F#

Making CaML fit with .NET

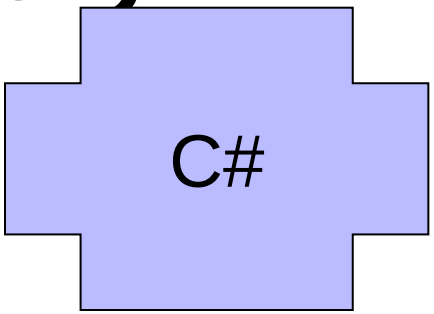
Don Syme

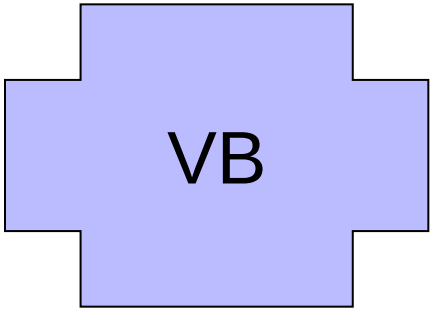
MSR Cambridge, October 2004

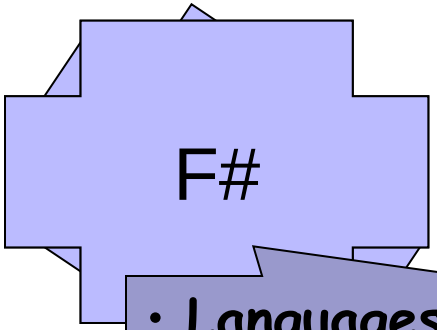
Topics for today (October 2004)

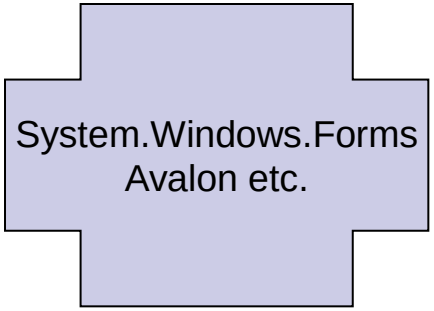
- What I would like people to think when they hear "F#"
- Why I did F# at a research lab
- F# status and future
- F#'s relation to ongoing research
- F# communication strategy

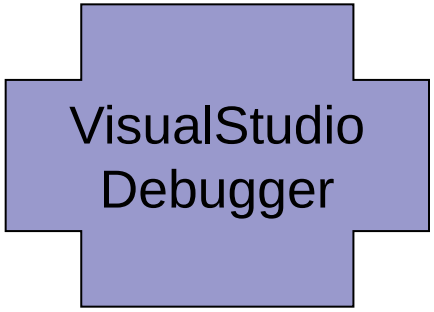
F# = ML as an equal player (October 2004)

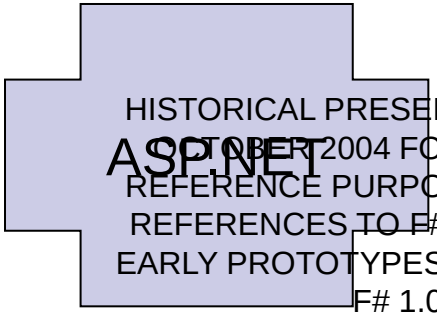












- Languages are far less dominant;
- A language is judged on its connectivity;
- Loss of dominance => some loss of control

HISTORICAL PRESENTATION FROM OCTOBER 2004 FOR ACADEMIC REFERENCE PURPOSES ONLY. ALL REFERENCES TO F# ARE TO VERY EARLY PROTOTYPES WELL BEFORE F# 1.0

When a C# programmer meets F#? (October 2004)

- "Mixed functional/imperative"
- "An ML that fits with .NET"
- "Binary components that have the same properties as C# components"
  - Don't really care what language the component was implemented in
  - Binary compatibility, versionable, profile, debug
- "Uncomplicated, reliable and cooperative"

When an OCaml programmer meets F#? (October 2004)

- "Can cross-compile applications as Core OCaml and F#"
- "Can migrate applications from Full OCaml to F#"
- "Works with .NET"
- "Tools are familiar"
- "Re-implementation, not port"
  - Note: F# is heavily inspired by OCaml. I always acknowledge and respect that.

What's the win (of ML over C#)? (October 2004)

- Terseness that works
- Inner definitions
- Type inference
- Tuples, lists
- Discriminated unions
- Functions as first-class values
- Simple but powerful optimization story
- Mutual dependencies/recursion explicit
- Immutability the norm
- However the same "basic model", e.g. w.r.t I/O, effects, exceptions

F# v. C# Examples (October 2004)

- Example: Object expressions

- Note, design takes care, e.g.:
  - Why not just add classes? (much is lost)
  - How about "this/self"? (see "reactive recursion")
  - Can object expressions be nested inside object expressions? (of course)
  - How about "base"? (`{ new X as base with … }`)

What's the win (of ML over C#)? (October 2004)

- Less is more
  - Only a handful of techniques to learn
  - Namespaces? Sealed? Private? Family? Readonly? Class Constructors? Constructors? Statics? Structs? Interface? Enums? Sheesh…

What's the win (of F# over ML)? (October 2004)

- Connectedness
  - Libraries galore
  - Tools
  - Bi-directional interop with C#, VB etc.
  - Bi-directional interop with C/COM (Wrap using C#)
  - Can build DLLs, services
  - Multi-threading that works
  - Versioning that works
  - No significant runtime components

F# Language Extensions (October 2004)

- Added to Core ML:
  - Interoperability constructs
    - Access properties, methods, fields, statics, instance, generics, delegates
    - Gave in to the power of the "."
  - Object expressions
    - Similar to inner classes, but much simpler
  - "Reactive Recursion"
    - For mutually referential objects

F# Language Restrictions (October 2004)

- Removed:
  - OCaml-style "objects"
  - Higher-kinded polymorphism
  - Modules-as-values
  - Some loss of performance (2-4x from OCaml native)
- Minor Changes from Core OCaml, e.g.
  - Strings are unicode

Why did I do F# at a MSRC? (October 2004)

- In a way it's what I had in mind since Dec 1999
  - Generics, ILX, Project-7, GHC.NET etc.
  - Wanted a "crisp" implementation that I would like to use
- Groundwork for delivery of research
  - Believe it's a feasible ship vehicle for type systems
  - C# is full
- Aside: an early goal was testing generics
  - Was a great way to get oodles of generic code
  - Many contributions, including perf refinements

Is F# Research? (October 2004)

- In many ways not – so far no papers on F# alone
- Actual:
  - "ILX: Extending the .NET Common IL for Functional Language Interoperability" (2001)
- Likely:
  - "Typed Compilation for Cross-Language Interoperability"
  - "ML as a .NET Language", c.f. other attempts
  - "ML code as components: initialization semantics & initialization granularities"
  - "Mutually dependent 'reactive' objects in strict languages"

Future F# Research? (October 2004)

- Using .NET libraries raises research issues
  - Modules and existential types
  - Type systems for concurrency (and other effects-related issues)
- Visual programming opportunities
  - Visual editing of embedded languages

F# Status (October 2004)

- Version "1.0" released externally yesterday
  - Limited commercial use license (must statically link the F# libraries, lots of curses and warnings)
  - Very stable core language and compiler
  - VS integration
  - ML compatibility library
  - Samples
  - Tools (Lexer, LALR)
- Some users, but not a stifling number yet:
  - 6000 downloads/year, nearly all playing
  - Dominic Cooney: Office XAP prototyping
  - Byron/MSR/SDV: Zap proof component of SDV
  - Jakob LichtenWindows: SDV (hopefully)
  - Karthik: TulaFale

F# Observations

- Surprisingly F# appears to be a better **client** of the .NET libraries than a language for **authoring** .NET libraries, i.e.
  - excellent for using .NET libraries
  - excellent for writing ML libraries
  - Can use ML libraries as .NET libraries, but could be better
- So the niche seems to be for writing sophisticated applications
  - probably making use of the .NET components
  - probably with a symbolic processing component
  - probably with some components written in C# etc.
  - probably with some high-value components written in F# and "spun-off" along the way

Questions?

HISTORICAL PRESENTATION FROM OCTOBER 2004 FOR ACADEMIC REFERENCE PURPOSES ONLY. ALL REFERENCES TO F# ARE TO VERY EARLY PROTOTYPES WELL BEFORE F# 1.0

What's the loss (of ML over C#)?

- Non-coercive subtyping
- Library Design doesn't work so well in ML