for functional programming”, to raise cash from this?

Time to write “non glory” stuff: e.g. numerical libraries. Can an organisation provide some sort of incentive for doing this? Pat on the back, at worst. “Well Typed” volunteers for this kind of work: call for sponsorship.

Driving adoption of FP?

First of those issues: F# is a great example. Are we all using the wrong language? Once it’s in Visual Studio, it will just happen. The F# group have also been active in producing exciting examples and videos. They also get the advantage of being in .NET: a 10 line program can access existing libraries.

Videos on line from this workshop will be a great help, if they can be away from ACM Digital Library. They were also on Google Video last year.

It was agreed that getting a whole lot of 10 minute videos out there would make a huge difference. There’s also a need for training courses for people wanting to dip their toes into the water, as well as more specialised courses.

How to hire programmers?

Make universities teach this … can do this by getting involved with local universities: given them interesting examples and projects.

Have IT recruitment sites include functional languages in tick lists, both individually and as a catch-all category.

SIGPLAN workshop on programming languages in the curriculum asked for 10 hours of functional programming instruction; 95% of comments wildly positive. Curriculum committee didn't adopt in its entirety: asked for at least 2 programming paradigms: functional or scripting languages. ACM and SIGPLAN members need to keep up the pressure here.

Issue about how to best write CVs: make sure that say "functional programming" as well as the particular language.

Hiring OCaml programmers is an "idiot filter": just get the good ones …