The F# Computation Expression Zoo

Tomas Petricek, University of Cambridge

Examples: http://tryjoinads.org/computations

Get in touch: @tomaspetricek | tomas@tomasp.net

software stacks

trainings teaching F# user groups snippets

mac and linux cross-platform tutorials

F# Software Foundation

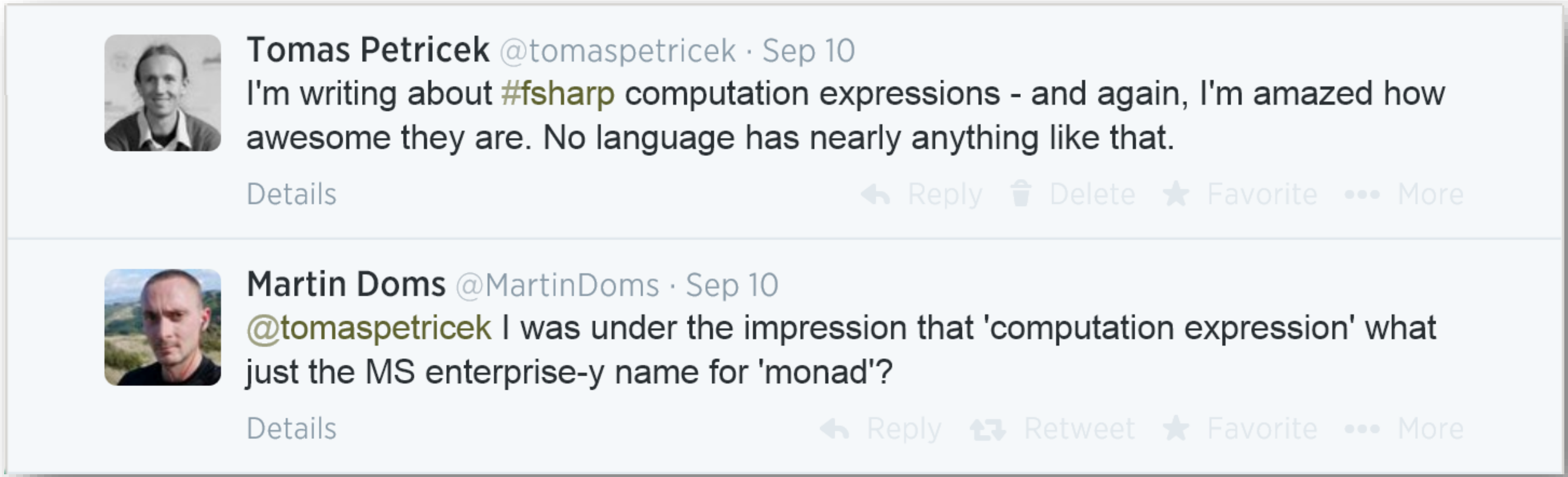
F# community open-source MonoDevelop

http://www.fsharp.org

contributions research support

consultancy Emacs and vim

What are computation expressions?



Tomas Petricek @tomaspetricek · Sep 10
I'm writing about #fsharp computation expressions - and again, I'm amazed how awesome they are. No language has nearly anything like that.

Martin Doms @MartinDoms · Sep 10
@tomaspetricek I was under the impression that 'computation expression' what just the MS enterprise-y name for 'monad'?

Syntax for non-standard computations

Haskell do, Python generators, C# async

```haskell
twiceState :: State Integer ()
twiceState = do
    x ← get
    set (x * 2)
```

```csharp
async Task<string> GetLength(string url) {
    var html = await DownloadAsync(url);
    return html.Length;
}
```

```python
def duplicate(inputs):
    for number in inputs:
        yield number
        yield number * 10
```

Language Feature	Nice syntax (await, yield) Just one use case
General Purpose	Many different uses May not be the best fit
Computation Expressions	Reusable but flexible syntax Library author can decide!

Computation expressions

Asynchronous workflows

Monadic computations

Bind	:	$M\alpha \to (\alpha \to M\beta) \to M\beta$
Return	:	$\alpha \to M\alpha$
Combine	:	$M\,\text{unit} \to M\alpha \to M\alpha$
Zero	:	$M\,\text{unit}$

Combine means sequencing

More operations enable more syntax (**for**, **while**, exception handling)

Syntax for additive computations

Parsers and sequence expressions

Additive computations

Bind	$: M\alpha \to (\alpha \to M\beta) \to M\beta$
Return	$: \alpha \to M\alpha$
Combine	$: M\alpha \to M\alpha \to M\alpha$
Zero	$: M\alpha$

Monoid structure: **MonadPlus** or **MonadOr**

Get a nice syntax if you have them!
Choose the right one (**yield** vs. **return**)

Composed computations

Using asynchronous sequences

Composed computations

```
type AsyncSeq<α> = Async<AsyncRes<α>>
type AsyncRes<α> = Nil | Cons of α * AsyncSeq<α>
```

For	:	$AS\ \alpha \rightarrow (\alpha \rightarrow AS\ \beta) \rightarrow AS\ \beta$
Bind	:	$A\ \alpha \rightarrow (\alpha \rightarrow AS\ \beta) \rightarrow AS\ \beta$

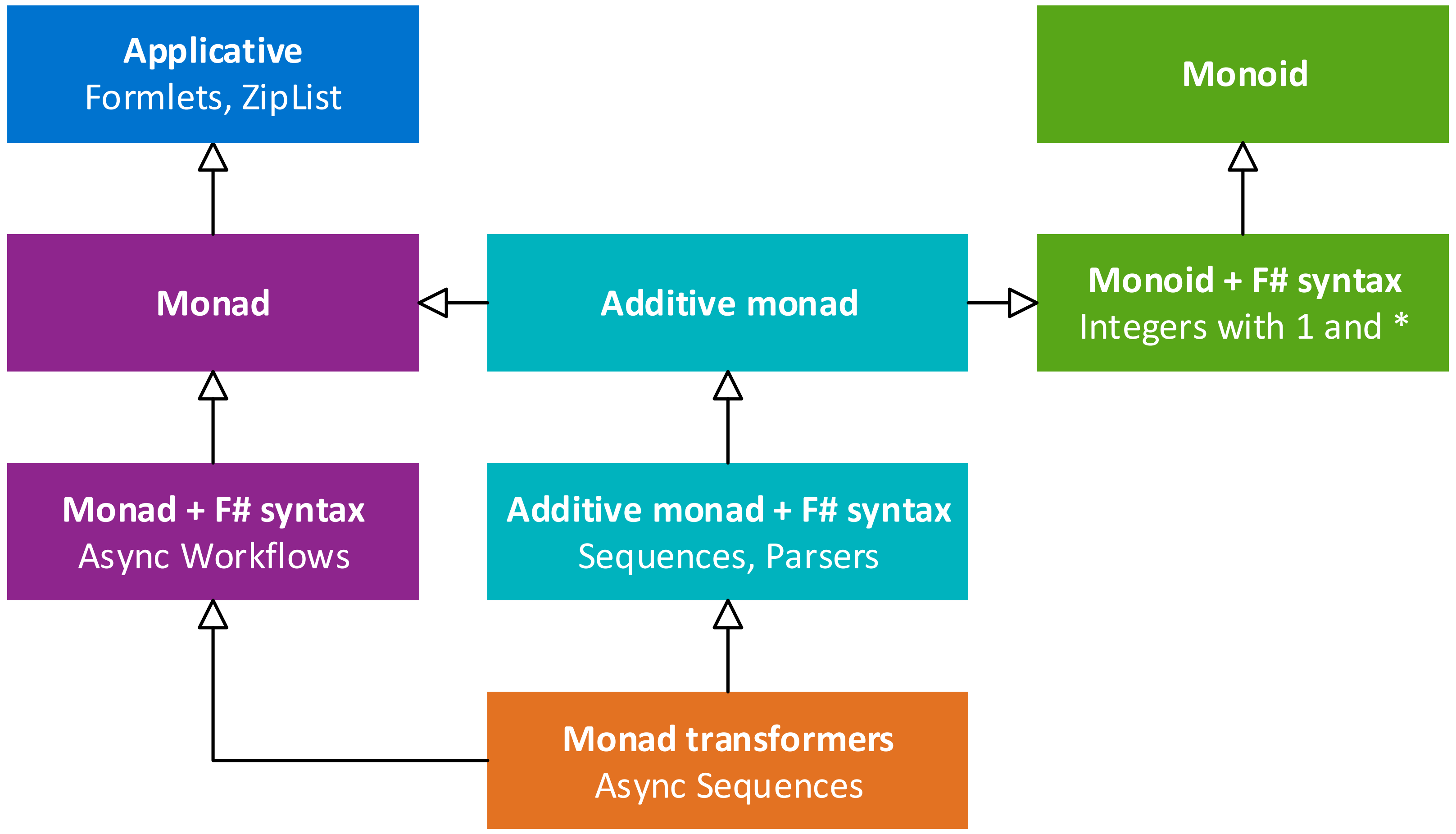
Syntax for monad transformers!

Let library author choose the notation
(define **For**, **Bind**, **Yield**, **Return** operations)

Summary

Why computation expressions?

What can you express?



Syntax matters!

Reuse standard keywords
Let library author decide

Examples & source: http://tryjoinads.org/computations

Paper: http://tomasp.net/academic/papers/computation-zoo

Get in touch: @tomaspetricek | tomas@tomasp.net